

# Algorithmische Graphentheorie

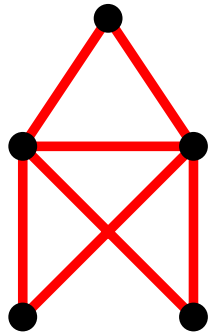
Sommersemester 2023

3. Vorlesung, Teil A

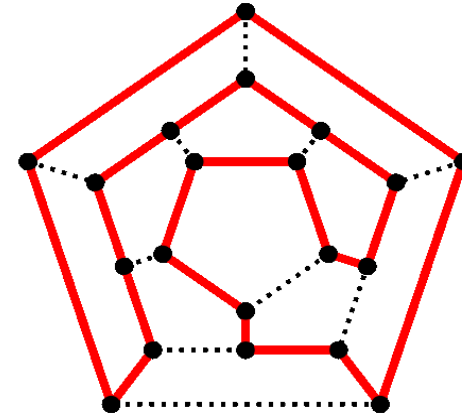
## Rundreiseprobleme – Teil III

# Übersicht

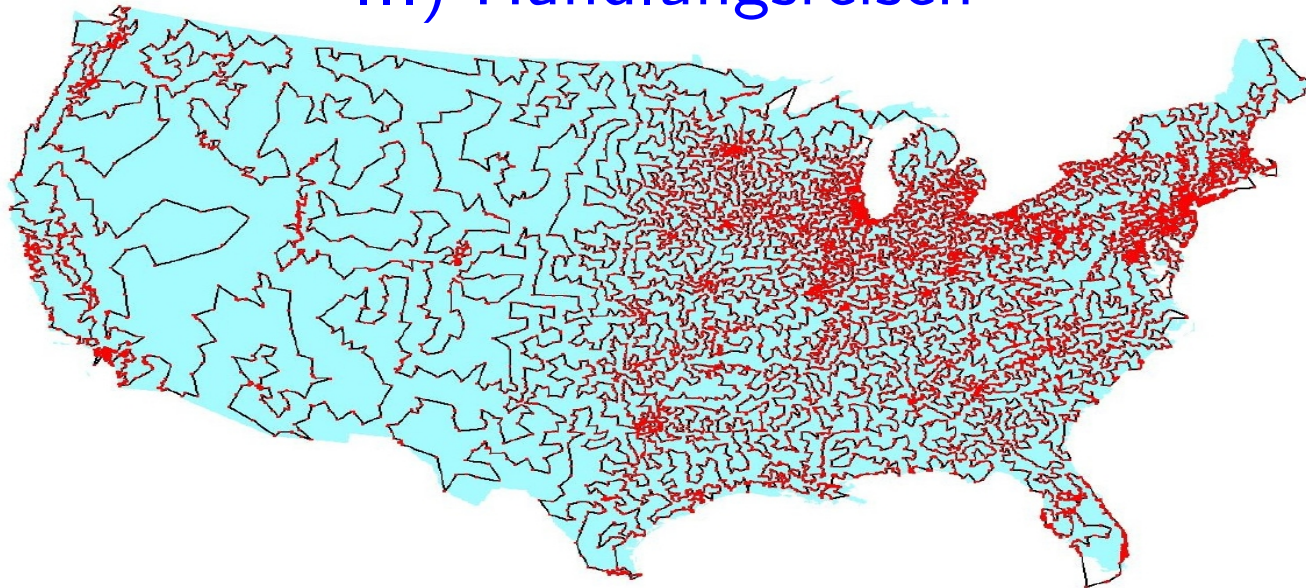
## I) Eulerkreise



## II) Hamiltonkreise



## III) Handlungsreisen



## III) Handlungsreisen

**Problem:** *Traveling Salesman/Salesperson Problem (TSP)*

Gegeben: unger. vollständiger Graph  $G = (V, E)$   
mit Kantenkosten  $c: E \rightarrow \mathbb{R}_{\geq 0}$

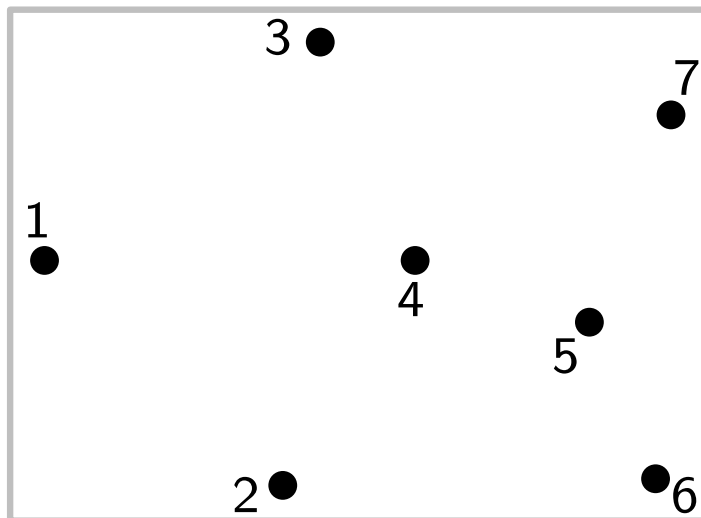
# III) Handlungsreisen

**Problem:** *Traveling Salesman/Salesperson Problem (TSP)*

Gegeben: unger. vollständiger Graph  $G = (V, E)$   
mit Kantenkosten  $c: E \rightarrow \mathbb{R}_{\geq 0}$

**Beispiel.**

$c \equiv d_{\text{Eukl.}}$



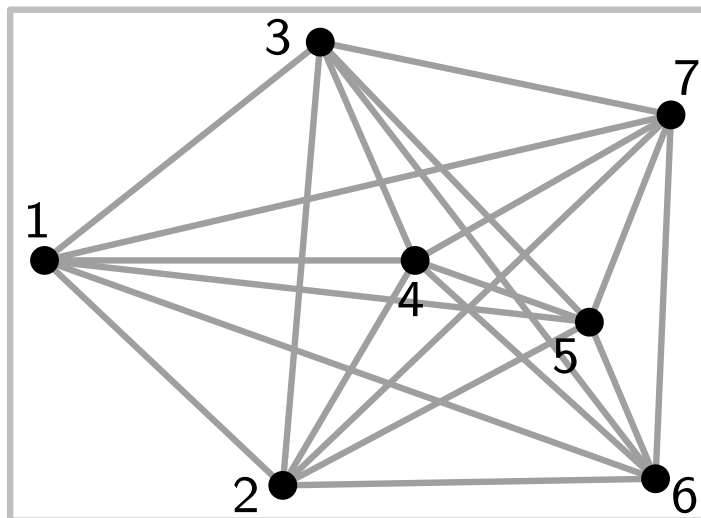
# III) Handlungsreisen

**Problem:** *Traveling Salesman/Salesperson Problem (TSP)*

Gegeben: unger. vollständiger Graph  $G = (V, E)$   
mit Kantenkosten  $c: E \rightarrow \mathbb{R}_{\geq 0}$

**Beispiel.**

$c \equiv d_{\text{Eukl.}}$



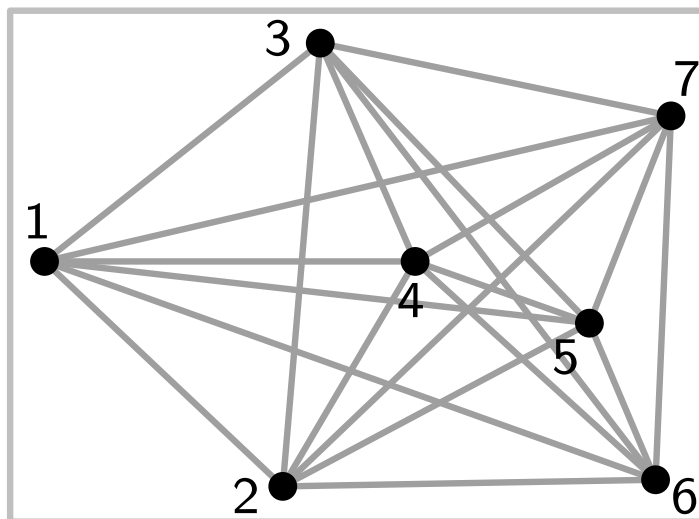
# III) Handlungsreisen

**Problem:** *Traveling Salesman/Salesperson Problem (TSP)*

Gegeben: unger. vollständiger Graph  $G = (V, E)$   
mit Kantenkosten  $c: E \rightarrow \mathbb{R}_{\geq 0}$

**Beispiel.**

$c \equiv d_{\text{Eukl.}}$



$$\begin{pmatrix} 0 & c_{12} & c_{13} & c_{14} & c_{15} & c_{16} & c_{17} \\ c_{21} & 0 & c_{23} & c_{24} & c_{25} & c_{26} & c_{27} \\ c_{31} & c_{32} & 0 & c_{34} & c_{35} & c_{36} & c_{37} \\ c_{41} & c_{42} & c_{43} & 0 & c_{45} & c_{46} & c_{47} \\ c_{51} & c_{52} & c_{53} & c_{54} & 0 & c_{56} & c_{57} \\ c_{61} & c_{62} & c_{63} & c_{64} & c_{65} & 0 & c_{67} \\ c_{71} & c_{72} & c_{73} & c_{74} & c_{75} & c_{76} & 0 \end{pmatrix}$$

# III) Handlungsreisen

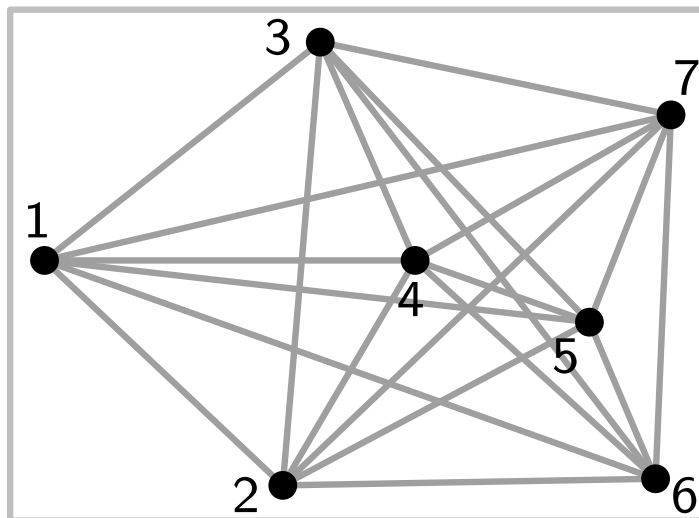
**Problem:** *Traveling Salesman/Salesperson Problem (TSP)*

Gegeben: unger. vollständiger Graph  $G = (V, E)$   
mit Kantenkosten  $c: E \rightarrow \mathbb{R}_{\geq 0}$

Gesucht: Hamiltonkreis  $K$  in  $G$  mit minimalen  
Kosten  $c(K)$ .

**Beispiel.**

$c \equiv d_{\text{Eukl.}}$



$$\begin{pmatrix} 0 & c_{12} & c_{13} & c_{14} & c_{15} & c_{16} & c_{17} \\ c_{21} & 0 & c_{23} & c_{24} & c_{25} & c_{26} & c_{27} \\ c_{31} & c_{32} & 0 & c_{34} & c_{35} & c_{36} & c_{37} \\ c_{41} & c_{42} & c_{43} & 0 & c_{45} & c_{46} & c_{47} \\ c_{51} & c_{52} & c_{53} & c_{54} & 0 & c_{56} & c_{57} \\ c_{61} & c_{62} & c_{63} & c_{64} & c_{65} & 0 & c_{67} \\ c_{71} & c_{72} & c_{73} & c_{74} & c_{75} & c_{76} & 0 \end{pmatrix}$$

# III) Handlungsreisen

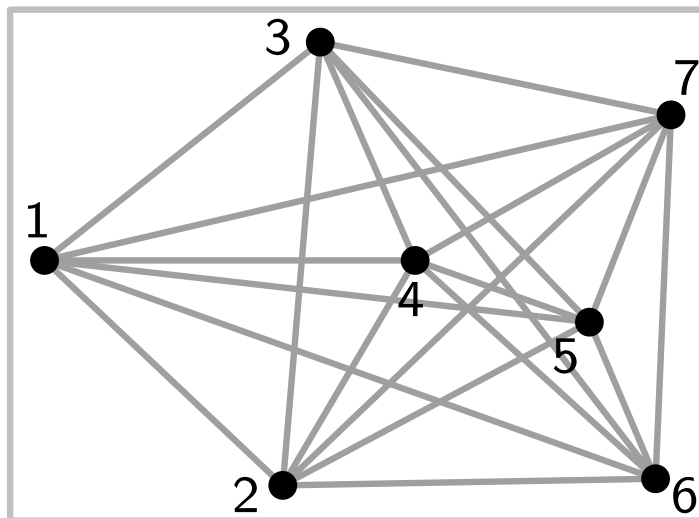
**Problem:** *Traveling Salesman/Salesperson Problem (TSP)*

Gegeben: unger. vollständiger Graph  $G = (V, E)$   
mit Kantenkosten  $c: E \rightarrow \mathbb{R}_{\geq 0}$

Gesucht: Hamiltonkreis  $K$  in  $G$  mit minimalen  
Kosten  $c(K) := \sum_{e \in K} c(e)$ .

**Beispiel.**

$c \equiv d_{\text{Eukl.}}$



$$\begin{pmatrix} 0 & c_{12} & c_{13} & c_{14} & c_{15} & c_{16} & c_{17} \\ c_{21} & 0 & c_{23} & c_{24} & c_{25} & c_{26} & c_{27} \\ c_{31} & c_{32} & 0 & c_{34} & c_{35} & c_{36} & c_{37} \\ c_{41} & c_{42} & c_{43} & 0 & c_{45} & c_{46} & c_{47} \\ c_{51} & c_{52} & c_{53} & c_{54} & 0 & c_{56} & c_{57} \\ c_{61} & c_{62} & c_{63} & c_{64} & c_{65} & 0 & c_{67} \\ c_{71} & c_{72} & c_{73} & c_{74} & c_{75} & c_{76} & 0 \end{pmatrix}$$



# III) Handlungsreisen

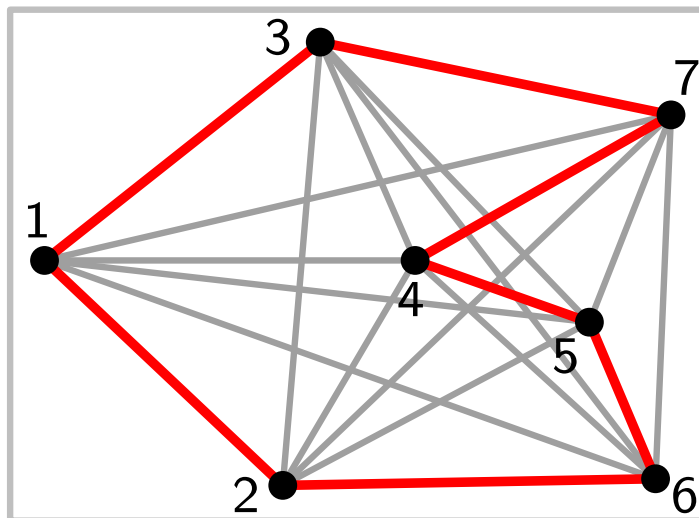
**Problem:** *Traveling Salesman/Salesperson Problem (TSP)*

Gegeben: unger. vollständiger Graph  $G = (V, E)$   
mit Kantenkosten  $c: E \rightarrow \mathbb{R}_{\geq 0}$

Gesucht: Hamiltonkreis  $K$  in  $G$  mit minimalen  
Kosten  $c(K) := \sum_{e \in K} c(e)$ .

**Beispiel.**

$c \equiv d_{\text{Eukl.}}$



$$\begin{pmatrix} 0 & c_{12} & c_{13} & c_{14} & c_{15} & c_{16} & c_{17} \\ c_{21} & 0 & c_{23} & c_{24} & c_{25} & c_{26} & c_{27} \\ c_{31} & c_{32} & 0 & c_{34} & c_{35} & c_{36} & c_{37} \\ c_{41} & c_{42} & c_{43} & 0 & c_{45} & c_{46} & c_{47} \\ c_{51} & c_{52} & c_{53} & c_{54} & 0 & c_{56} & c_{57} \\ c_{61} & c_{62} & c_{63} & c_{64} & c_{65} & 0 & c_{67} \\ c_{71} & c_{72} & c_{73} & c_{74} & c_{75} & c_{76} & 0 \end{pmatrix}$$

# III) Handlungsreisen

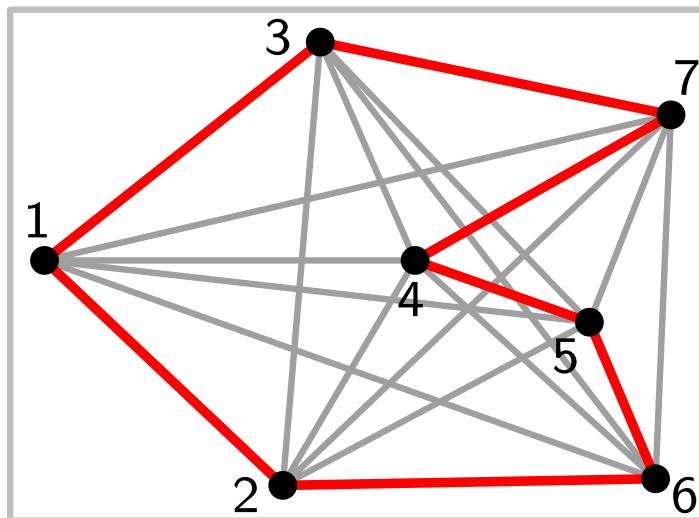
**Problem:** *Traveling Salesman/Salesperson Problem (TSP)*

Gegeben: unger. vollständiger Graph  $G = (V, E)$   
mit Kantenkosten  $c: E \rightarrow \mathbb{R}_{\geq 0}$

Gesucht: Hamiltonkreis  $K$  in  $G$  mit minimalen  
Kosten  $c(K) := \sum_{e \in K} c(e)$ .

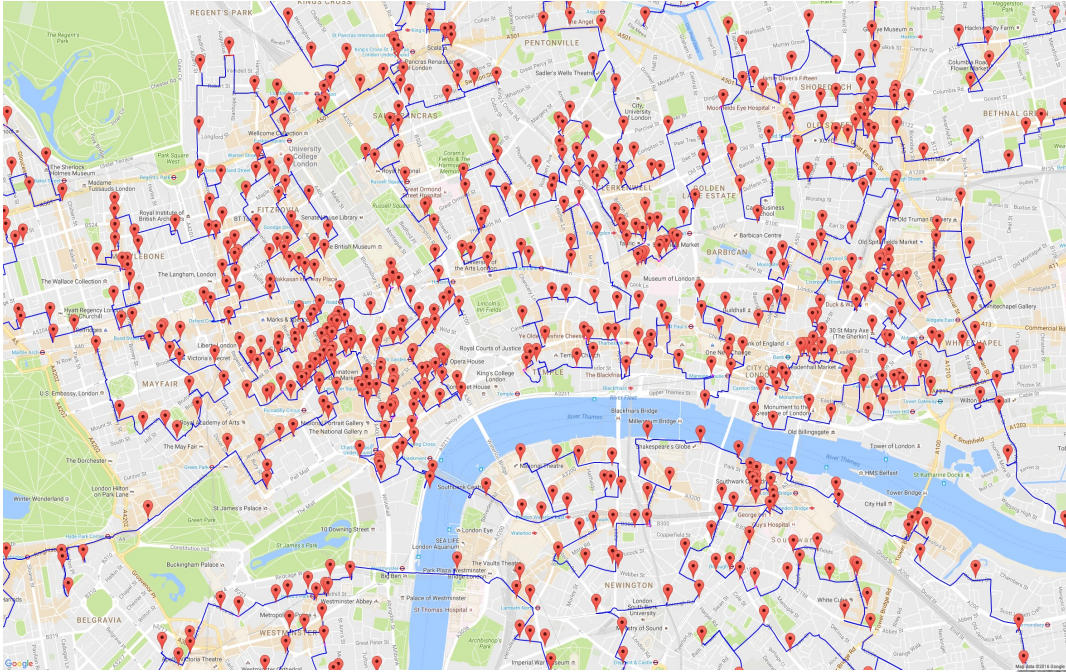
**Beispiel.**

$c \equiv d_{\text{Eukl.}}$



0	$c_{12}$	$c_{13}$	$c_{14}$	$c_{15}$	$c_{16}$	$c_{17}$
$c_{21}$	0	$c_{23}$	$c_{24}$	$c_{25}$	$c_{26}$	$c_{27}$
$c_{31}$	$c_{32}$	0	$c_{34}$	$c_{35}$	$c_{36}$	$c_{37}$
$c_{41}$	$c_{42}$	$c_{43}$	0	$c_{45}$	$c_{46}$	$c_{47}$
$c_{51}$	$c_{52}$	$c_{53}$	$c_{54}$	0	$c_{56}$	$c_{57}$
$c_{61}$	$c_{62}$	$c_{63}$	$c_{64}$	$c_{65}$	0	$c_{67}$
$c_{71}$	$c_{72}$	$c_{73}$	$c_{74}$	$c_{75}$	$c_{76}$	0

# Beispielinstanzen



Alle 24.727 Pubs in Großbritannien.  
(45.495,239 km)



Mona Lisa TSP Challenge.

# Exakte Berechnung: Brute Force

**Algorithmus:** • Für jede Permutation  $\sigma$  von  $\langle 1, 2, \dots, n \rangle$ :

# Exakte Berechnung: Brute Force

- Algorithmus:**
- Für jede Permutation  $\sigma$  von  $\langle 1, 2, \dots, n \rangle$ :  
Berechne die Kosten der Tour durch die Knoten  $v_1, \dots, v_n$  in dieser Reihenfolge:  
$$c(\sigma) = c(v_{\sigma(n)}v_{\sigma(1)}) + \sum_{i=1}^{n-1} c(v_{\sigma(i)}v_{\sigma(i+1)})$$

# Exakte Berechnung: Brute Force

## Algorithmus:

- Für jede Permutation  $\sigma$  von  $\langle 1, 2, \dots, n \rangle$ :  
Berechne die Kosten der Tour durch die Knoten  $v_1, \dots, v_n$  in dieser Reihenfolge:  
$$c(\sigma) = c(v_{\sigma(n)}v_{\sigma(1)}) + \sum_{i=1}^{n-1} c(v_{\sigma(i)}v_{\sigma(i+1)})$$
- Gib die kürzeste Tour zurück.

# Exakte Berechnung: Brute Force

## Algorithmus:

- Für jede Permutation  $\sigma$  von  $\langle 1, 2, \dots, n \rangle$ :  
Berechne die Kosten der Tour durch die Knoten  $v_1, \dots, v_n$  in dieser Reihenfolge:  
$$c(\sigma) = c(v_{\sigma(n)}v_{\sigma(1)}) + \sum_{i=1}^{n-1} c(v_{\sigma(i)}v_{\sigma(i+1)})$$
- Gib die kürzeste Tour zurück.

## Laufzeit:

# Exakte Berechnung: Brute Force

- Algorithmus:**
- Für jede Permutation  $\sigma$  von  $\langle 1, 2, \dots, n \rangle$ :  
Berechne die Kosten der Tour durch die Knoten  $v_1, \dots, v_n$  in dieser Reihenfolge:  
$$c(\sigma) = c(v_{\sigma(n)}v_{\sigma(1)}) + \sum_{i=1}^{n-1} c(v_{\sigma(i)}v_{\sigma(i+1)})$$
  - Gib die kürzeste Tour zurück.

**Laufzeit:** Anzahl Permutationen von  $n$  Objekten:



# Exakte Berechnung: Brute Force

- Algorithmus:**
- Für jede Permutation  $\sigma$  von  $\langle 1, 2, \dots, n \rangle$ :  
Berechne die Kosten der Tour durch die Knoten  $v_1, \dots, v_n$  in dieser Reihenfolge:  
$$c(\sigma) = c(v_{\sigma(n)}v_{\sigma(1)}) + \sum_{i=1}^{n-1} c(v_{\sigma(i)}v_{\sigma(i+1)})$$
  - Gib die kürzeste Tour zurück.
- Laufzeit:** Anzahl Permutationen von  $n$  Objekten:  $n!$

# Exakte Berechnung: Brute Force

## Algorithmus:

- Für jede Permutation  $\sigma$  von  $\langle 1, 2, \dots, n \rangle$ :  
Berechne die Kosten der Tour durch die Knoten  $v_1, \dots, v_n$  in dieser Reihenfolge:  
$$c(\sigma) = c(v_{\sigma(n)}v_{\sigma(1)}) + \sum_{i=1}^{n-1} c(v_{\sigma(i)}v_{\sigma(i+1)})$$
- Gib die kürzeste Tour zurück.

## Laufzeit:

Anzahl Permutationen von  $n$  Objekten:  $n!$

Hält man den 1. Knoten fest, so bleiben „nur“  $(n - 1)!$  Permutationen.

# Exakte Berechnung: Brute Force

- Algorithmus:**
- Für jede Permutation  $\sigma$  von  $\langle 1, 2, \dots, n \rangle$ :  
 Berechne die Kosten der Tour durch die Knoten  $v_1, \dots, v_n$  in dieser Reihenfolge:  

$$c(\sigma) = c(v_{\sigma(n)}v_{\sigma(1)}) + \sum_{i=1}^{n-1} c(v_{\sigma(i)}v_{\sigma(i+1)})$$
  - Gib die kürzeste Tour zurück.

**Laufzeit:** Anzahl Permutationen von  $n$  Objekten:  $n!$

Hält man den 1. Knoten fest, so bleiben „nur“  $(n - 1)!$  Permutationen.

Berechnung einer Tourlänge  $c(\sigma)$ :

# Exakte Berechnung: Brute Force

- Algorithmus:**
- Für jede Permutation  $\sigma$  von  $\langle 1, 2, \dots, n \rangle$ :  
Berechne die Kosten der Tour durch die Knoten  $v_1, \dots, v_n$  in dieser Reihenfolge:  
$$c(\sigma) = c(v_{\sigma(n)}v_{\sigma(1)}) + \sum_{i=1}^{n-1} c(v_{\sigma(i)}v_{\sigma(i+1)})$$
  - Gib die kürzeste Tour zurück.

- Laufzeit:**
- Anzahl Permutationen von  $n$  Objekten:  $n!$
- Hält man den 1. Knoten fest, so bleiben „nur“  $(n - 1)!$  Permutationen.
- Berechnung einer Tourlänge  $c(\sigma)$ :  $O(n)$  Zeit.

# Exakte Berechnung: Brute Force

- Algorithmus:**
- Für jede Permutation  $\sigma$  von  $\langle 1, 2, \dots, n \rangle$ :  
 Berechne die Kosten der Tour durch die Knoten  $v_1, \dots, v_n$  in dieser Reihenfolge:  

$$c(\sigma) = c(v_{\sigma(n)}v_{\sigma(1)}) + \sum_{i=1}^{n-1} c(v_{\sigma(i)}v_{\sigma(i+1)})$$
  - Gib die kürzeste Tour zurück.

- Laufzeit:**
- Anzahl Permutationen von  $n$  Objekten:  $n!$
- Hält man den 1. Knoten fest, so bleiben „nur“  $(n - 1)!$  Permutationen.
- Berechnung einer Tourlänge  $c(\sigma)$ :  $O(n)$  Zeit.
- Berechnung der nächsten Permutation:

# Exakte Berechnung: Brute Force

- Algorithmus:**
- Für jede Permutation  $\sigma$  von  $\langle 1, 2, \dots, n \rangle$ :  
 Berechne die Kosten der Tour durch die Knoten  $v_1, \dots, v_n$  in dieser Reihenfolge:  


$$c(\sigma) = c(v_{\sigma(n)}v_{\sigma(1)}) + \sum_{i=1}^{n-1} c(v_{\sigma(i)}v_{\sigma(i+1)})$$
  - Gib die kürzeste Tour zurück.

- Laufzeit:**
- Anzahl Permutationen von  $n$  Objekten:  $n!$
- Hält man den 1. Knoten fest, so bleiben „nur“  $(n - 1)!$  Permutationen.
- Berechnung einer Tourlänge  $c(\sigma)$ :  $O(n)$  Zeit.
- Berechnung der nächsten Permutation: ???

# Exakte Berechnung: Brute Force

- Algorithmus:**
- Für jede Permutation  $\sigma$  von  $\langle 1, 2, \dots, n \rangle$ :  
 Berechne die Kosten der Tour durch die Knoten  $v_1, \dots, v_n$  in dieser Reihenfolge:  

$$c(\sigma) = c(v_{\sigma(n)}v_{\sigma(1)}) + \sum_{i=1}^{n-1} c(v_{\sigma(i)}v_{\sigma(i+1)})$$
  - Gib die kürzeste Tour zurück.

- Laufzeit:**
- Anzahl Permutationen von  $n$  Objekten:  $n!$
- Hält man den 1. Knoten fest, so bleiben „nur“  $(n - 1)!$  Permutationen.
- Berechnung einer Tourlänge  $c(\sigma)$ :  $O(n)$  Zeit.
- Berechnung der nächsten Permutation: ???
- Ang. ??? =  $O(n)$ , dann ist die Laufzeit 

# Exakte Berechnung: Brute Force

- Algorithmus:**
- Für jede Permutation  $\sigma$  von  $\langle 1, 2, \dots, n \rangle$ :  
 Berechne die Kosten der Tour durch die Knoten  $v_1, \dots, v_n$  in dieser Reihenfolge:  

$$c(\sigma) = c(v_{\sigma(n)}v_{\sigma(1)}) + \sum_{i=1}^{n-1} c(v_{\sigma(i)}v_{\sigma(i+1)})$$
  - Gib die kürzeste Tour zurück.

- Laufzeit:**
- Anzahl Permutationen von  $n$  Objekten:  $n!$
- Hält man den 1. Knoten fest, so bleiben „nur“  $(n - 1)!$  Permutationen.
- Berechnung einer Tourlänge  $c(\sigma)$ :  $O(n)$  Zeit.
- Berechnung der nächsten Permutation: ???
- Ang. ??? =  $O(n)$ , dann ist die Laufzeit  $O(n!)$ .



# Exakte Berechnung: Brute Force

- Algorithmus:**
- Für jede Permutation  $\sigma$  von  $\langle 1, 2, \dots, n \rangle$ :  
 Berechne die Kosten der Tour durch die Knoten  $v_1, \dots, v_n$  in dieser Reihenfolge:  

$$c(\sigma) = c(v_{\sigma(n)}v_{\sigma(1)}) + \sum_{i=1}^{n-1} c(v_{\sigma(i)}v_{\sigma(i+1)})$$
  - Gib die kürzeste Tour zurück.

- Laufzeit:**
- Anzahl Permutationen von  $n$  Objekten:  $n!$
- Hält man den 1. Knoten fest, so bleiben „nur“  $(n - 1)!$  Permutationen.
- Berechnung einer Tourlänge  $c(\sigma)$ :  $O(n)$  Zeit.
- Berechnung der nächsten Permutation: ???
- Ang. ??? =  $O(n)$ , dann ist die Laufzeit  $O(n!)$ .

**Speicher:**

# Exakte Berechnung: Brute Force

- Algorithmus:**
- Für jede Permutation  $\sigma$  von  $\langle 1, 2, \dots, n \rangle$ :  
 Berechne die Kosten der Tour durch die Knoten  $v_1, \dots, v_n$  in dieser Reihenfolge:  

$$c(\sigma) = c(v_{\sigma(n)}v_{\sigma(1)}) + \sum_{i=1}^{n-1} c(v_{\sigma(i)}v_{\sigma(i+1)})$$
  - Gib die kürzeste Tour zurück.

- Laufzeit:**
- Anzahl Permutationen von  $n$  Objekten:  $n!$
- Hält man den 1. Knoten fest, so bleiben „nur“  $(n - 1)!$  Permutationen.
- Berechnung einer Tourlänge  $c(\sigma)$ :  $O(n)$  Zeit.
- Berechnung der nächsten Permutation: ???
- Ang. ??? =  $O(n)$ , dann ist die Laufzeit  $O(n!)$ .

- Speicher:**  $O(n)$  für aktuelle und bisher beste Permutation.

# Wie iteriert man durch alle Permutationen?

Z.B. in lexikografischer Ordnung:

$\langle 1, 2, 3, 4, 5, 6 \rangle, \langle 1, 2, 3, 4, 6, 5 \rangle, \langle 1, 2, 3, 5, 4, 6 \rangle, \dots, \langle 6, 5, 4, 3, 2, 1 \rangle.$

# Wie iteriert man durch alle Permutationen?

Z.B. in lexikografischer Ordnung:

$\langle 1, 2, 3, 4, 5, 6 \rangle, \langle 1, 2, 3, 4, 6, 5 \rangle, \langle 1, 2, 3, 5, 4, 6 \rangle, \dots, \langle 6, 5, 4, 3, 2, 1 \rangle$ .

Für gegebene Permutation  $\sigma$  finde Nachfolger in  $O(n)$  Zeit:

# Wie iteriert man durch alle Permutationen?

Z.B. in lexikografischer Ordnung:

$\langle 1, 2, 3, 4, 5, 6 \rangle, \langle 1, 2, 3, 4, 6, 5 \rangle, \langle 1, 2, 3, 5, 4, 6 \rangle, \dots, \langle 6, 5, 4, 3, 2, 1 \rangle$ .

Für gegebene Permutation  $\sigma$  finde Nachfolger in  $O(n)$  Zeit:

- Bestimme größten Index  $i \in \{1, \dots, n-1\}$  mit  $\sigma(i) < \sigma(i+1)$ .

$\langle 1, 4, 3, 6, 5, 2 \rangle$

# Wie iteriert man durch alle Permutationen?


Z.B. in lexikografischer Ordnung:

$\langle 1, 2, 3, 4, 5, 6 \rangle, \langle 1, 2, 3, 4, 6, 5 \rangle, \langle 1, 2, 3, 5, 4, 6 \rangle, \dots, \langle 6, 5, 4, 3, 2, 1 \rangle$ .

Für gegebene Permutation  $\sigma$  finde Nachfolger in  $O(n)$  Zeit:

- Bestimme größten Index  $i \in \{1, \dots, n-1\}$  mit  $\sigma(i) < \sigma(i+1)$ .

$\langle 1, 4, 3, 6, 5, 2 \rangle$   
 $i$



# Wie iteriert man durch alle Permutationen?


Z.B. in lexikografischer Ordnung:

$\langle 1, 2, 3, 4, 5, 6 \rangle, \langle 1, 2, 3, 4, 6, 5 \rangle, \langle 1, 2, 3, 5, 4, 6 \rangle, \dots, \langle 6, 5, 4, 3, 2, 1 \rangle$ .

Für gegebene Permutation  $\sigma$  finde Nachfolger in  $O(n)$  Zeit:

- Bestimme größten Index  $i \in \{1, \dots, n-1\}$  mit  $\sigma(i) < \sigma(i+1)$ .
- Falls nicht existiert, fertig ( $\sigma =$  letzte Permutation).

$\langle 1, 4, 3, 6, 5, 2 \rangle$   
 $i$



# Wie iteriert man durch alle Permutationen?


Z.B. in lexikografischer Ordnung:

$\langle 1, 2, 3, 4, 5, 6 \rangle, \langle 1, 2, 3, 4, 6, 5 \rangle, \langle 1, 2, 3, 5, 4, 6 \rangle, \dots, \langle 6, 5, 4, 3, 2, 1 \rangle$ .

Für gegebene Permutation  $\sigma$  finde Nachfolger in  $O(n)$  Zeit:

- Bestimme größten Index  $i \in \{1, \dots, n-1\}$  mit  $\sigma(i) < \sigma(i+1)$ .
- Falls nicht existiert, fertig ( $\sigma =$  letzte Permutation).
- Sonst bestimme größten Index  $j$  mit  $\sigma(i) < \sigma(j)$ .

$\langle 1, 4, 3, 6, 5, 2 \rangle$   
 $i$





# Wie iteriert man durch alle Permutationen?


Z.B. in lexikografischer Ordnung:

$\langle 1, 2, 3, 4, 5, 6 \rangle, \langle 1, 2, 3, 4, 6, 5 \rangle, \langle 1, 2, 3, 5, 4, 6 \rangle, \dots, \langle 6, 5, 4, 3, 2, 1 \rangle.$

Für gegebene Permutation  $\sigma$  finde Nachfolger in  $O(n)$  Zeit:

- Bestimme größten Index  $i \in \{1, \dots, n-1\}$  mit  $\sigma(i) < \sigma(i+1)$ .
- Falls nicht existiert, fertig ( $\sigma =$  letzte Permutation).
- Sonst bestimme größten Index  $j$  mit  $\sigma(i) < \sigma(j)$ .

$\langle 1, 4, 3, 6, 5, 2 \rangle$   
 $\quad \quad \quad i \quad \quad j$



# Wie iteriert man durch alle Permutationen?


Z.B. in lexikografischer Ordnung:

$\langle 1, 2, 3, 4, 5, 6 \rangle, \langle 1, 2, 3, 4, 6, 5 \rangle, \langle 1, 2, 3, 5, 4, 6 \rangle, \dots, \langle 6, 5, 4, 3, 2, 1 \rangle.$

Für gegebene Permutation  $\sigma$  finde Nachfolger in  $O(n)$  Zeit:

- Bestimme größten Index  $i \in \{1, \dots, n-1\}$  mit  $\sigma(i) < \sigma(i+1)$ .
- Falls nicht existiert, fertig ( $\sigma =$  letzte Permutation).
- Sonst bestimme größten Index  $j$  mit  $\sigma(i) < \sigma(j)$ .

$\langle 1, 4, 3, 6, 5, 2 \rangle$   
 $\quad \quad \quad i \quad \quad j$



- Vertausche  $\sigma(i)$  und  $\sigma(j)$ .

# Wie iteriert man durch alle Permutationen?

Z.B. in lexikografischer Ordnung:

$\langle 1, 2, 3, 4, 5, 6 \rangle, \langle 1, 2, 3, 4, 6, 5 \rangle, \langle 1, 2, 3, 5, 4, 6 \rangle, \dots, \langle 6, 5, 4, 3, 2, 1 \rangle.$

Für gegebene Permutation  $\sigma$  finde Nachfolger in  $O(n)$  Zeit:

- Bestimme größten Index  $i \in \{1, \dots, n-1\}$  mit  $\sigma(i) < \sigma(i+1)$ .
- Falls nicht existiert, fertig ( $\sigma =$  letzte Permutation).
- Sonst bestimme größten Index  $j$  mit  $\sigma(i) < \sigma(j)$ .

$\langle 1, 4, 3, 6, 5, 2 \rangle$   $\longrightarrow$

$i$        $j$

- Vertausche  $\sigma(i)$  und  $\sigma(j)$ .

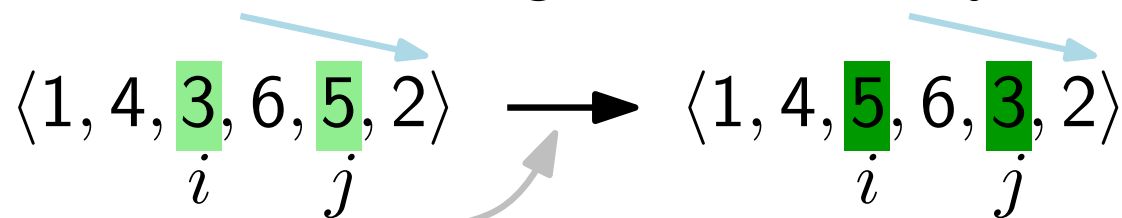
# Wie iteriert man durch alle Permutationen?

Z.B. in lexikografischer Ordnung:

$\langle 1, 2, 3, 4, 5, 6 \rangle, \langle 1, 2, 3, 4, 6, 5 \rangle, \langle 1, 2, 3, 5, 4, 6 \rangle, \dots, \langle 6, 5, 4, 3, 2, 1 \rangle$ .

Für gegebene Permutation  $\sigma$  finde Nachfolger in  $O(n)$  Zeit:

- Bestimme größten Index  $i \in \{1, \dots, n-1\}$  mit  $\sigma(i) < \sigma(i+1)$ .
- Falls nicht existiert, fertig ( $\sigma =$  letzte Permutation).
- Sonst bestimme größten Index  $j$  mit  $\sigma(i) < \sigma(j)$ .



- Vertausche  $\sigma(i)$  und  $\sigma(j)$ .

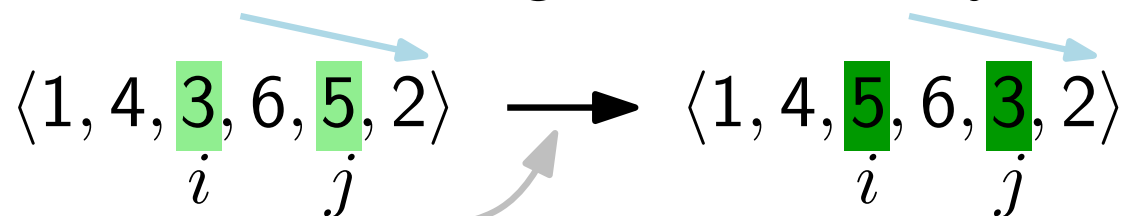
# Wie iteriert man durch alle Permutationen?

Z.B. in lexikografischer Ordnung:

$\langle 1, 2, 3, 4, 5, 6 \rangle, \langle 1, 2, 3, 4, 6, 5 \rangle, \langle 1, 2, 3, 5, 4, 6 \rangle, \dots, \langle 6, 5, 4, 3, 2, 1 \rangle$ .

Für gegebene Permutation  $\sigma$  finde Nachfolger in  $O(n)$  Zeit:

- Bestimme größten Index  $i \in \{1, \dots, n-1\}$  mit  $\sigma(i) < \sigma(i+1)$ .
- Falls nicht existiert, fertig ( $\sigma =$  letzte Permutation).
- Sonst bestimme größten Index  $j$  mit  $\sigma(i) < \sigma(j)$ .



- Vertausche  $\sigma(i)$  und  $\sigma(j)$ .
- Kehre die Teilfolge  $\langle \sigma(i+1), \sigma(i+2), \dots, \sigma(n) \rangle$  um.

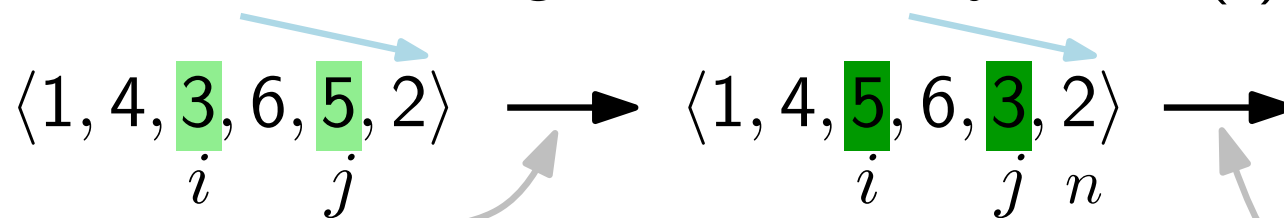
# Wie iteriert man durch alle Permutationen?

Z.B. in lexikografischer Ordnung:

$\langle 1, 2, 3, 4, 5, 6 \rangle, \langle 1, 2, 3, 4, 6, 5 \rangle, \langle 1, 2, 3, 5, 4, 6 \rangle, \dots, \langle 6, 5, 4, 3, 2, 1 \rangle$ .

Für gegebene Permutation  $\sigma$  finde Nachfolger in  $O(n)$  Zeit:

- Bestimme größten Index  $i \in \{1, \dots, n-1\}$  mit  $\sigma(i) < \sigma(i+1)$ .
- Falls nicht existiert, fertig ( $\sigma =$  letzte Permutation).
- Sonst bestimme größten Index  $j$  mit  $\sigma(i) < \sigma(j)$ .



- Vertausche  $\sigma(i)$  und  $\sigma(j)$ .
- Kehre die Teilfolge  $\langle \sigma(i+1), \sigma(i+2), \dots, \sigma(n) \rangle$  um.

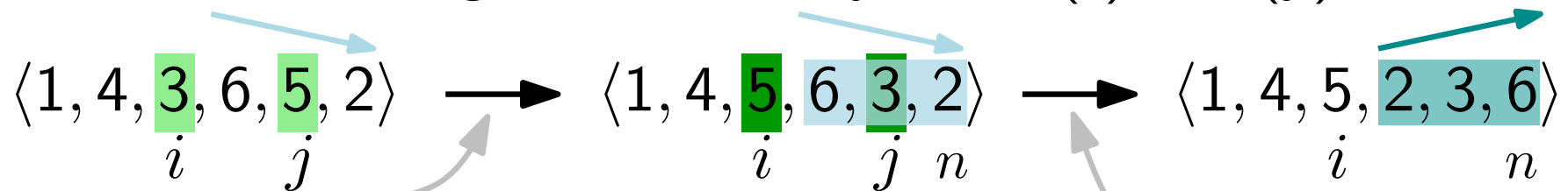
# Wie iteriert man durch alle Permutationen?

Z.B. in lexikografischer Ordnung:

$\langle 1, 2, 3, 4, 5, 6 \rangle, \langle 1, 2, 3, 4, 6, 5 \rangle, \langle 1, 2, 3, 5, 4, 6 \rangle, \dots, \langle 6, 5, 4, 3, 2, 1 \rangle$ .

Für gegebene Permutation  $\sigma$  finde Nachfolger in  $O(n)$  Zeit:

- Bestimme größten Index  $i \in \{1, \dots, n-1\}$  mit  $\sigma(i) < \sigma(i+1)$ .
- Falls nicht existiert, fertig ( $\sigma =$  letzte Permutation).
- Sonst bestimme größten Index  $j$  mit  $\sigma(i) < \sigma(j)$ .



- Vertausche  $\sigma(i)$  und  $\sigma(j)$ .
- Kehre die Teilfolge  $\langle \sigma(i+1), \sigma(i+2), \dots, \sigma(n) \rangle$  um.

Wie groß ist  $n!$  ?

$$n! = 1 \cdot 2 \cdot \dots \cdot n$$



Wie groß ist  $n!$  ?

$$\leq n! = 1 \cdot 2 \cdot \dots \cdot n \leq$$

Wie groß ist  $n!$  ?

$$\leq n! = 1 \cdot 2 \cdot \dots \cdot n \leq n \cdot n \cdot \dots \cdot n$$

Wie groß ist  $n!$  ?

$$\leq n! = 1 \cdot 2 \cdot \dots \cdot n \leq n \cdot n \cdot \dots \cdot n = n^n$$

Wie groß ist  $n!$  ?

$$n/2 \cdot n/2 \cdot \dots \cdot n/2 \leq n! = 1 \cdot 2 \cdot \dots \cdot n \leq n \cdot n \cdot \dots \cdot n = n^n$$

Wie groß ist  $n!$  ?

$$\underbrace{n/2 \cdot n/2 \cdot \dots \cdot n/2}_{n/2 \text{ mal}} \leq n! = 1 \cdot 2 \cdot \dots \cdot n \leq n \cdot n \cdot \dots \cdot n = n^n$$

# Wie groß ist $n!$ ?

$$\underbrace{n/2 \cdot n/2 \cdot \dots \cdot n/2}_{n/2 \text{ mal}} \leq n! = 1 \cdot 2 \cdot \dots \cdot n \leq n \cdot n \cdot \dots \cdot n = n^n$$

$$\Rightarrow n! \leq n^n =$$

# Wie groß ist $n!$ ?

$$\underbrace{n/2 \cdot n/2 \cdot \dots \cdot n/2}_{n/2 \text{ mal}} \leq n! = 1 \cdot 2 \cdot \dots \cdot n \leq n \cdot n \cdot \dots \cdot n = n^n$$

$$\Rightarrow n! \leq n^n = (2 \text{ [yellow box] })^{\text{[blue box]}}$$

# Wie groß ist $n!$ ?

$$\underbrace{n/2 \cdot n/2 \cdot \dots \cdot n/2}_{n/2 \text{ mal}} \leq n! = 1 \cdot 2 \cdot \dots \cdot n \leq n \cdot n \cdot \dots \cdot n = n^n$$

$$\Rightarrow n! \leq n^n = (2 \text{        })^n$$



# Wie groß ist $n!$ ?

$$\underbrace{n/2 \cdot n/2 \cdot \dots \cdot n/2}_{n/2 \text{ mal}} \leq n! = 1 \cdot 2 \cdot \dots \cdot n \leq n \cdot n \cdot \dots \cdot n = n^n$$

$$\Rightarrow n! \leq n^n = (2^{\log_2 n})^n$$

# Wie groß ist $n!$ ?

$$\underbrace{n/2 \cdot n/2 \cdot \dots \cdot n/2}_{n/2 \text{ mal}} \leq n! = 1 \cdot 2 \cdot \dots \cdot n \leq n \cdot n \cdot \dots \cdot n = n^n$$

$$\Rightarrow n! \leq n^n = (2^{\log_2 n})^n =$$

# Wie groß ist $n!$ ?

$$\underbrace{n/2 \cdot n/2 \cdot \dots \cdot n/2}_{n/2 \text{ mal}} \leq n! = 1 \cdot 2 \cdot \dots \cdot n \leq n \cdot n \cdot \dots \cdot n = n^n$$

$$\Rightarrow n! \leq n^n = (2^{\log_2 n})^n = 2^{n \log_2 n}$$

# Wie groß ist $n!$ ?

$$\underbrace{n/2 \cdot n/2 \cdot \dots \cdot n/2}_{n/2 \text{ mal}} \leq n! = 1 \cdot 2 \cdot \dots \cdot n \leq n \cdot n \cdot \dots \cdot n = n^n$$

$$\Rightarrow \leq n! \leq n^n = (2^{\log_2 n})^n = 2^{n \log_2 n}$$

# Wie groß ist $n!$ ?

$$\underbrace{n/2 \cdot n/2 \cdot \dots \cdot n/2}_{n/2 \text{ mal}} \leq n! = 1 \cdot 2 \cdot \dots \cdot n \leq n \cdot n \cdot \dots \cdot n = n^n$$

$$\Rightarrow 2^{n/2 \log_2 n/2} \leq n! \leq n^n = (2^{\log_2 n})^n = 2^{n \log_2 n}$$

# Wie groß ist $n!$ ?

$$\underbrace{n/2 \cdot n/2 \cdot \dots \cdot n/2}_{n/2 \text{ mal}} \leq n! = 1 \cdot 2 \cdot \dots \cdot n \leq n \cdot n \cdot \dots \cdot n = n^n$$

$$\Rightarrow 2^{n/2 \log_2 n/2} \leq n! \leq n^n = (2^{\log_2 n})^n = 2^{n \log_2 n}$$

 $\Rightarrow$ 

$$n! \in$$

# Wie groß ist $n!$ ?

$$\underbrace{n/2 \cdot n/2 \cdot \dots \cdot n/2}_{n/2 \text{ mal}} \leq n! = 1 \cdot 2 \cdot \dots \cdot n \leq n \cdot n \cdot \dots \cdot n = n^n$$

$$\Rightarrow 2^{n/2 \log_2 n/2} \leq n! \leq n^n = (2^{\log_2 n})^n = 2^{n \log_2 n}$$

 $\Rightarrow$ 

$$n! \in 2^{\Theta(n \log n)}$$

# Wie groß ist $n!$ ?

$$\underbrace{n/2 \cdot n/2 \cdot \dots \cdot n/2}_{n/2 \text{ mal}} \leq n! = 1 \cdot 2 \cdot \dots \cdot n \leq n \cdot n \cdot \dots \cdot n = n^n$$

$$\Rightarrow 2^{n/2 \log_2 n/2} \leq n! \leq n^n = (2^{\log_2 n})^n = 2^{n \log_2 n}$$

$$\Rightarrow n! \in 2^{\Theta(n \log n)}$$

Genauer: Sterlingformel

[James Sterling, 1692–1770]



# Wie groß ist $n!$ ?

$$\underbrace{n/2 \cdot n/2 \cdot \dots \cdot n/2}_{n/2 \text{ mal}} \leq n! = 1 \cdot 2 \cdot \dots \cdot n \leq n \cdot n \cdot \dots \cdot n = n^n$$

$$\Rightarrow 2^{n/2 \log_2 n/2} \leq n! \leq n^n = (2^{\log_2 n})^n = 2^{n \log_2 n}$$

$$\Rightarrow n! \in 2^{\Theta(n \log n)}$$

Genauer: Sterlingformel

[James Sterling, 1692–1770]

Für  $n \rightarrow \infty$  gilt

$$n! \sim \sqrt{2\pi n} \left(\frac{n}{e}\right)^n.$$

# Wie groß ist $n!$ ?

$$\underbrace{n/2 \cdot n/2 \cdot \dots \cdot n/2}_{n/2 \text{ mal}} \leq n! = 1 \cdot 2 \cdot \dots \cdot n \leq n \cdot n \cdot \dots \cdot n = n^n$$

$$\Rightarrow 2^{n/2 \log_2 n/2} \leq n! \leq n^n = (2^{\log_2 n})^n = 2^{n \log_2 n}$$

$$\Rightarrow n! \in 2^{\Theta(n \log n)}$$

Genauer: Sterlingformel

[James Sterling, 1692–1770]

Für  $n \rightarrow \infty$  gilt

$$n! \sim \sqrt{2\pi n} \left(\frac{n}{e}\right)^n.$$

Noch genauer:

$$\sqrt{2\pi} \sqrt{n} \left(\frac{n}{e}\right)^n \leq n! \leq e \sqrt{n} \left(\frac{n}{e}\right)^n$$

# Exakt, aber schneller: Dynamisches Programm

Wir beginnen alle Rundtouren im Knoten  $v_1$ .

Für eine Knotenmenge  $W \subseteq V \setminus \{v_1\}$  mit  $v_i \in W$  definieren wir

$\text{OPT}[W, v_i] :=$  optimale (kürzeste) Länge eines  $v_1$ - $v_i$ -Wegs

# Exakt, aber schneller: Dynamisches Programm

Wir beginnen alle Rundtouren im Knoten  $v_1$ .

Für eine Knotenmenge  $W \subseteq V \setminus \{v_1\}$  mit  $v_i \in W$  definieren wir

$\text{OPT}[W, v_i] :=$  optimale (kürzeste) Länge eines  $v_1$ - $v_i$ -Wegs  
*durch alle Knoten in  $W$ .*

# Exakt, aber schneller: Dynamisches Programm

Wir beginnen alle Rundtouren im Knoten  $v_1$ .

Für eine Knotenmenge  $W \subseteq V \setminus \{v_1\}$  mit  $v_i \in W$  definieren wir

$\text{OPT}[W, v_i] :=$  optimale (kürzeste) Länge eines  $v_1$ - $v_i$ -Wegs  
*durch alle Knoten in  $W$ .*

Idee für DP: *Definiere Wert einer opt. Lösung rekursiv!*

# Exakt, aber schneller: Dynamisches Programm

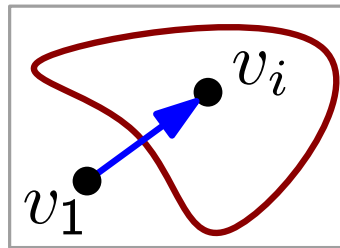
Wir beginnen alle Rundtouren im Knoten  $v_1$ .

Für eine Knotenmenge  $W \subseteq V \setminus \{v_1\}$  mit  $v_i \in W$  definieren wir  
 $\text{OPT}[W, v_i] :=$  optimale (kürzeste) Länge eines  $v_1$ - $v_i$ -Wegs  
*durch alle Knoten in  $W$ .*

Idee für DP: *Definiere Wert einer opt. Lösung rekursiv!*

Dann gilt für  $W = \{v_i\}$ :

$$\text{OPT}[W, v_i] =$$



# Exakt, aber schneller: Dynamisches Programm

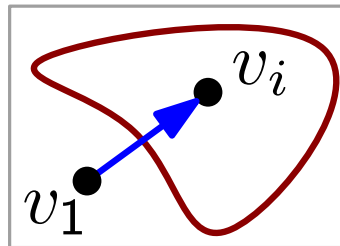
Wir beginnen alle Rundtouren im Knoten  $v_1$ .

Für eine Knotenmenge  $W \subseteq V \setminus \{v_1\}$  mit  $v_i \in W$  definieren wir  $\text{OPT}[W, v_i] :=$  optimale (kürzeste) Länge eines  $v_1$ - $v_i$ -Wegs durch alle Knoten in  $W$ .

Idee für DP: *Definiere Wert einer opt. Lösung rekursiv!*

Dann gilt für  $W = \{v_i\}$ :

$$\text{OPT}[W, v_i] = c(v_1, v_i)$$



# Exakt, aber schneller: Dynamisches Programm

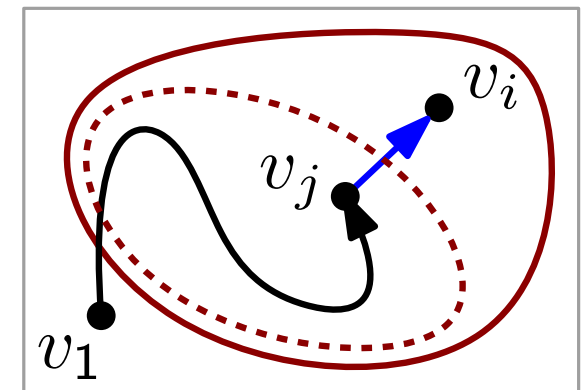
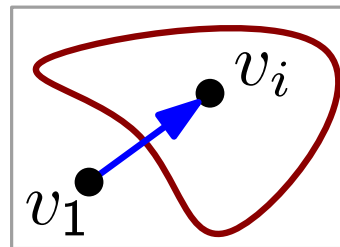
Wir beginnen alle Rundtouren im Knoten  $v_1$ .

Für eine Knotenmenge  $W \subseteq V \setminus \{v_1\}$  mit  $v_i \in W$  definieren wir  $\text{OPT}[W, v_i] :=$  optimale (kürzeste) Länge eines  $v_1$ - $v_i$ -Wegs *durch alle Knoten in  $W$* .

Idee für DP: *Definiere Wert einer opt. Lösung rekursiv!*

Dann gilt für  $W = \{v_i\}$ :

$$\text{OPT}[W, v_i] = c(v_1, v_i)$$



Und für  $W$  mit  $\{v_i\} \subsetneq W$ :

$$\text{OPT}[W, v_i] =$$



# Exakt, aber schneller: Dynamisches Programm

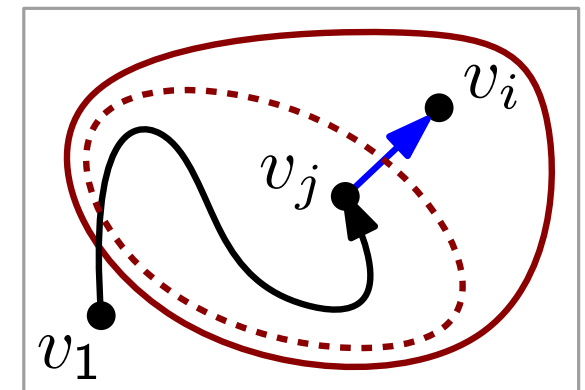
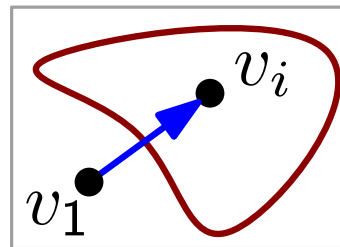
Wir beginnen alle Rundtouren im Knoten  $v_1$ .

Für eine Knotenmenge  $W \subseteq V \setminus \{v_1\}$  mit  $v_i \in W$  definieren wir  $\text{OPT}[W, v_i] :=$  optimale (kürzeste) Länge eines  $v_1$ - $v_i$ -Wegs *durch alle Knoten in  $W$* .

Idee für DP: *Definiere Wert einer opt. Lösung rekursiv!*

Dann gilt für  $W = \{v_i\}$ :

$$\text{OPT}[W, v_i] = c(v_1, v_i)$$



Und für  $W$  mit  $\{v_i\} \subsetneq W$ :

$$\text{OPT}[W, v_i] = \min_{v_j \in W \setminus \{v_i\}}$$

Letzter Knoten vor  $v_i$

# Exakt, aber schneller: Dynamisches Programm

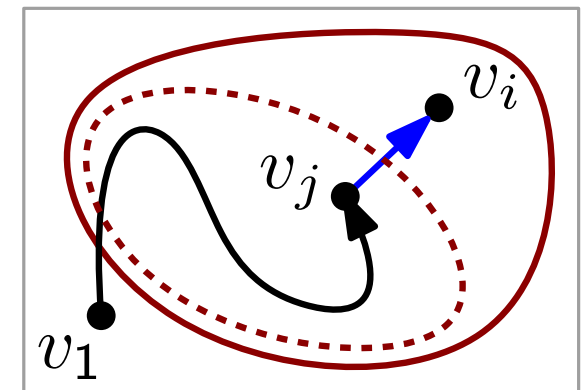
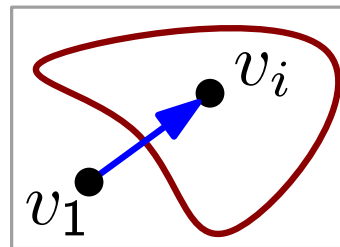
Wir beginnen alle Rundtouren im Knoten  $v_1$ .

Für eine Knotenmenge  $W \subseteq V \setminus \{v_1\}$  mit  $v_i \in W$  definieren wir  $\text{OPT}[W, v_i] :=$  optimale (kürzeste) Länge eines  $v_1$ - $v_i$ -Wegs durch alle Knoten in  $W$ .

Idee für DP: *Definiere Wert einer opt. Lösung rekursiv!*

Dann gilt für  $W = \{v_i\}$ :

$$\text{OPT}[W, v_i] = c(v_1, v_i)$$



Und für  $W$  mit  $\{v_i\} \subsetneq W$ :

$$\text{OPT}[W, v_i] = \min_{v_j \in W \setminus \{v_i\}} \text{OPT}[W \setminus \{v_i\}, v_j]$$

Letzter Knoten vor  $v_i$

# Exakt, aber schneller: Dynamisches Programm

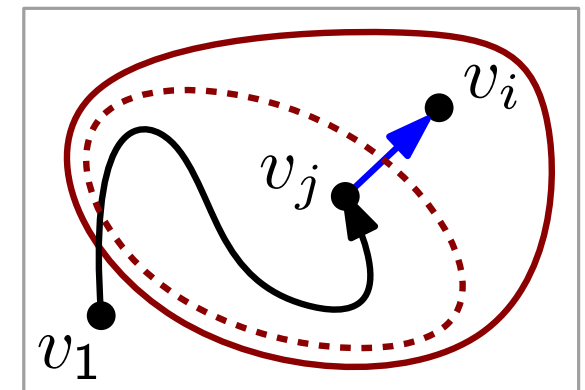
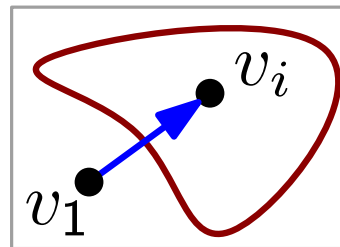
Wir beginnen alle Rundtouren im Knoten  $v_1$ .

Für eine Knotenmenge  $W \subseteq V \setminus \{v_1\}$  mit  $v_i \in W$  definieren wir  $\text{OPT}[W, v_i] :=$  optimale (kürzeste) Länge eines  $v_1$ - $v_i$ -Wegs durch alle Knoten in  $W$ .

Idee für DP: *Definiere Wert einer opt. Lösung rekursiv!*

Dann gilt für  $W = \{v_i\}$ :

$$\text{OPT}[W, v_i] = c(v_1, v_i)$$



Und für  $W$  mit  $\{v_i\} \subsetneq W$ :

$$\text{OPT}[W, v_i] = \min_{v_j \in W \setminus \{v_i\}} \text{OPT}[W \setminus \{v_i\}, v_j] +$$

# Exakt, aber schneller: Dynamisches Programm

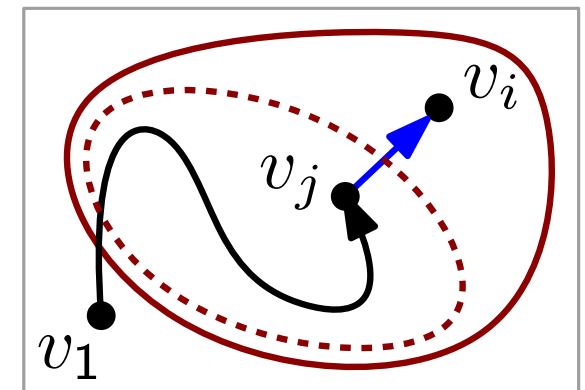
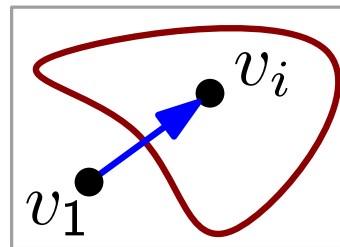
Wir beginnen alle Rundtouren im Knoten  $v_1$ .

Für eine Knotenmenge  $W \subseteq V \setminus \{v_1\}$  mit  $v_i \in W$  definieren wir  $\text{OPT}[W, v_i] :=$  optimale (kürzeste) Länge eines  $v_1$ - $v_i$ -Wegs *durch alle Knoten in  $W$* .

Idee für DP: *Definiere Wert einer opt. Lösung rekursiv!*

Dann gilt für  $W = \{v_i\}$ :

$$\text{OPT}[W, v_i] = c(v_1, v_i)$$



Und für  $W$  mit  $\{v_i\} \subsetneq W$ :

$$\text{OPT}[W, v_i] = \min_{v_j \in W \setminus \{v_i\}} \text{OPT}[W \setminus \{v_i\}, v_j] + c(v_j, v_i)$$

# Exakt, aber schneller: Dynamisches Programm

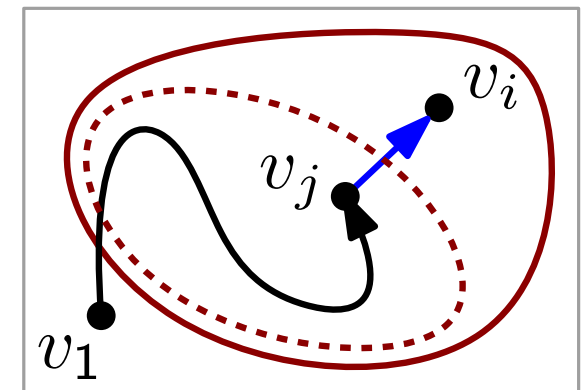
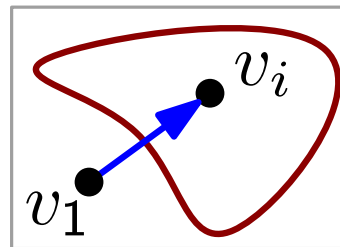
Wir beginnen alle Rundtouren im Knoten  $v_1$ .

Für eine Knotenmenge  $W \subseteq V \setminus \{v_1\}$  mit  $v_i \in W$  definieren wir  $\text{OPT}[W, v_i] :=$  optimale (kürzeste) Länge eines  $v_1$ - $v_i$ -Wegs *durch alle Knoten in  $W$* .

Idee für DP: *Definiere Wert einer opt. Lösung rekursiv!*

Dann gilt für  $W = \{v_i\}$ :

$$\text{OPT}[W, v_i] = c(v_1, v_i)$$



Und für  $W$  mit  $\{v_i\} \subsetneq W$ :

$$\text{OPT}[W, v_i] = \min_{v_j \in W \setminus \{v_i\}} (\text{OPT}[W \setminus \{v_i\}, v_j] + c(v_j, v_i))$$

Letzter Knoten vor  $v_i$

# Exakt, aber schneller: Dynamisches Programm

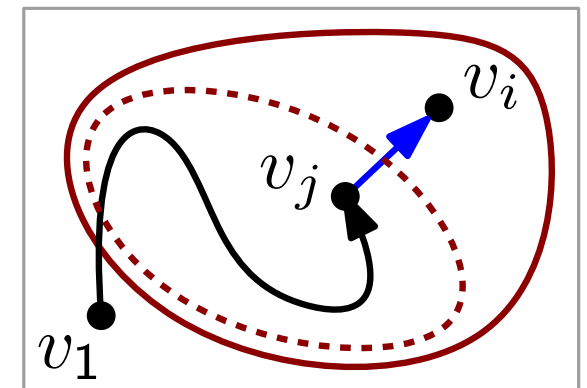
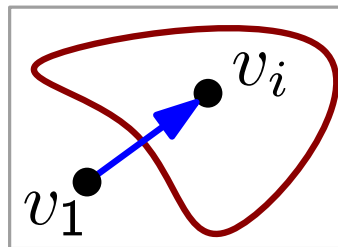
Wir beginnen alle Rundtouren im Knoten  $v_1$ .

Für eine Knotenmenge  $W \subseteq V \setminus \{v_1\}$  mit  $v_i \in W$  definieren wir  $\text{OPT}[W, v_i] :=$  optimale (kürzeste) Länge eines  $v_1$ - $v_i$ -Wegs durch alle Knoten in  $W$ .

Idee für DP: *Definiere Wert einer opt. Lösung rekursiv!*

Dann gilt für  $W = \{v_i\}$ :

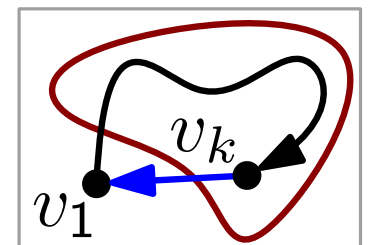
$$\text{OPT}[W, v_i] = c(v_1, v_i)$$



Und für  $W$  mit  $\{v_i\} \subsetneq W$ :

$$\text{OPT}[W, v_i] = \min_{v_j \in W \setminus \{v_i\}} (\text{OPT}[W \setminus \{v_i\}, v_j] + c(v_j, v_i))$$

$\Rightarrow \text{OPT} =$



# Exakt, aber schneller: Dynamisches Programm

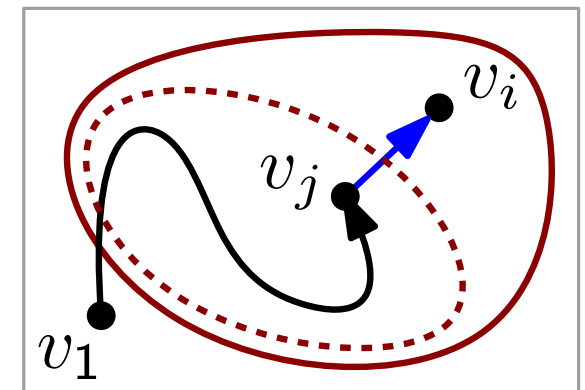
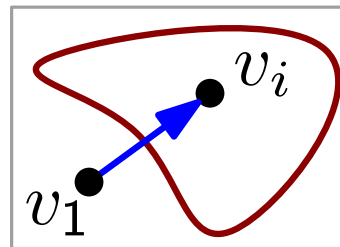
Wir beginnen alle Rundtouren im Knoten  $v_1$ .

Für eine Knotenmenge  $W \subseteq V \setminus \{v_1\}$  mit  $v_i \in W$  definieren wir  $\text{OPT}[W, v_i] :=$  optimale (kürzeste) Länge eines  $v_1$ - $v_i$ -Wegs durch alle Knoten in  $W$ .

Idee für DP: *Definiere Wert einer opt. Lösung rekursiv!*

Dann gilt für  $W = \{v_i\}$ :

$$\text{OPT}[W, v_i] = c(v_1, v_i)$$

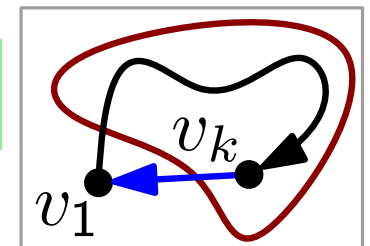


Und für  $W$  mit  $\{v_i\} \subsetneq W$ :

$$\text{OPT}[W, v_i] = \min_{v_j \in W \setminus \{v_i\}} (\text{OPT}[W \setminus \{v_i\}, v_j] + c(v_j, v_i))$$

$$\Rightarrow \text{OPT} = \min_{k \neq 1}$$

Index des letzten Knotens vor  $v_1$



# Exakt, aber schneller: Dynamisches Programm

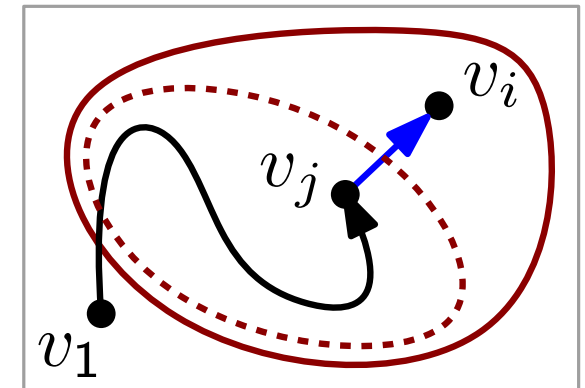
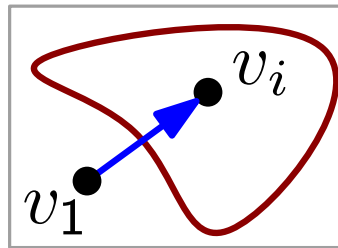
Wir beginnen alle Rundtouren im Knoten  $v_1$ .

Für eine Knotenmenge  $W \subseteq V \setminus \{v_1\}$  mit  $v_i \in W$  definieren wir  $\text{OPT}[W, v_i] :=$  optimale (kürzeste) Länge eines  $v_1$ - $v_i$ -Wegs durch alle Knoten in  $W$ .

Idee für DP: *Definiere Wert einer opt. Lösung rekursiv!*

Dann gilt für  $W = \{v_i\}$ :

$$\text{OPT}[W, v_i] = c(v_1, v_i)$$

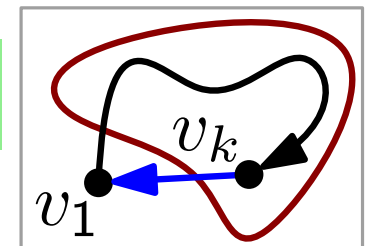


Und für  $W$  mit  $\{v_i\} \subsetneq W$ :

$$\text{OPT}[W, v_i] = \min_{v_j \in W \setminus \{v_i\}} (\text{OPT}[W \setminus \{v_i\}, v_j] + c(v_j, v_i))$$

$$\Rightarrow \text{OPT} = \min_{k \neq 1} \text{OPT}[V \setminus \{v_1\}, v_k]$$

Index des letzten Knotens vor  $v_1$





# Exakt, aber schneller: Dynamisches Programm

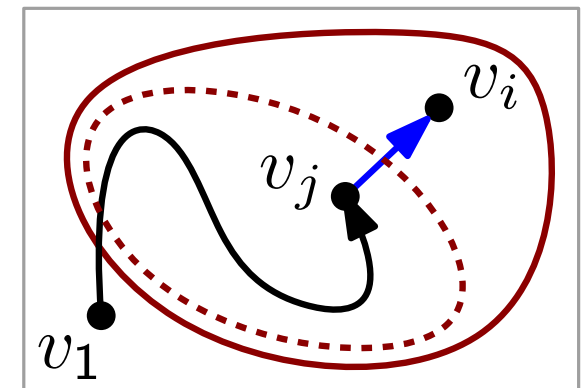
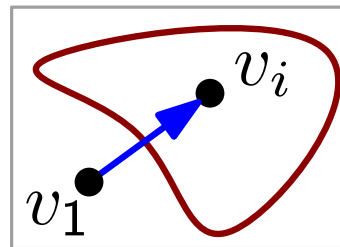
Wir beginnen alle Rundtouren im Knoten  $v_1$ .

Für eine Knotenmenge  $W \subseteq V \setminus \{v_1\}$  mit  $v_i \in W$  definieren wir  $\text{OPT}[W, v_i] :=$  optimale (kürzeste) Länge eines  $v_1$ - $v_i$ -Wegs durch alle Knoten in  $W$ .

Idee für DP: *Definiere Wert einer opt. Lösung rekursiv!*

Dann gilt für  $W = \{v_i\}$ :

$$\text{OPT}[W, v_i] = c(v_1, v_i)$$

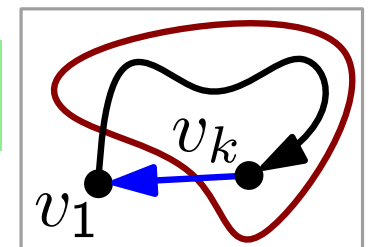


Und für  $W$  mit  $\{v_i\} \subsetneq W$ :

$$\text{OPT}[W, v_i] = \min_{v_j \in W \setminus \{v_i\}} (\text{OPT}[W \setminus \{v_i\}, v_j] + c(v_j, v_i))$$

$$\Rightarrow \text{OPT} = \min_{k \neq 1} \text{OPT}[V \setminus \{v_1\}, v_k] +$$

Index des letzten Knotens vor  $v_1$



# Exakt, aber schneller: Dynamisches Programm

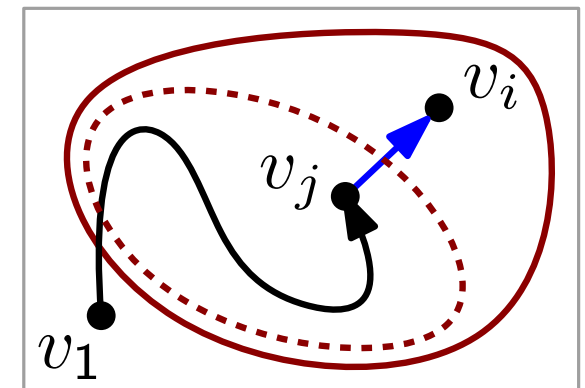
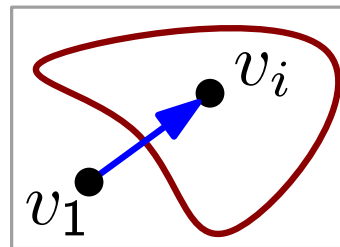
Wir beginnen alle Rundtouren im Knoten  $v_1$ .

Für eine Knotenmenge  $W \subseteq V \setminus \{v_1\}$  mit  $v_i \in W$  definieren wir  $\text{OPT}[W, v_i] :=$  optimale (kürzeste) Länge eines  $v_1$ - $v_i$ -Wegs durch alle Knoten in  $W$ .

Idee für DP: *Definiere Wert einer opt. Lösung rekursiv!*

Dann gilt für  $W = \{v_i\}$ :

$$\text{OPT}[W, v_i] = c(v_1, v_i)$$

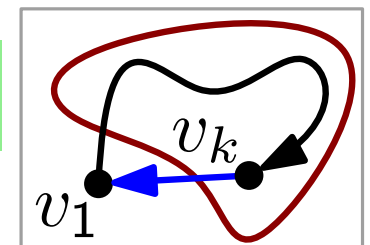


Und für  $W$  mit  $\{v_i\} \subsetneq W$ :

$$\text{OPT}[W, v_i] = \min_{v_j \in W \setminus \{v_i\}} (\text{OPT}[W \setminus \{v_i\}, v_j] + c(v_j, v_i))$$

$$\Rightarrow \text{OPT} = \min_{k \neq 1} \text{OPT}[V \setminus \{v_1\}, v_k] + c(v_k, v_1)$$

Index des letzten Knotens vor  $v_1$



# Exakt, aber schneller: Dynamisches Programm

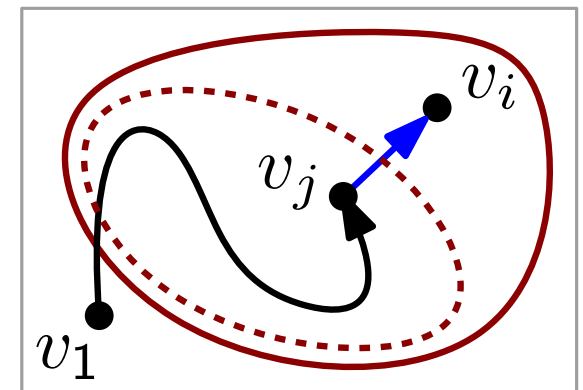
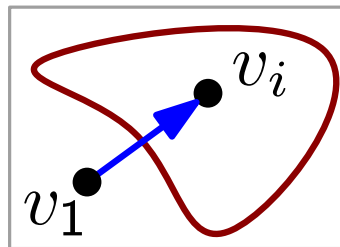
Wir beginnen alle Rundtouren im Knoten  $v_1$ .

Für eine Knotenmenge  $W \subseteq V \setminus \{v_1\}$  mit  $v_i \in W$  definieren wir  $\text{OPT}[W, v_i] :=$  optimale (kürzeste) Länge eines  $v_1$ - $v_i$ -Wegs durch alle Knoten in  $W$ .

Idee für DP: *Definiere Wert einer opt. Lösung rekursiv!*

Dann gilt für  $W = \{v_i\}$ :

$$\text{OPT}[W, v_i] = c(v_1, v_i)$$

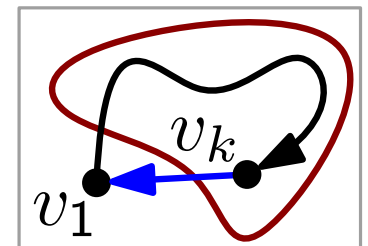


Und für  $W$  mit  $\{v_i\} \subsetneq W$ :

$$\text{OPT}[W, v_i] = \min_{v_j \in W \setminus \{v_i\}} (\text{OPT}[W \setminus \{v_i\}, v_j] + c(v_j, v_i))$$

$$\Rightarrow \text{OPT} = \min_{k \neq 1} (\text{OPT}[V \setminus \{v_1\}, v_k] + c(v_k, v_1))$$

Index des letzten Knotens vor  $v_1$



# Der Algorithmus von Bellman & Held-Karp

# Der Algorithmus von Bellman & Held-Karp

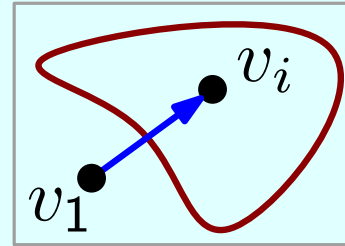
BellmanHeldKarp(Knotenmenge  $V$ , Abstände  $c: V \times V \rightarrow \mathbb{R}_{\geq 0}$ )

# Der Algorithmus von Bellman & Held-Karp

BellmanHeldKarp(Knotenmenge  $V$ , Abstände  $c: V \times V \rightarrow \mathbb{R}_{\geq 0}$ )

**for**  $i = 2$  **to**  $n$  **do**

└  $\text{OPT}[\{v_i\}, v_i] = c$

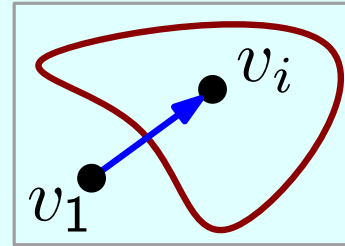


# Der Algorithmus von Bellman & Held-Karp

BellmanHeldKarp(Knotenmenge  $V$ , Abstände  $c: V \times V \rightarrow \mathbb{R}_{\geq 0}$ )

**for**  $i = 2$  **to**  $n$  **do**

└ **OPT**[\{ $v_i$ \},  $v_i$ ] =  $c(v_1, v_i)$

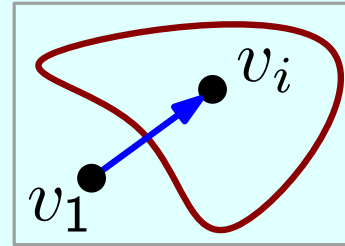


# Der Algorithmus von Bellman & Held-Karp

BellmanHeldKarp(Knotenmenge  $V$ , Abstände  $c: V \times V \rightarrow \mathbb{R}_{\geq 0}$ )

**for**  $i = 2$  **to**  $n$  **do**

└ **OPT**  $[\{v_i\}, v_i] = c(v_1, v_i)$



**for**  $j = 2$  **to**  $n - 1$  **do**

└ **foreach**  $W \subseteq \{v_2, \dots, v_n\}$  mit  $|W| = j$  **do**

└ **for**

└

└

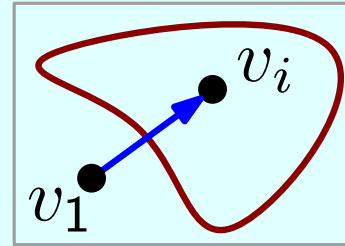


# Der Algorithmus von Bellman & Held-Karp

BellmanHeldKarp(Knotenmenge  $V$ , Abstände  $c: V \times V \rightarrow \mathbb{R}_{\geq 0}$ )

**for**  $i = 2$  **to**  $n$  **do**

└  $\text{OPT}[\{v_i\}, v_i] = c(v_1, v_i)$

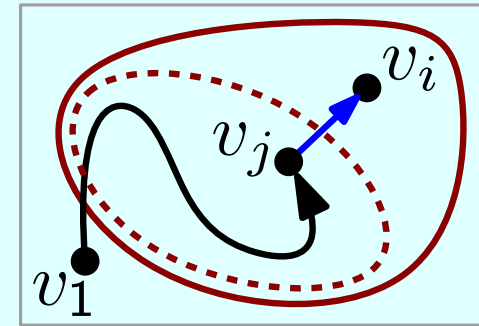


**for**  $j = 2$  **to**  $n - 1$  **do**

└ **foreach**  $W \subseteq \{v_2, \dots, v_n\}$  mit  $|W| = j$  **do**

└└ **foreach**  $v_i \in W$  **do**

└└└  $\text{OPT}[W, v_i] = r$

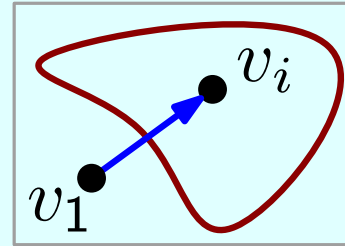


# Der Algorithmus von Bellman & Held-Karp

BellmanHeldKarp(Knotenmenge  $V$ , Abstände  $c: V \times V \rightarrow \mathbb{R}_{\geq 0}$ )

**for**  $i = 2$  **to**  $n$  **do**

└  $\text{OPT}[\{v_i\}, v_i] = c(v_1, v_i)$

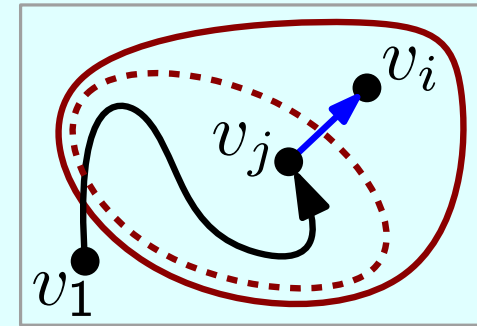


**for**  $j = 2$  **to**  $n - 1$  **do**

└ **foreach**  $W \subseteq \{v_2, \dots, v_n\}$  mit  $|W| = j$  **do**

└ **foreach**  $v_i \in W$  **do**

└  $\text{OPT}[W, v_i] = \min_{v_j \in W \setminus \{v_i\}} (c(v_1, v_j) + \text{OPT}[W \setminus \{v_j\}, v_i])$

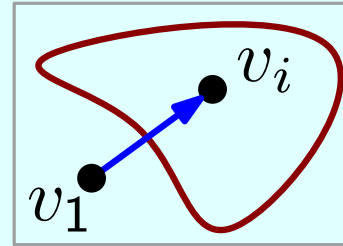


# Der Algorithmus von Bellman & Held-Karp

BellmanHeldKarp(Knotenmenge  $V$ , Abstände  $c: V \times V \rightarrow \mathbb{R}_{\geq 0}$ )

**for**  $i = 2$  **to**  $n$  **do**

└ **OPT** $[\{v_i\}, v_i] = c(v_1, v_i)$

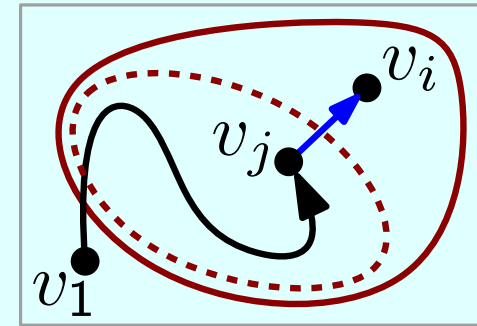


**for**  $j = 2$  **to**  $n - 1$  **do**

└ **foreach**  $W \subseteq \{v_2, \dots, v_n\}$  mit  $|W| = j$  **do**

└ **foreach**  $v_i \in W$  **do**

└ **OPT** $[W, v_i] = \min_{v_j \in W \setminus \{v_i\}} (\text{OPT}[W \setminus \{v_i\}, v_j] -$

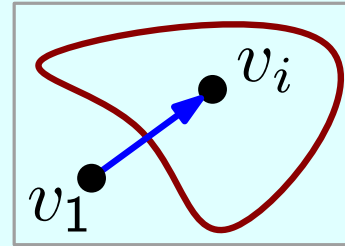


# Der Algorithmus von Bellman & Held-Karp

BellmanHeldKarp(Knotenmenge  $V$ , Abstände  $c: V \times V \rightarrow \mathbb{R}_{\geq 0}$ )

**for**  $i = 2$  **to**  $n$  **do**

└ **OPT** $[\{v_i\}, v_i] = c(v_1, v_i)$

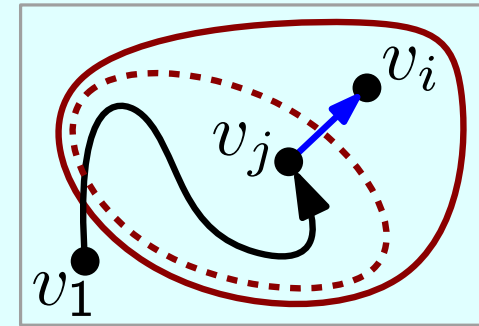


**for**  $j = 2$  **to**  $n - 1$  **do**

└ **foreach**  $W \subseteq \{v_2, \dots, v_n\}$  mit  $|W| = j$  **do**

└ **foreach**  $v_i \in W$  **do**

└ **OPT** $[W, v_i] = \min_{v_j \in W \setminus \{v_i\}} (\text{OPT}[W \setminus \{v_i\}, v_j] + c(v_j, v_i))$

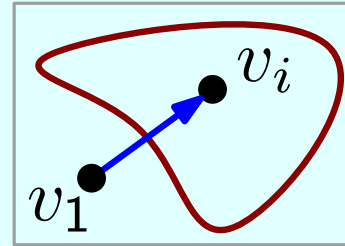


# Der Algorithmus von Bellman & Held-Karp

BellmanHeldKarp(Knotenmenge  $V$ , Abstände  $c: V \times V \rightarrow \mathbb{R}_{\geq 0}$ )

**for**  $i = 2$  **to**  $n$  **do**

└  $\text{OPT}[\{v_i\}, v_i] = c(v_1, v_i)$

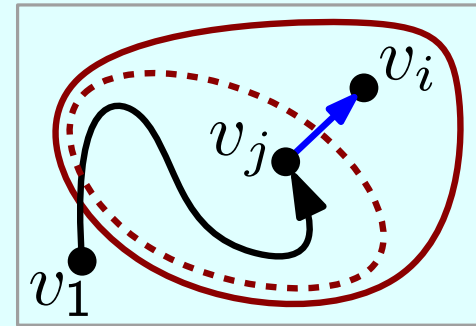


**for**  $j = 2$  **to**  $n - 1$  **do**

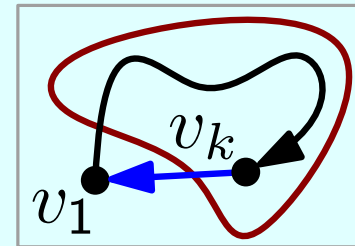
└ **foreach**  $W \subseteq \{v_2, \dots, v_n\}$  mit  $|W| = j$  **do**

└ **foreach**  $v_i \in W$  **do**

└  $\text{OPT}[W, v_i] = \min_{v_j \in W \setminus \{v_i\}} (\text{OPT}[W \setminus \{v_i\}, v_j] + c(v_j, v_i))$



**return**

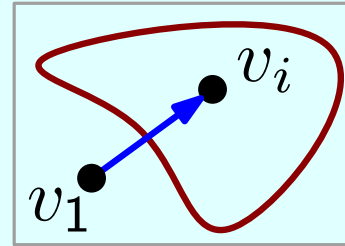


# Der Algorithmus von Bellman & Held-Karp

BellmanHeldKarp(Knotenmenge  $V$ , Abstände  $c: V \times V \rightarrow \mathbb{R}_{\geq 0}$ )

**for**  $i = 2$  **to**  $n$  **do**

└  $\text{OPT}[\{v_i\}, v_i] = c(v_1, v_i)$

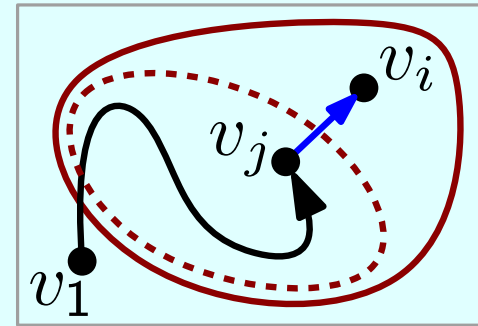


**for**  $j = 2$  **to**  $n - 1$  **do**

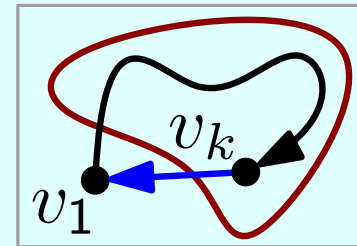
└ **foreach**  $W \subseteq \{v_2, \dots, v_n\}$  mit  $|W| = j$  **do**

└ **foreach**  $v_i \in W$  **do**

└  $\text{OPT}[W, v_i] = \min_{v_j \in W \setminus \{v_i\}} (\text{OPT}[W \setminus \{v_i\}, v_j] + c(v_j, v_i))$



**return**  $\min_{k \neq 1} (\text{OPT}[V \setminus \{v_1\}, v_k] + c(v_k, v_1))$

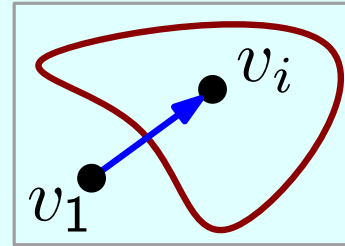


# Der Algorithmus von Bellman & Held-Karp

BellmanHeldKarp(Knotenmenge  $V$ , Abstände  $c: V \times V \rightarrow \mathbb{R}_{\geq 0}$ )

**for**  $i = 2$  **to**  $n$  **do**

└  $\text{OPT}[\{v_i\}, v_i] = c(v_1, v_i)$

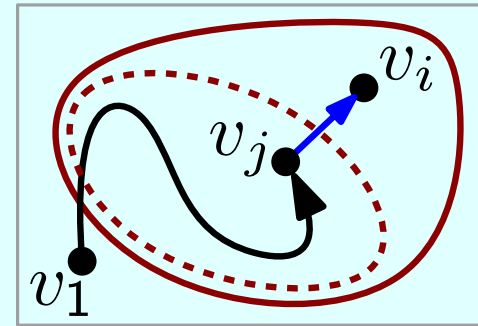


**for**  $j = 2$  **to**  $n - 1$  **do**

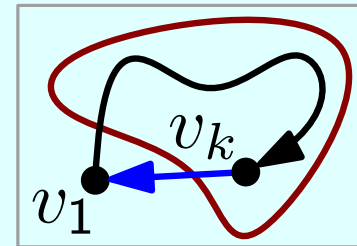
└ **foreach**  $W \subseteq \{v_2, \dots, v_n\}$  mit  $|W| = j$  **do**

└ **foreach**  $v_i \in W$  **do**

└  $\text{OPT}[W, v_i] = \min_{v_j \in W \setminus \{v_i\}} (\text{OPT}[W \setminus \{v_i\}, v_j] + c(v_j, v_i))$



**return**  $\min_{k \neq 1} (\text{OPT}[V \setminus \{v_1\}, v_k] + c(v_k, v_1))$



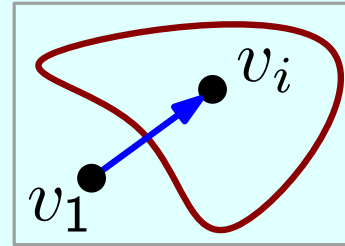
**Laufzeit:** Berechnung von  $\text{OPT}[W, v_i]$ :

# Der Algorithmus von Bellman & Held-Karp

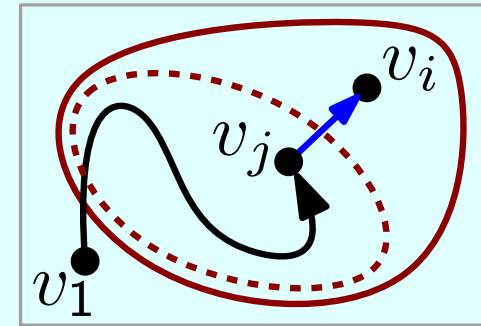
BellmanHeldKarp(Knotenmenge  $V$ , Abstände  $c: V \times V \rightarrow \mathbb{R}_{\geq 0}$ )

**for**  $i = 2$  **to**  $n$  **do**

└  $\text{OPT}[\{v_i\}, v_i] = c(v_1, v_i)$



**for**  $j = 2$  **to**  $n - 1$  **do**

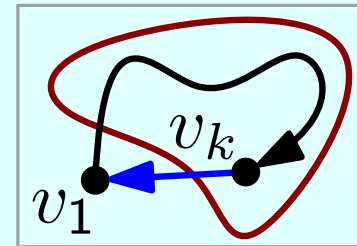


└ **foreach**  $W \subseteq \{v_2, \dots, v_n\}$  mit  $|W| = j$  **do**

└ **foreach**  $v_i \in W$  **do**

└  $\text{OPT}[W, v_i] = \min_{v_j \in W \setminus \{v_i\}} (\text{OPT}[W \setminus \{v_i\}, v_j] + c(v_j, v_i))$

**return**  $\min_{k \neq 1} (\text{OPT}[V \setminus \{v_1\}, v_k] + c(v_k, v_1))$



**Laufzeit:** Berechnung von  $\text{OPT}[W, v_i]$ :  $O(n)$

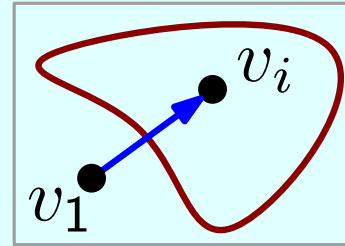


# Der Algorithmus von Bellman & Held-Karp

BellmanHeldKarp(Knotenmenge  $V$ , Abstände  $c: V \times V \rightarrow \mathbb{R}_{\geq 0}$ )

**for**  $i = 2$  **to**  $n$  **do**

└ **OPT** $[\{v_i\}, v_i] = c(v_1, v_i)$

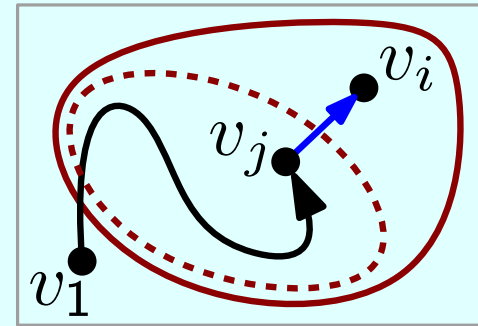


**for**  $j = 2$  **to**  $n - 1$  **do**

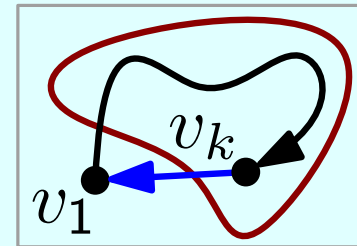
└ **foreach**  $W \subseteq \{v_2, \dots, v_n\}$  mit  $|W| = j$  **do**

└ **foreach**  $v_i \in W$  **do**

└ **OPT** $[W, v_i] = \min_{v_j \in W \setminus \{v_i\}} (\text{OPT}[W \setminus \{v_i\}, v_j] + c(v_j, v_i))$



**return**  $\min_{k \neq 1} (\text{OPT}[V \setminus \{v_1\}, v_k] + c(v_k, v_1))$



**Laufzeit:** Berechnung von  $\text{OPT}[W, v_i]$ :  $O(n)$

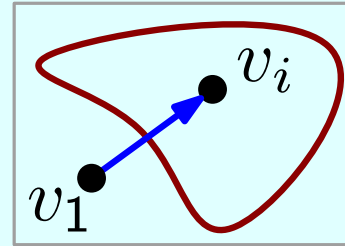
Wie viele Paare  $(W, v_i)$  mit  $v_i \in W$  gibt's?

# Der Algorithmus von Bellman & Held-Karp

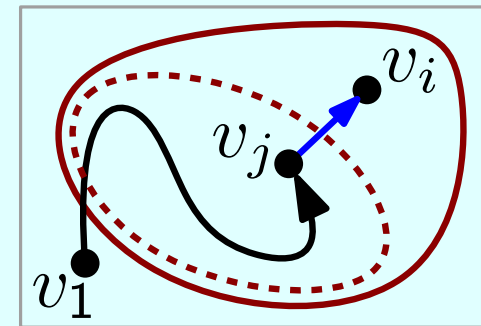
BellmanHeldKarp(Knotenmenge  $V$ , Abstände  $c: V \times V \rightarrow \mathbb{R}_{\geq 0}$ )

**for**  $i = 2$  **to**  $n$  **do**

└  $\text{OPT}[\{v_i\}, v_i] = c(v_1, v_i)$



**for**  $j = 2$  **to**  $n - 1$  **do**

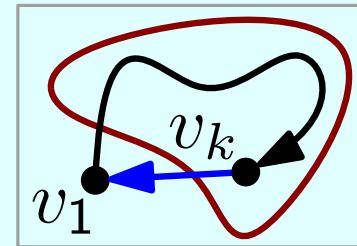


└ **foreach**  $W \subseteq \{v_2, \dots, v_n\}$  mit  $|W| = j$  **do**

└ **foreach**  $v_i \in W$  **do**

└  $\text{OPT}[W, v_i] = \min_{v_j \in W \setminus \{v_i\}} (\text{OPT}[W \setminus \{v_i\}, v_j] + c(v_j, v_i))$

**return**  $\min_{k \neq 1} (\text{OPT}[V \setminus \{v_1\}, v_k] + c(v_k, v_1))$



**Laufzeit:** Berechnung von  $\text{OPT}[W, v_i]$ :  $O(n)$

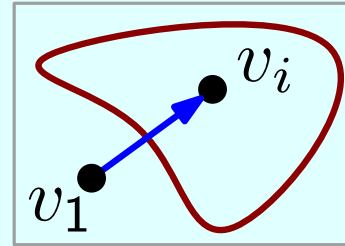
Wie viele Paare  $(W, v_i)$  mit  $v_i \in W$  gibt's?  $\leq n \cdot 2^{n-1}$

# Der Algorithmus von Bellman & Held-Karp

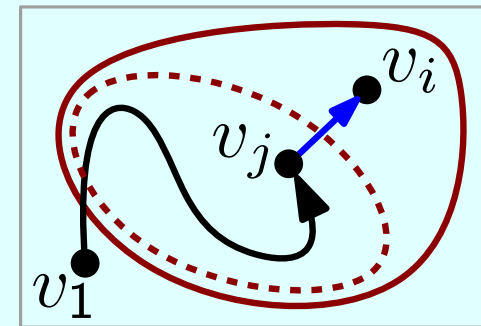
BellmanHeldKarp(Knotenmenge  $V$ , Abstände  $c: V \times V \rightarrow \mathbb{R}_{\geq 0}$ )

**for**  $i = 2$  **to**  $n$  **do**

└  $\text{OPT}[\{v_i\}, v_i] = c(v_1, v_i)$



**for**  $j = 2$  **to**  $n - 1$  **do**

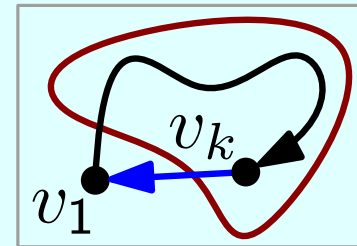


└ **foreach**  $W \subseteq \{v_2, \dots, v_n\}$  mit  $|W| = j$  **do**

└ **foreach**  $v_i \in W$  **do**

└  $\text{OPT}[W, v_i] = \min_{v_j \in W \setminus \{v_i\}} (\text{OPT}[W \setminus \{v_i\}, v_j] + c(v_j, v_i))$

**return**  $\min_{k \neq 1} (\text{OPT}[V \setminus \{v_1\}, v_k] + c(v_k, v_1))$



**Laufzeit:** Berechnung von  $\text{OPT}[W, v_i]$ :  $O(n)$

Wie viele Paare  $(W, v_i)$  mit  $v_i \in W$  gibt's?  $\leq n \cdot 2^{n-1}$

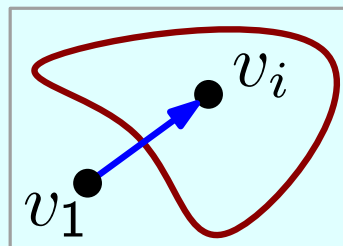
$\Rightarrow$  Gesamtlaufzeit  $\in O(\quad)$

# Der Algorithmus von Bellman & Held-Karp

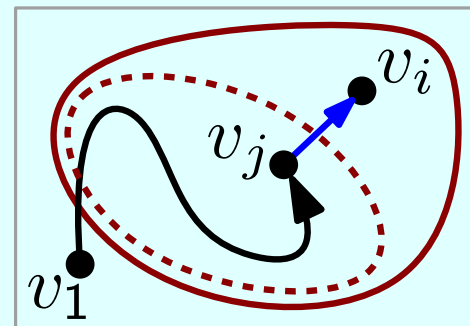
BellmanHeldKarp(Knotenmenge  $V$ , Abstände  $c: V \times V \rightarrow \mathbb{R}_{\geq 0}$ )

**for**  $i = 2$  **to**  $n$  **do**

└ **OPT** $[\{v_i\}, v_i] = c(v_1, v_i)$



**for**  $j = 2$  **to**  $n - 1$  **do**

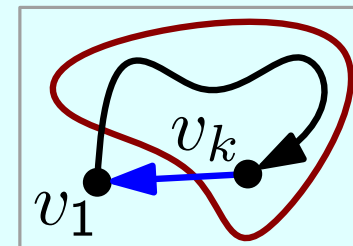


└ **foreach**  $W \subseteq \{v_2, \dots, v_n\}$  mit  $|W| = j$  **do**

└ **foreach**  $v_i \in W$  **do**

└ **OPT** $[W, v_i] = \min_{v_j \in W \setminus \{v_i\}} (\text{OPT}[W \setminus \{v_i\}, v_j] + c(v_j, v_i))$

**return**  $\min_{k \neq 1} (\text{OPT}[V \setminus \{v_1\}, v_k] + c(v_k, v_1))$



**Laufzeit:** Berechnung von  $\text{OPT}[W, v_i]$ :  $O(n)$

Wie viele Paare  $(W, v_i)$  mit  $v_i \in W$  gibt's?  $\leq n \cdot 2^{n-1}$

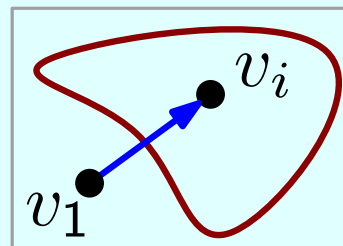
$\Rightarrow$  Gesamtlaufzeit  $\in O(n^2 \cdot 2^n)$

# Der Algorithmus von Bellman & Held-Karp

BellmanHeldKarp(Knotenmenge  $V$ , Abstände  $c: V \times V \rightarrow \mathbb{R}_{\geq 0}$ )

**for**  $i = 2$  **to**  $n$  **do**

└  $\text{OPT}[\{v_i\}, v_i] = c(v_1, v_i)$

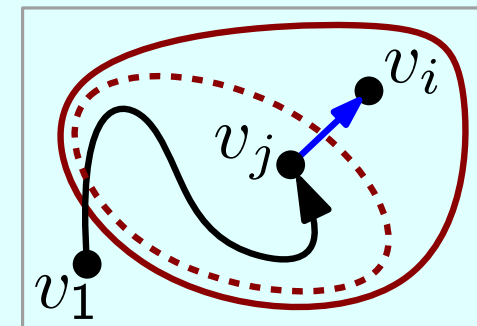


**for**  $j = 2$  **to**  $n - 1$  **do**

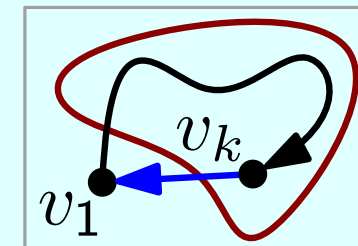
└ **foreach**  $W \subseteq \{v_2, \dots, v_n\}$  mit  $|W| = j$  **do**

└ **foreach**  $v_i \in W$  **do**

└  $\text{OPT}[W, v_i] = \min_{v_j \in W \setminus \{v_i\}} (\text{OPT}[W \setminus \{v_i\}, v_j] + c(v_j, v_i))$



**return**  $\min_{k \neq 1} (\text{OPT}[V \setminus \{v_1\}, v_k] + c(v_k, v_1))$



**Laufzeit:** Berechnung von  $\text{OPT}[W, v_i]$ :  $O(n)$

Wie viele Paare  $(W, v_i)$  mit  $v_i \in W$  gibt's?  $\leq n \cdot 2^{n-1}$

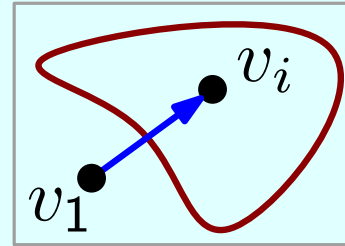
$\Rightarrow$  Gesamtlaufzeit  $\in O(n^2 \cdot 2^n)$     **Speicher:**

# Der Algorithmus von Bellman & Held-Karp

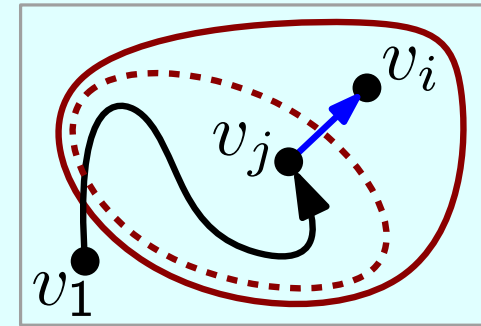
BellmanHeldKarp(Knotenmenge  $V$ , Abstände  $c: V \times V \rightarrow \mathbb{R}_{\geq 0}$ )

**for**  $i = 2$  **to**  $n$  **do**

└  $\text{OPT}[\{v_i\}, v_i] = c(v_1, v_i)$



**for**  $j = 2$  **to**  $n - 1$  **do**

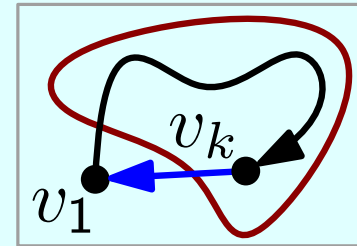


└ **foreach**  $W \subseteq \{v_2, \dots, v_n\}$  mit  $|W| = j$  **do**

└ **foreach**  $v_i \in W$  **do**

└  $\text{OPT}[W, v_i] = \min_{v_j \in W \setminus \{v_i\}} (\text{OPT}[W \setminus \{v_i\}, v_j] + c(v_j, v_i))$

**return**  $\min_{k \neq 1} (\text{OPT}[V \setminus \{v_1\}, v_k] + c(v_k, v_1))$



**Laufzeit:** Berechnung von  $\text{OPT}[W, v_i]$ :  $O(n)$

Wie viele Paare  $(W, v_i)$  mit  $v_i \in W$  gibt's?  $\leq n \cdot 2^{n-1}$

$\Rightarrow$  Gesamtlaufzeit  $\in O(n^2 \cdot 2^n)$     **Speicher:**  $O(n \cdot 2^n)$

# Vergleich

	Brute Force	Bellman/Held-Karp
Laufzeit	$2^{\Theta(n \log n)}$	$O(n^2 \cdot 2^n)$
Speicher	$O(n)$	$O(n \cdot 2^n)$

# Vergleich

Brute Force

Bellman/Held-Karp

---

Laufzeit

$$2^{\Theta(n \log n)}$$

$$O(n^2 \cdot 2^n)$$

Speicher

$$O(n)$$

$$O(n \cdot 2^n)$$



# Vergleich

	Brute Force	Bellman/Held-Karp
Laufzeit	$2^{\Theta(n \log n)}$	$O(n^2 \cdot 2^n)$
Speicher	$O(n)$	$O(n \cdot 2^n)$

Der Algorithmus von Held und Karp verringert also die Laufzeit zu Kosten des Speicherplatzverbrauchs.

# Vergleich

	Brute Force	Bellman/Held-Karp
Laufzeit	$2^{\Theta(n \log n)}$	$O(n^2 \cdot 2^n)$
Speicher	$O(n)$	$O(n \cdot 2^n)$

Der Algorithmus von Held und Karp verringert also die Laufzeit zu Kosten des Speicherplatzverbrauchs.

Das bezeichnet man als Laufzeit-Speicherplatz-*Trade-Off*.

# Vergleich

	Brute Force	Bellman/Held-Karp
Laufzeit	$2^{\Theta(n \log n)}$	$O(n^2 \cdot 2^n)$
Speicher	$O(n)$	$O(n \cdot 2^n)$

Der Algorithmus von Held und Karp verringert also die Laufzeit zu Kosten des Speicherplatzverbrauchs.

Das bezeichnet man als Laufzeit-Speicherplatz-*Trade-Off*.

**Bem.** Nederlof hat vor kurzem [STOC 2020] einen randomisierten Algorithmus für *bipartites* Matching vorgeschlagen, der in  $O(\text{poly}(n) \cdot 1.9999^n)$  Zeit läuft (falls man Matrizen schnell genug multiplizieren kann).

# Bad News

**Satz.** TSP ist NP-schwer.

# Bad News

**Satz.** TSP ist NP-schwer.

*Beweis.*

# Bad News

**Satz.** TSP ist NP-schwer.

*Beweis.* Durch Reduktion vom Problem Hamiltonkreis.

# Bad News

**Satz.** TSP ist NP-schwer.

*Beweis.* Durch Reduktion vom Problem Hamiltonkreis.  
Zu zeigen: wenn wir TSP effizient lösen könnten,  
dann auch HK.

# Bad News

**Satz.** TSP ist NP-schwer.

*Beweis.* Durch Reduktion vom Problem Hamiltonkreis.  
Zu zeigen: wenn wir TSP **effizient** lösen könnten,  
dann auch HK.



# Bad News

**Satz.** TSP ist NP-schwer.

*Beweis.* Durch Reduktion vom Problem Hamiltonkreis.  
Zu zeigen: wenn wir TSP **effizient** lösen könnten,  
dann auch HK. *in Polynomialzeit!*

# Bad News

**Satz.** TSP ist NP-schwer.

*Beweis.* Durch Reduktion vom Problem Hamiltonkreis.  
Zu zeigen: wenn wir TSP **effizient** lösen könnten,  
dann auch HK. *in Polynomialzeit!*  
Gegeben: ungerichteter (i.A. *nicht* vollständiger)  
Graph  $H = (V, E)$ .

# Bad News

**Satz.** TSP ist NP-schwer.

*Beweis.* Durch Reduktion vom Problem Hamiltonkreis.

Zu zeigen: wenn wir TSP **effizient** lösen könnten,  
dann auch HK. *in Polynomialzeit!*

Gegeben: ungerichteter (i.A. *nicht* vollständiger)  
Graph  $H = (V, E)$ .

Def. vollständigen Graphen  $G$  mit Kosten  $c$ , so dass:

$G$  hat billige TSP-Tour  $\Leftrightarrow H$  hamiltonsch.

# Bad News

**Satz.** TSP ist NP-schwer.

*Beweis.* Durch Reduktion vom Problem Hamiltonkreis.

Zu zeigen: wenn wir TSP **effizient** lösen könnten,  
dann auch HK. *in Polynomialzeit!*

Gegeben: ungerichteter (i.A. *nicht* vollständiger)  
Graph  $H = (V, E)$ .

Def. vollständigen Graphen  $G$  mit Kosten  $c$ , so dass:

$G$  hat billige TSP-Tour  $\Leftrightarrow H$  hamiltonsch.

Nimm  $G = \left( V, \binom{V}{2} \right)$  und für  $u, v \in V$  setze

$$c(uv) = \begin{cases} ? & \text{falls } uv \in E, \\ ? & \text{sonst.} \end{cases}$$

# Bad News

**Satz.** TSP ist NP-schwer.

*Beweis.* Durch Reduktion vom Problem Hamiltonkreis.

Zu zeigen: wenn wir TSP **effizient** lösen könnten,  
dann auch HK. *in Polynomialzeit!*

Gegeben: ungerichteter (i.A. *nicht* vollständiger)  
Graph  $H = (V, E)$ .

Def. vollständigen Graphen  $G$  mit Kosten  $c$ , so dass:

$G$  hat billige TSP-Tour  $\Leftrightarrow H$  hamiltonsch.

Nimm  $G = \left( V, \binom{V}{2} \right)$  und für  $u, v \in V$  setze

$$c(uv) = \begin{cases} 1, & \text{falls } uv \in E, \\ |V| & \text{sonst.} \end{cases}$$

# Bad News

**Satz.** TSP ist NP-schwer.

*Beweis.* Durch Reduktion vom Problem Hamiltonkreis.  
 Zu zeigen: wenn wir TSP **effizient** lösen könnten,  
 dann auch HK. *in Polynomialzeit!*

Gegeben: ungerichteter (i.A. *nicht* vollständiger)  
 Graph  $H = (V, E)$ .

Def. vollständigen Graphen  $G$  mit Kosten  $c$ , so dass:

$G$  hat billige TSP-Tour  $\Leftrightarrow H$  hamiltonsch.

Nimm  $G = \left( V, \binom{V}{2} \right)$  und für  $u, v \in V$  setze

$$c(uv) = \begin{cases} 1, & \text{falls } uv \in E, \\ |V| & \text{sonst.} \end{cases}$$

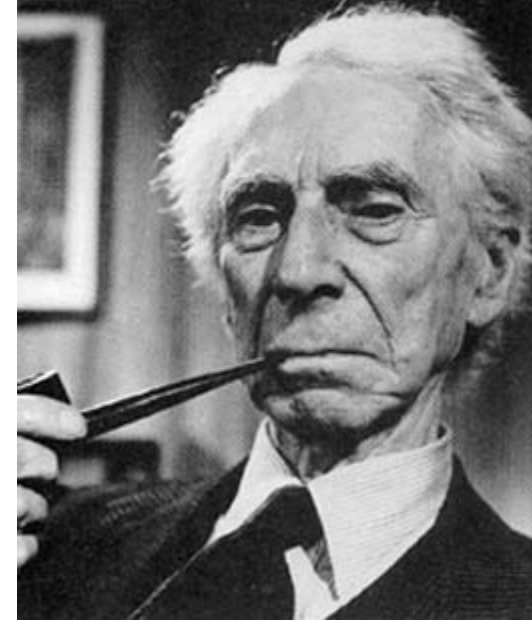
Also: Optimale TSP-Tour kostet  $|V| \Leftrightarrow H$  ham.

Was tun?

Was tun?

Bertrand Russell  
(1872–1970)

„All exact science is dominated  
by the idea of approximation.“



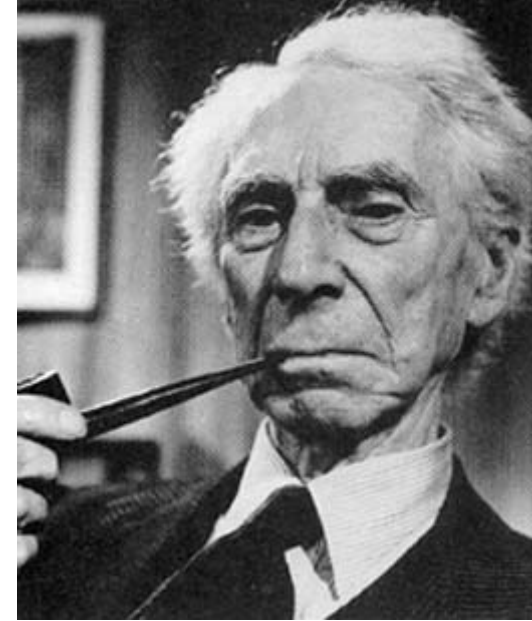


Was tun?

Bertrand Russell  
(1872–1970)

„All exact science is dominated  
by the idea of approximation.“

Sei  $\Pi$  ein *Minimierungsproblem*.

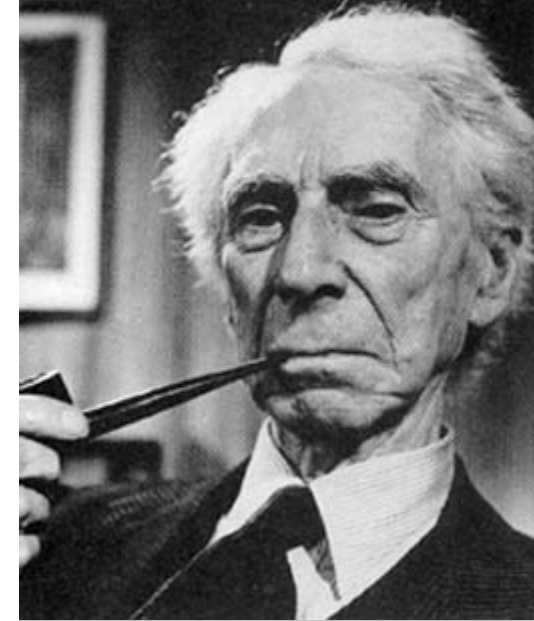


Bertrand Russell  
(1872–1970)

Was tun?

„All exact science is dominated  
by the idea of approximation.“

Sei  $\Pi$  ein *Minimierungsproblem*.



z.B. TSP

Bertrand Russell  
(1872–1970)

Was tun?

„All exact science is dominated  
by the idea of approximation.“

Sei  $\Pi$  ein *Minimierungsproblem*.

Sei  $z$  die *Zielfunktion* von  $\Pi$ : Lösung  $\mapsto \mathbb{Q}_{\geq 0}$ .



z.B. TSP

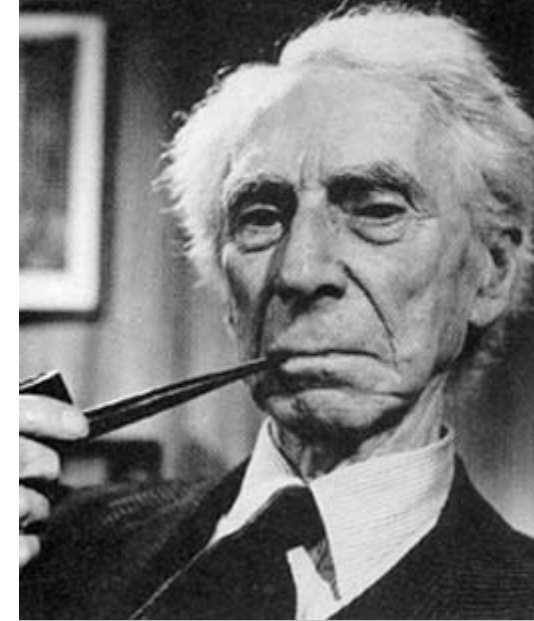
Bertrand Russell  
(1872–1970)

# Was tun?

„All exact science is dominated  
by the idea of approximation.“

Sei  $\Pi$  ein *Minimierungsproblem*.

Sei *ziel* die *Zielfunktion* von  $\Pi$ : Lösung  $\mapsto \mathbb{Q}_{\geq 0}$ .



z.B. TSP

ziel  $\equiv c$

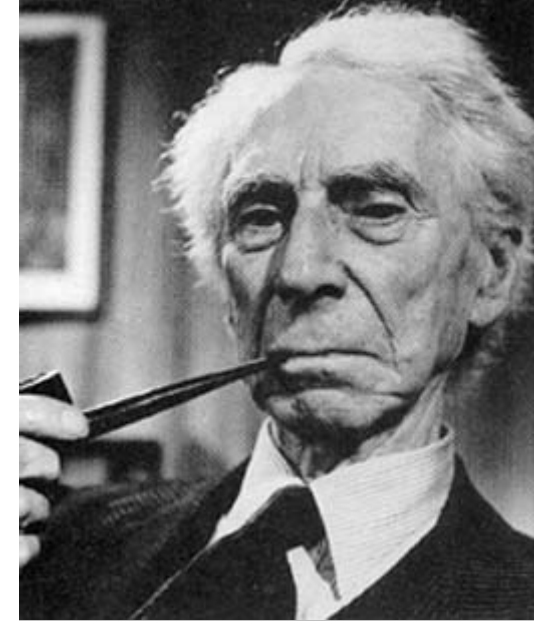
# Was tun?

„All exact science is dominated  
by the idea of approximation.“

Sei  $\Pi$  ein *Minimierungsproblem*.

Sei *ziel* die *Zielfunktion* von  $\Pi$ : Lösung  $\mapsto \mathbb{Q}_{\geq 0}$ .

Sei  $\gamma$  eine Zahl  $\geq 1$ .



z.B. TSP

ziel  $\equiv c$

# Was tun?

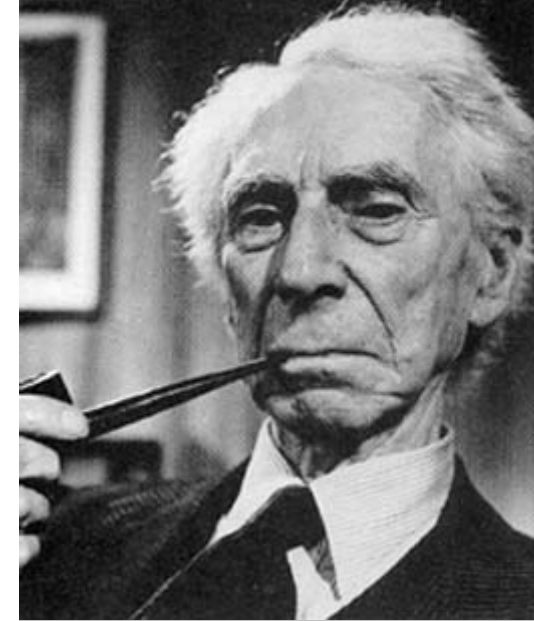
„All exact science is dominated  
by the idea of approximation.“

Sei  $\Pi$  ein *Minimierungsproblem*.

Sei *ziel* die *Zielfunktion* von  $\Pi$ : Lösung  $\mapsto \mathbb{Q}_{\geq 0}$ .

Sei  $\gamma$  eine Zahl  $\geq 1$ .

Ein Algorithmus  $\mathcal{A}$  heißt  $\gamma$ -*Approximation*, wenn



z.B. TSP

ziel  $\equiv c$

# Was tun?

„All exact science is dominated  
by the idea of approximation.“

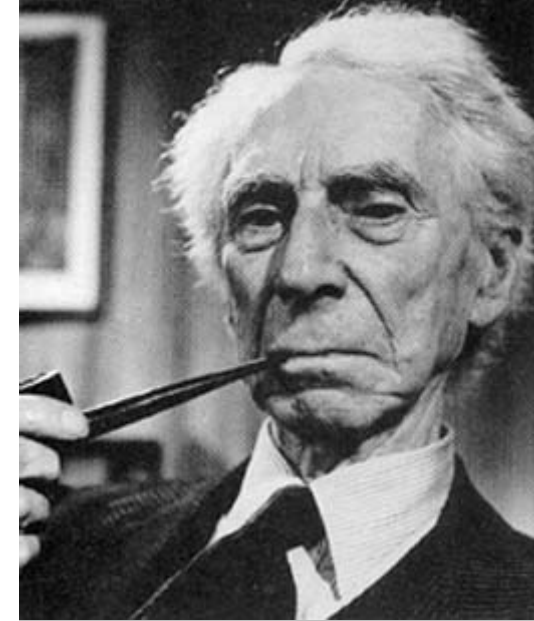
Sei  $\Pi$  ein *Minimierungsproblem*.

Sei *ziel* die *Zielfunktion* von  $\Pi$ : Lösung  $\mapsto \mathbb{Q}_{\geq 0}$ .

Sei  $\gamma$  eine Zahl  $\geq 1$ .

Güte von  $\mathcal{A}$

Ein Algorithmus  $\mathcal{A}$  heißt  $\gamma$ -*Approximation*, wenn



z.B. TSP

ziel  $\equiv c$

# Was tun?

„All exact science is dominated  
by the idea of approximation.“

Sei  $\Pi$  ein *Minimierungsproblem*.

Sei *ziel* die *Zielfunktion* von  $\Pi$ : Lösung  $\mapsto \mathbb{Q}_{\geq 0}$ .

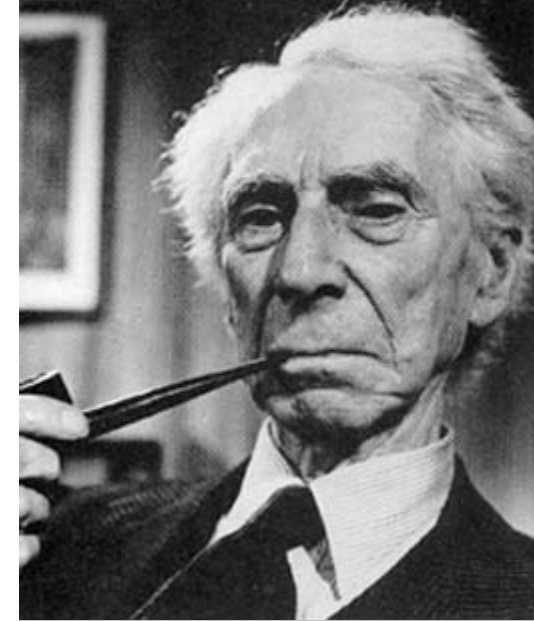
Sei  $\gamma$  eine Zahl  $\geq 1$ .

Güte von  $\mathcal{A}$

Ein Algorithmus  $\mathcal{A}$  heißt  $\gamma$ -*Approximation*, wenn

- $\mathcal{A}$  für jede Instanz  $I$  von  $\Pi$  eine Lösung  $\mathcal{A}(I)$  berechnet, so dass

$$\frac{\text{ziel}(\mathcal{A}(I))}{\text{OPT}(I)} \leq \gamma$$



z.B. TSP

ziel  $\equiv c$



# Was tun?

„All exact science is dominated  
by the idea of approximation.“

Sei  $\Pi$  ein *Minimierungsproblem*.

Sei *ziel* die *Zielfunktion* von  $\Pi$ : Lösung  $\mapsto \mathbb{Q}_{\geq 0}$ .

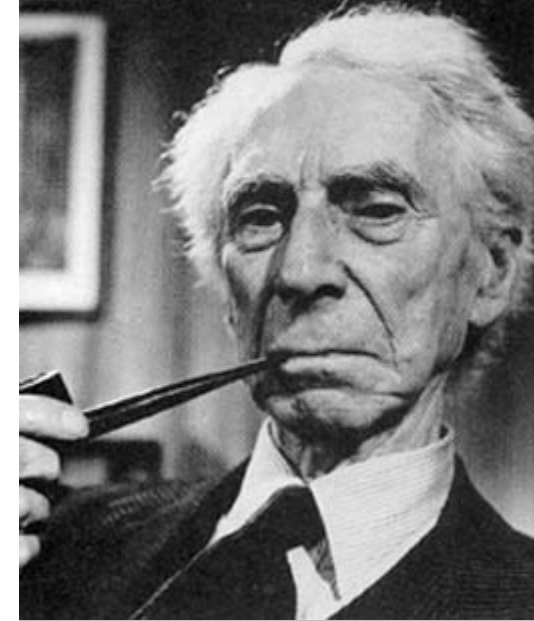
Sei  $\gamma$  eine Zahl  $\geq 1$ .

Güte von  $\mathcal{A}$

Ein Algorithmus  $\mathcal{A}$  heißt  $\gamma$ -*Approximation*, wenn

- $\mathcal{A}$  für jede Instanz  $I$  von  $\Pi$  eine Lösung  $\mathcal{A}(I)$  berechnet, so dass

$$\frac{\text{ziel}(\mathcal{A}(I))}{\text{OPT}(I)} \leq \gamma$$



z.B. TSP

ziel  $\equiv c$

$\gamma$ -Approx. für  
TSP liefert Tour  
die höchstens  $\gamma$   
mal so teuer ist  
wie billigste Tour.

# Was tun?

„All exact science is dominated  
by the idea of approximation.“

Sei  $\Pi$  ein *Minimierungsproblem*.

Sei *ziel* die *Zielfunktion* von  $\Pi$ : Lösung  $\mapsto \mathbb{Q}_{\geq 0}$ .

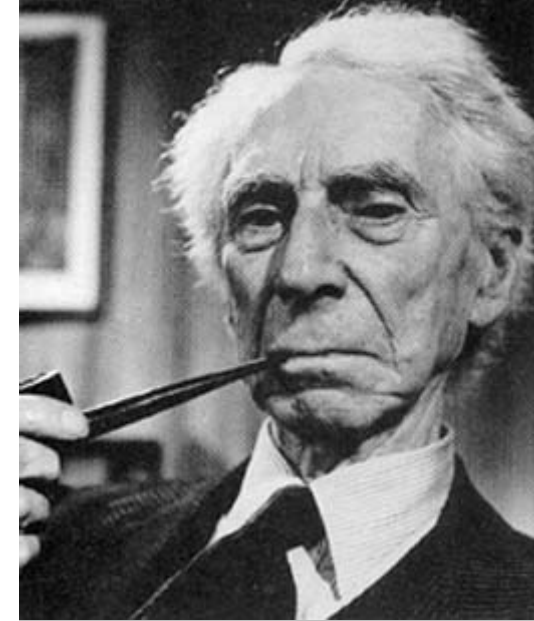
Sei  $\gamma$  eine Zahl  $\geq 1$ .

Güte von  $\mathcal{A}$

Ein Algorithmus  $\mathcal{A}$  heißt  $\gamma$ -*Approximation*, wenn

- $\mathcal{A}$  für jede Instanz  $I$  von  $\Pi$  eine Lösung  $\mathcal{A}(I)$  berechnet, so dass

$$\frac{\text{ziel}(\mathcal{A}(I))}{\text{OPT}(I)} \leq \gamma$$



z.B. TSP

ziel  $\equiv c$

$\gamma$ -Approx. für  
TSP liefert Tour  
die höchstens  $\gamma$   
mal so teuer ist  
wie billigste Tour.

# Was tun?

„All exact science is dominated  
by the idea of approximation.“

Sei  $\Pi$  ein *Minimierungsproblem*.

Sei *ziel* die *Zielfunktion* von  $\Pi$ : Lösung  $\mapsto \mathbb{Q}_{\geq 0}$ .

Sei  $\gamma$  eine Zahl  $\geq 1$ .

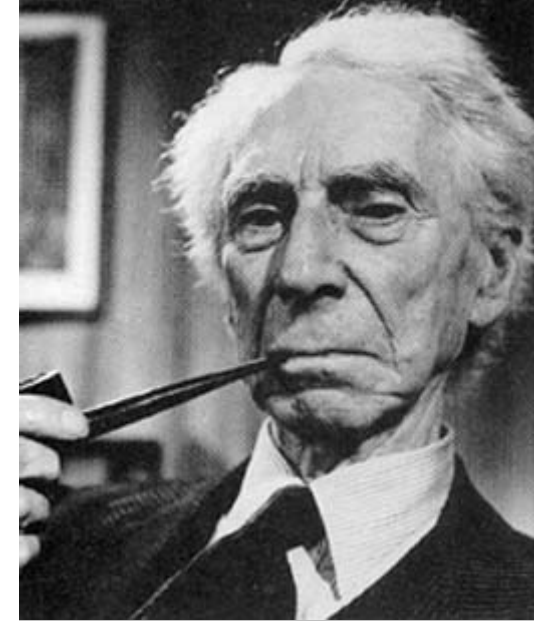
Güte von  $\mathcal{A}$

Ein Algorithmus  $\mathcal{A}$  heißt  $\gamma$ -Approximation, wenn

- $\mathcal{A}$  für jede Instanz  $I$  von  $\Pi$  eine Lösung  $\mathcal{A}(I)$  berechnet, so dass

$$\frac{\text{ziel}(\mathcal{A}(I))}{\text{OPT}(I)} \leq \gamma$$

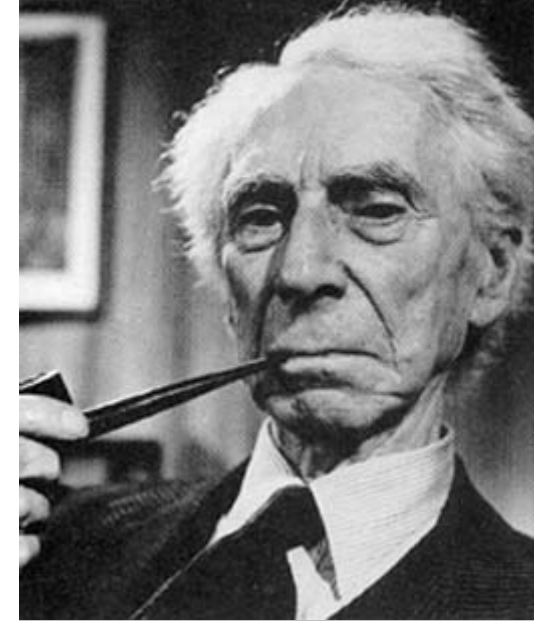
ziel(optimale Lösung)  $\rightarrow$  OPT(I)



z.B. TSP

ziel  $\equiv c$

$\gamma$ -Approx. für TSP liefert Tour die höchstens  $\gamma$  mal so teuer ist wie billigste Tour.



# Was tun?

„All exact science is dominated by the idea of approximation.“

Sei  $\Pi$  ein *Minimierungsproblem*.

Sei *ziel* die *Zielfunktion* von  $\Pi$ : Lösung  $\mapsto \mathbb{Q}_{\geq 0}$ .

Sei  $\gamma$  eine Zahl  $\geq 1$ .

Güte von  $\mathcal{A}$

Ein Algorithmus  $\mathcal{A}$  heißt  $\gamma$ -Approximation, wenn

- $\mathcal{A}$  für jede Instanz  $I$  von  $\Pi$  eine Lösung  $\mathcal{A}(I)$  berechnet, so dass

$$\frac{\text{ziel}(\mathcal{A}(I))}{\text{OPT}(I)} \leq \gamma$$

ziel(optimale Lösung) → OPT(I)

- die Laufzeit von  $\mathcal{A}$  polynomiell in  $|I|$  ist.

z.B. TSP

ziel  $\equiv c$

$\gamma$ -Approx. für TSP liefert Tour die höchstens  $\gamma$  mal so teuer ist wie billigste Tour.

# Was tun?

„All exact science is dominated  
by the idea of approximation.“

Sei  $\Pi$  ein *Minimierungsproblem*.

Sei *ziel* die *Zielfunktion* von  $\Pi$ : Lösung  $\mapsto \mathbb{Q}_{\geq 0}$ .

Sei  $\gamma$  eine Zahl  $\geq 1$ .

Güte von  $\mathcal{A}$

Ein Algorithmus  $\mathcal{A}$  heißt  $\gamma$ -*Approximation*, wenn

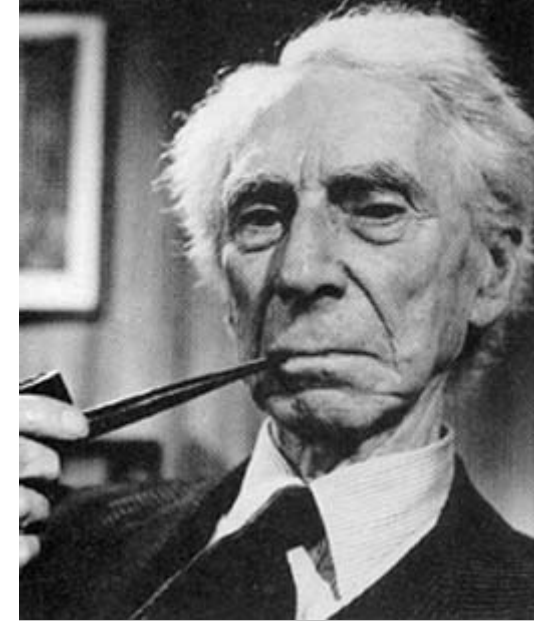
- $\mathcal{A}$  für jede Instanz  $I$  von  $\Pi$  eine Lösung  $\mathcal{A}(I)$  berechnet, so dass

$$\frac{\text{ziel}(\mathcal{A}(I))}{\text{OPT}(I)} \leq \gamma$$

ziel(optimale Lösung)

OPT(I)

- die Laufzeit von  $\mathcal{A}$  polynomiell in  $|I|$  ist.



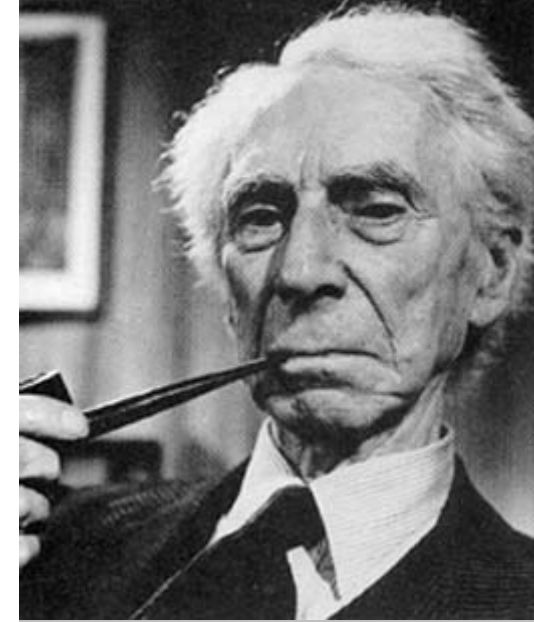
z.B. TSP

ziel  $\equiv c$

$\gamma$ -Approx. für  
TSP liefert Tour  
die höchstens  $\gamma$   
mal so teuer ist  
wie billigste Tour.

# Was tun?

„All exact science is dominated  
by the idea of approximation.“



Sei  $\Pi$  ein *Minimierungsproblem*.

Sei *ziel* die *Zielfunktion* von  $\Pi$ : Lösung  $\mapsto \mathbb{Q}_{\geq 0}$ .

Sei  $\gamma$  eine Zahl  $\geq 1$ .

Güte von  $\mathcal{A}$

Ein Algorithmus  $\mathcal{A}$  heißt  $\gamma$ -*Approximation*, wenn

- $\mathcal{A}$  für jede Instanz  $I$  von  $\Pi$  eine Lösung  $\mathcal{A}(I)$  berechnet, so dass

$$\frac{\text{ziel}(\mathcal{A}(I))}{\text{OPT}(I)} \leq \gamma$$

ziel(optimale Lösung)

OPT(I)

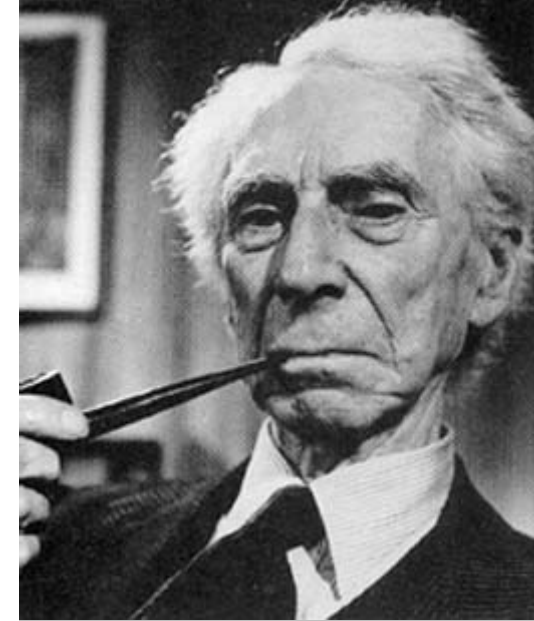
Größe der Instanz  $I$

- die Laufzeit von  $\mathcal{A}$  polynomiell in  $|I|$  ist.

z.B. TSP

ziel  $\equiv c$

$\gamma$ -Approx. für TSP liefert Tour die höchstens  $\gamma$  mal so teuer ist wie billigste Tour.



# Was tun?

„All exact science is dominated by the idea of approximation.“

Sei  $\Pi$  ein *Minimierungsproblem*.

Sei *ziel* die *Zielfunktion* von  $\Pi$ : Lösung  $\mapsto \mathbb{Q}_{\geq 0}$ .

Sei  $\gamma$  eine Zahl  $\geq 1$ .

Güte von  $\mathcal{A}$

Ein Algorithmus  $\mathcal{A}$  heißt  $\gamma$ -Approximation, wenn

- $\mathcal{A}$  für jede Instanz  $I$  von  $\Pi$  eine Lösung  $\mathcal{A}(I)$  berechnet, so dass

$$\frac{\text{ziel}(\mathcal{A}(I))}{\text{OPT}(I)} \leq \gamma$$

Größe der Instanz  $I$

- die Laufzeit von  $\mathcal{A}$  polynomiell in  $|I|$  ist.

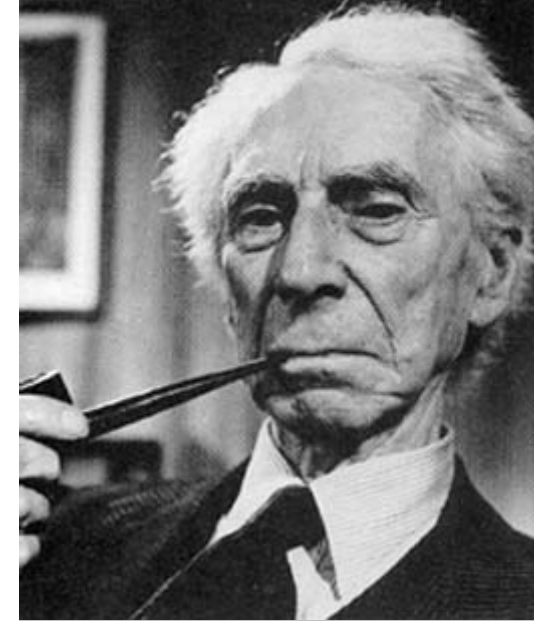
z.B. TSP

ziel  $\equiv c$

$\gamma$ -Approx. für TSP liefert Tour die höchstens  $\gamma$  mal so teuer ist wie billigste Tour.

poly( $|V|$ )





# Was tun?

„All exact science is dominated by the idea of approximation.“

## Maximierungsproblem

Sei  $\Pi$  ein ~~Minimierungsproblem~~.

Sei *ziel* die Zielfunktion von  $\Pi$ : Lösung  $\mapsto \mathbb{Q}_{\geq 0}$ .

Sei  $\gamma$  eine Zahl  $\geq 1$ .

Güte von  $\mathcal{A}$

Ein Algorithmus  $\mathcal{A}$  heißt  $\gamma$ -Approximation, wenn

- $\mathcal{A}$  für jede Instanz  $I$  von  $\Pi$  eine Lösung  $\mathcal{A}(I)$  berechnet, so dass

$$\frac{\text{ziel}(\mathcal{A}(I))}{\text{OPT}(I)} \leq \gamma$$

Größe der Instanz  $I$

- die Laufzeit von  $\mathcal{A}$  polynomiell in  $|I|$  ist.

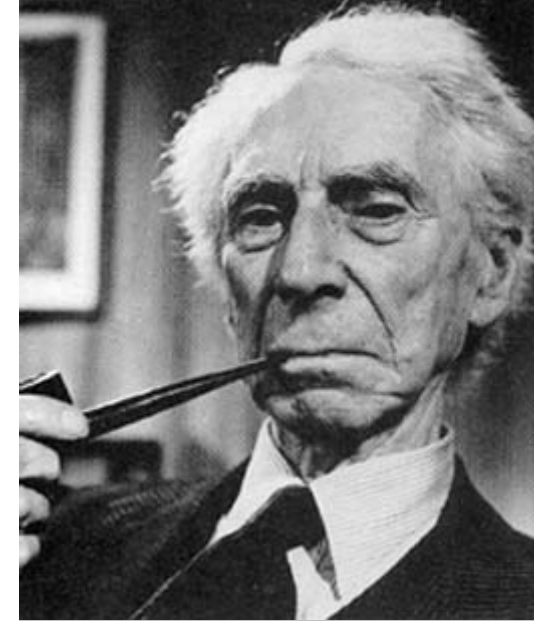
z.B. TSP

ziel  $\equiv c$

$\gamma$ -Approx. für TSP liefert Tour die höchstens  $\gamma$  mal so teuer ist wie billigste Tour.

poly( $|V|$ )





# Was tun?

„All exact science is dominated by the idea of approximation.“

## Maximierungsproblem

Sei  $\Pi$  ein ~~Minimierungsproblem~~.

Sei  $ziel$  die Zielfunktion von  $\Pi$ : Lösung  $\mapsto \mathbb{Q}_{\geq 0}$ .

Sei  $\gamma$  eine Zahl  $\not\geq 1$ .

Güte von  $\mathcal{A}$

Ein Algorithmus  $\mathcal{A}$  heißt  $\gamma$ -Approximation, wenn

- $\mathcal{A}$  für jede Instanz  $I$  von  $\Pi$  eine Lösung  $\mathcal{A}(I)$  berechnet, so dass

$$\frac{ziel(\mathcal{A}(I))}{OPT(I)} \stackrel{\geq}{\neq} \gamma$$

Größe der Instanz  $I$

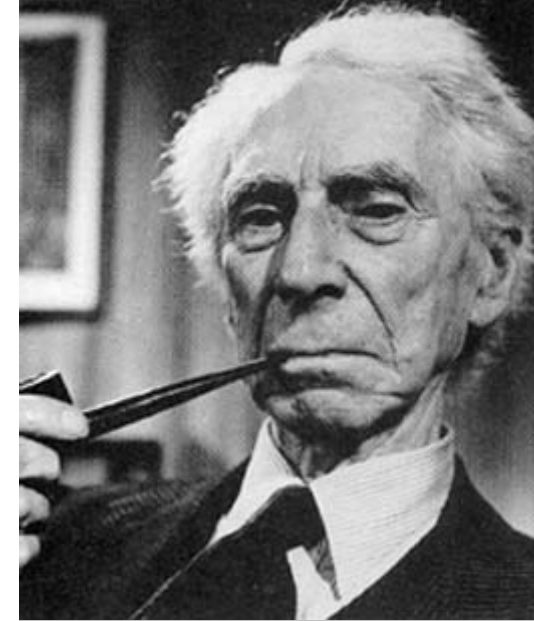
- die Laufzeit von  $\mathcal{A}$  polynomiell in  $|I|$  ist.

z.B. TSP

ziel  $\equiv c$

$\gamma$ -Approx. für TSP liefert Tour die höchstens  $\gamma$  mal so teuer ist wie billigste Tour.

poly( $|V|$ )



# Was tun?

„All exact science is dominated by the idea of approximation.“

## Maximierungsproblem

Sei  $\Pi$  ein ~~Minimierungsproblem~~.

Sei  $ziel$  die Zielfunktion von  $\Pi$ : Lösung  $\mapsto \mathbb{Q}_{\geq 0}$ .

Sei  $\gamma$  eine Zahl  $\not\geq 1$ .

Güte von  $\mathcal{A}$

Ein Algorithmus  $\mathcal{A}$  heißt  $\gamma$ -Approximation, wenn

- $\mathcal{A}$  für jede Instanz  $I$  von  $\Pi$  eine Lösung  $\mathcal{A}(I)$  berechnet, so dass

$$\frac{ziel(\mathcal{A}(I))}{\text{ziel(optimale Lösung)}} \stackrel{\geq}{\neq} \gamma$$

$\text{OPT}(I)$

- die Laufzeit von  $\mathcal{A}$  polynomiell in  $|I|$  ist.

# Was tun?

„All exact science is dominated  
by the idea of approximation.“

## Maximierungsproblem

Sei  $\Pi$  ein ~~Minimierungsproblem~~.

Sei  $ziel$  die Zielfunktion von  $\Pi$ : Lösung  $\mapsto \mathbb{Q}_{\geq 0}$ .

Sei  $\gamma$  eine Zahl  $\not\geq 1$ .

Güte von  $\mathcal{A}$

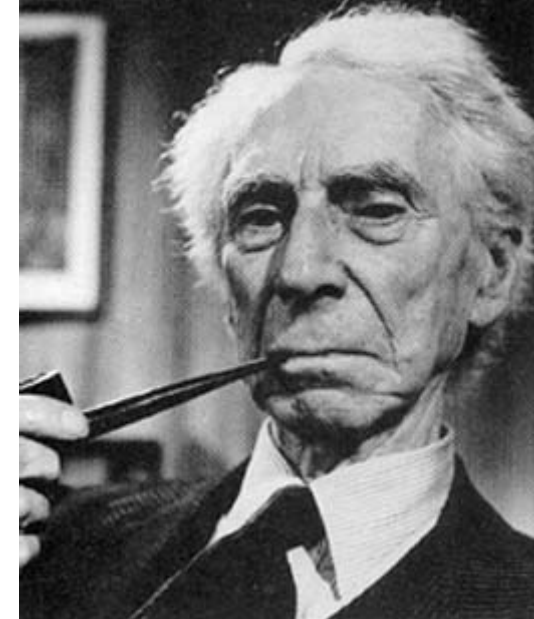
Ein Algorithmus  $\mathcal{A}$  heißt  $\gamma$ -Approximation, wenn

- $\mathcal{A}$  für jede Instanz  $I$  von  $\Pi$  eine Lösung  $\mathcal{A}(I)$  berechnet, so dass

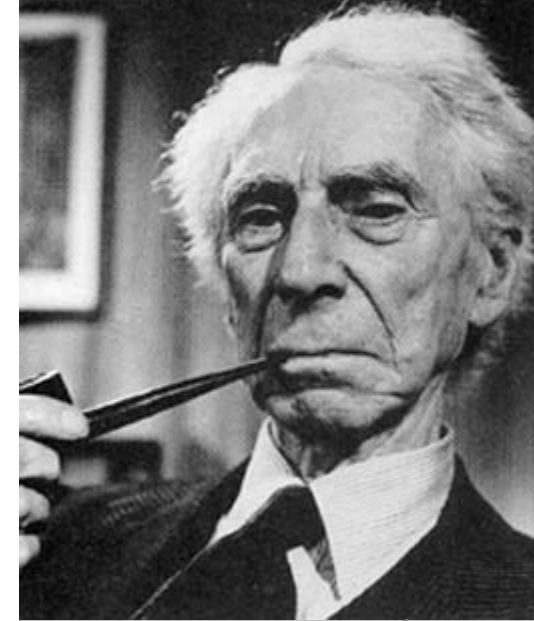
$$\frac{\text{ziel}(\mathcal{A}(I))}{\text{OPT}(I)} \stackrel{\geq}{\not\leq} \gamma$$

ziel(optimale Lösung)  $\rightarrow$  OPT(I)

- die Laufzeit von  $\mathcal{A}$  polynomiell in  $|I|$  ist.



z.B. LSP = { longest simple path



# Was tun?

„All exact science is dominated by the idea of approximation.“

## Maximierungsproblem

Sei  $\Pi$  ein ~~Minimierungsproblem~~.

Sei *ziel* die Zielfunktion von  $\Pi$ : Lösung  $\mapsto \mathbb{Q}_{\geq 0}$ .

Sei  $\gamma$  eine Zahl  $\not\geq 1$ .

Güte von  $\mathcal{A}$

Ein Algorithmus  $\mathcal{A}$  heißt  $\gamma$ -Approximation, wenn

- $\mathcal{A}$  für jede Instanz  $I$  von  $\Pi$  eine Lösung  $\mathcal{A}(I)$  berechnet, so dass

$$\frac{\text{ziel}(\mathcal{A}(I))}{\text{OPT}(I)} \not\leq \gamma$$

ziel(optimale Lösung)  $\rightarrow$  OPT(I)

- die Laufzeit von  $\mathcal{A}$  polynomiell in  $|I|$  ist.

z.B. LSP = { longest simple path }  
ziel  $\equiv$  Länge

# Was tun?

„All exact science is dominated  
by the idea of approximation.“

## Maximierungsproblem

Sei  $\Pi$  ein ~~Minimierungsproblem~~.

Sei  $ziel$  die Zielfunktion von  $\Pi$ : Lösung  $\mapsto \mathbb{Q}_{\geq 0}$ .

Sei  $\gamma$  eine Zahl  $\not\geq 1$ .

Güte von  $\mathcal{A}$

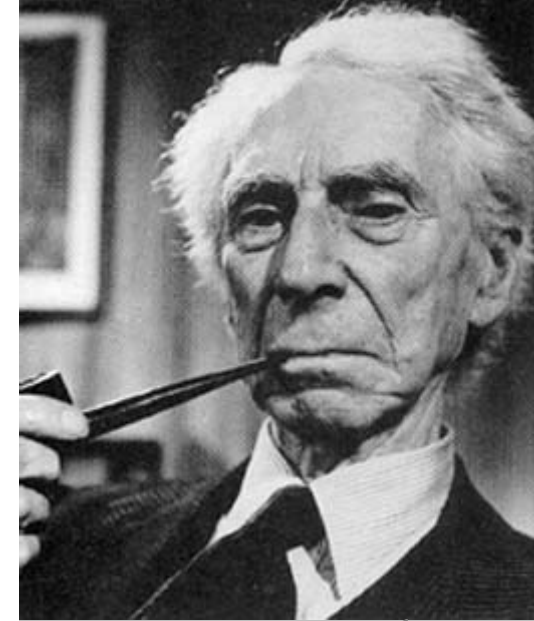
Ein Algorithmus  $\mathcal{A}$  heißt  $\gamma$ -Approximation, wenn

- $\mathcal{A}$  für jede Instanz  $I$  von  $\Pi$  eine Lösung  $\mathcal{A}(I)$  berechnet, so dass

$$\frac{\text{ziel}(\mathcal{A}(I))}{\text{OPT}(I)} \geq \gamma$$

ziel(optimale Lösung)  $\rightarrow$  OPT(I)

- die Laufzeit von  $\mathcal{A}$  polynomiell in  $|I|$  ist.



z.B. LSP = { longest  
simple  
path  
ziel  $\equiv$  Länge

$\gamma$ -Approx. für  
LSP liefert  
einfachen Pfad,  
der *mindestens*  $\gamma$   
mal so lang ist  
wie der längste.

# Was tun?

„All exact science is dominated  
by the idea of approximation.“

## Maximierungsproblem

Sei  $\Pi$  ein ~~Minimierungsproblem~~.

Sei  $ziel$  die Zielfunktion von  $\Pi$ : Lösung  $\mapsto \mathbb{Q}_{\geq 0}$ .

Sei  $\gamma$  eine Zahl  $\not\geq 1$ .

Güte von  $\mathcal{A}$

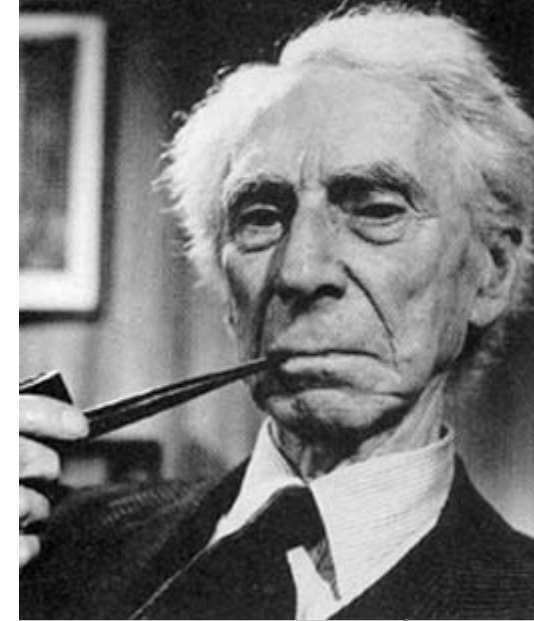
Ein Algorithmus  $\mathcal{A}$  heißt  $\gamma$ -Approximation, wenn

- $\mathcal{A}$  für jede Instanz  $I$  von  $\Pi$  eine Lösung  $\mathcal{A}(I)$  berechnet, so dass

$$\frac{\text{ziel}(\mathcal{A}(I))}{\text{OPT}(I)} \geq \gamma$$

ziel(optimale Lösung) → OPT(I)

- die Laufzeit von  $\mathcal{A}$  polynomiell in  $|I|$  ist.



z.B. LSP = { longest  
simple  
path }  
ziel  $\equiv$  Länge

$\gamma$ -Approx. für  
LSP liefert  
einfachen Pfad,  
der *mindestens*  $\gamma$   
mal so lang ist  
wie der längste.

poly( $|V|$ )

# Bad News II

**Satz.** TSP ist NP-schwer.

*Beweis.* Durch Reduktion vom Problem Hamiltonkreis.

Zu zeigen: wenn wir TSP effizient lösen könnten, dann könnten wir auch HK eff. lösen.

Gegeben: ungerichteter (i.A. *nicht* vollständiger) Graph  $H = (V, E)$ .

Def. vollständigen Graphen  $G$  mit Kosten  $c$ , so dass:  
 $G$  hat billige TSP-Tour  $\Leftrightarrow H$  hamiltonsch.

Nimm  $G = \left( V, \binom{V}{2} \right)$  und für  $u, v \in V$  setze

$$c(uv) := \begin{cases} 1, & \text{falls } uv \in E, \\ |V| & \text{sonst.} \end{cases}$$

$\Rightarrow$  optimale TSP-Tour kostet  $|V| \Leftrightarrow H$  ham.



# Bad News II

**Satz.** TSP ist NP-schwer **zu approximieren!**

*Beweis.* Durch Reduktion vom Problem Hamiltonkreis.

Zu zeigen: wenn wir TSP effizient lösen könnten, dann könnten wir auch HK eff. lösen.

Gegeben: ungerichteter (i.A. *nicht* vollständiger) Graph  $H = (V, E)$ .

Def. vollständigen Graphen  $G$  mit Kosten  $c$ , so dass:  
 $G$  hat billige TSP-Tour  $\Leftrightarrow H$  hamiltonsch.

Nimm  $G = \left( V, \binom{V}{2} \right)$  und für  $u, v \in V$  setze

$$c(uv) := \begin{cases} 1, & \text{falls } uv \in E, \\ |V| & \text{sonst.} \end{cases}$$

$\Rightarrow$  optimale TSP-Tour kostet  $|V| \Leftrightarrow H$  ham.



# Bad News II

D.h. für kein  $\gamma \geq 1$  gibt's eine  $\gamma$ -Approximation für TSP (außer  $P=NP$ ).

**Satz.** TSP ist NP-schwer **zu approximieren!**

*Beweis.* Durch Reduktion vom Problem Hamiltonkreis.

Zu zeigen: wenn wir TSP effizient lösen könnten, dann könnten wir auch HK eff. lösen.

Gegeben: ungerichteter (i.A. *nicht* vollständiger) Graph  $H = (V, E)$ .

Def. vollständigen Graphen  $G$  mit Kosten  $c$ , so dass:  
 $G$  hat billige TSP-Tour  $\Leftrightarrow H$  hamiltonsch.

Nimm  $G = \left( V, \binom{V}{2} \right)$  und für  $u, v \in V$  setze

$$c(uv) := \begin{cases} 1, & \text{falls } uv \in E, \\ |V| & \text{sonst.} \end{cases}$$

$\Rightarrow$  optimale TSP-Tour kostet  $|V| \Leftrightarrow H$  ham.

# Bad News II

D.h. für kein  $\gamma \geq 1$  gibt's eine  $\gamma$ -Approximation für TSP (außer P=NP).

**Satz.** TSP ist NP-schwer **zu approximieren!**

*Beweis.* Durch Reduktion vom Problem Hamiltonkreis.

Zu zeigen: wenn wir TSP ~~effizient lösen~~ <sup>mit Güte  $\gamma$  approximieren</sup> könnten, dann könnten wir auch HK eff. lösen.

Gegeben: ungerichteter (i.A. *nicht* vollständiger) Graph  $H = (V, E)$ .

Def. vollständigen Graphen  $G$  mit Kosten  $c$ , so dass:  
 $G$  hat billige TSP-Tour  $\Leftrightarrow H$  hamiltonsch.

Nimm  $G = \left( V, \binom{V}{2} \right)$  und für  $u, v \in V$  setze

$$c(uv) := \begin{cases} 1, & \text{falls } uv \in E, \\ |V| & \text{sonst.} \end{cases}$$

$\Rightarrow$  optimale TSP-Tour kostet  $|V| \Leftrightarrow H$  ham.

# Bad News II

D.h. für kein  $\gamma \geq 1$  gibt's eine  $\gamma$ -Approximation für TSP (außer P=NP).

**Satz.** TSP ist NP-schwer **zu approximieren!**

*Beweis.* Durch Reduktion vom Problem Hamiltonkreis.

Zu zeigen: wenn wir TSP ~~effizient lösen~~ <sup>mit Güte  $\gamma$  approximieren</sup> könnten, dann könnten wir auch HK eff. lösen.

Gegeben: ungerichteter (i.A. *nicht* vollständiger) Graph  $H = (V, E)$ .

Def. vollständigen Graphen  $G$  mit Kosten  $c$ , so dass:  
 $G$  hat billige TSP-Tour  $\Leftrightarrow H$  hamiltonsch.

Nimm  $G = \left( V, \binom{V}{2} \right)$  und für  $u, v \in V$  setze

$$c(uv) := \begin{cases} 1, & \text{falls } uv \in E, \\ |V| \cdot \gamma & \text{sonst.} \end{cases}$$

$\Rightarrow$  optimale TSP-Tour kostet  $|V| \Leftrightarrow H$  ham.

# Bad News II

D.h. für kein  $\gamma \geq 1$  gibt's eine  $\gamma$ -Approximation für TSP (außer P=NP).

**Satz.** TSP ist NP-schwer **zu approximieren!**

*Beweis.* Durch Reduktion vom Problem Hamiltonkreis.

Zu zeigen: wenn wir TSP ~~effizient lösen~~ <sup>mit Güte  $\gamma$  approximieren</sup> könnten, dann könnten wir auch HK eff. lösen.

Gegeben: ungerichteter (i.A. *nicht* vollständiger) Graph  $H = (V, E)$ .

Def. vollständigen Graphen  $G$  mit Kosten  $c$ , so dass:  
 $G$  hat billige TSP-Tour  $\Leftrightarrow H$  hamiltonsch.

Nimm  $G = \left( V, \binom{V}{2} \right)$  und für  $u, v \in V$  setze

$$c(uv) := \begin{cases} 1, & \text{falls } uv \in E, \\ |V| \cdot \gamma & \text{sonst.} \end{cases}$$

$\Rightarrow$  ~~optimale~~ TSP-Tour <sup>mit Güte  $\gamma$</sup>  kostet  $\leq \gamma \cdot |V| \Leftrightarrow H$  ham.

# Was tun?

## Problem:

### *Traveling Salesman Problem*

Gegeben: unger. vollständiger Graph  $G = (V, E)$   
mit Kantenkosten  $c: E \rightarrow \mathbb{R}_{\geq 0}$

Gesucht: Hamiltonkreis in  $G$  mit minimalen Kosten.

# Was tun? – Mach das Problem leichter!

## Problem:

### *Traveling Salesman Problem*

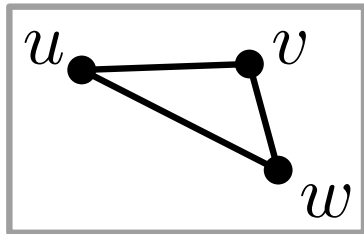
Gegeben: unger. vollständiger Graph  $G = (V, E)$   
mit Kantenkosten  $c: E \rightarrow \mathbb{R}_{\geq 0}$

Gesucht: Hamiltonkreis in  $G$  mit minimalen Kosten.

# Was tun? – Mach das Problem leichter!

## Problem:

### *Traveling Salesman Problem*

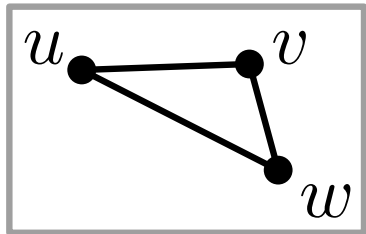


Gegeben: unger. vollständiger Graph  $G = (V, E)$   
mit Kantenkosten  $c: E \rightarrow \mathbb{R}_{\geq 0}$ ,  
die die Dreiecksungleichung erfüllen,  
d.h.  $\forall u, v, w \in V: c(u, w) \leq c(u, v) + c(v, w)$ .

Gesucht: Hamiltonkreis in  $G$  mit minimalen Kosten.

# Was tun? – Mach das Problem leichter!

**Problem:** *Metrisches Traveling Salesman Problem* ( $\Delta$ -TSP)



Gegeben: unger. vollständiger Graph  $G = (V, E)$

mit Kantenkosten  $c: E \rightarrow \mathbb{R}_{\geq 0}$ ,

die die Dreiecksungleichung erfüllen,

d.h.  $\forall u, v, w \in V: c(u, w) \leq c(u, v) + c(v, w)$ .

Gesucht: Hamiltonkreis in  $G$  mit minimalen Kosten.



# Was tun? – Mach das Problem leichter!

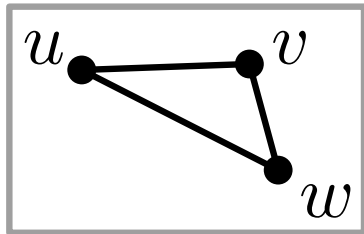
**Problem:** *Metrisches Traveling Salesman Problem* ( $\Delta$ -TSP)

Gegeben: unger. vollständiger Graph  $G = (V, E)$

mit Kantenkosten  $c: E \rightarrow \mathbb{R}_{\geq 0}$ ,

die die Dreiecksungleichung erfüllen,

d.h.  $\forall u, v, w \in V: c(u, w) \leq c(u, v) + c(v, w)$ .



Gesucht: Hamiltonkreis in  $G$  mit minimalen Kosten.

**Satz.**

Es gibt eine 2-Approximation für  $\Delta$ -TSP.

# Was tun? – Mach das Problem leichter!

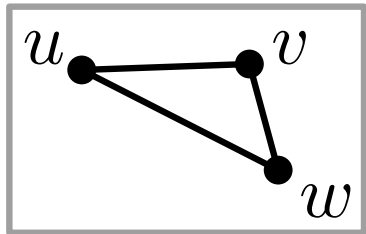
**Problem:** *Metrisches Traveling Salesman Problem* ( $\Delta$ -TSP)

Gegeben: unger. vollständiger Graph  $G = (V, E)$

mit Kantenkosten  $c: E \rightarrow \mathbb{R}_{\geq 0}$ ,  
die die Dreiecksungleichung erfüllen,

d.h.  $\forall u, v, w \in V: c(u, w) \leq c(u, v) + c(v, w)$ .

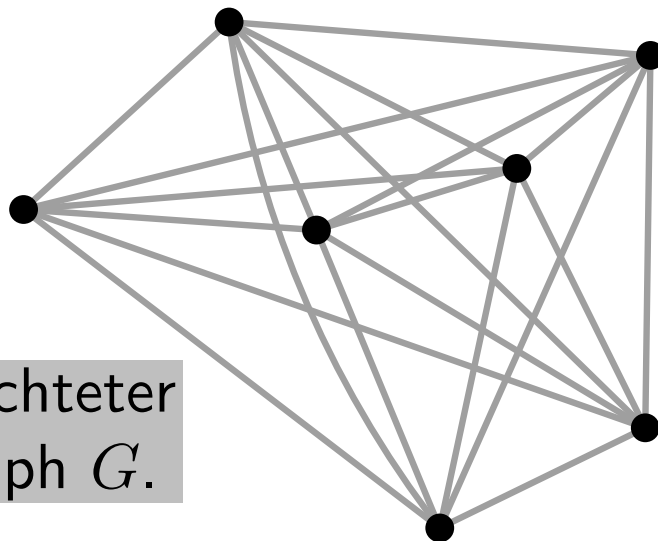
Gesucht: Hamiltonkreis in  $G$  mit minimalen Kosten.



**Satz.**

Es gibt eine 2-Approximation für  $\Delta$ -TSP.

*Beweis.*



Geg. gewichteter  
vollst. Graph  $G$ .

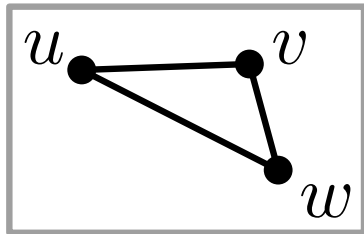
# Was tun? – Mach das Problem leichter!

**Problem:** *Metrisches Traveling Salesman Problem* ( $\Delta$ -TSP)

Gegeben: unger. vollständiger Graph  $G = (V, E)$

mit Kantenkosten  $c: E \rightarrow \mathbb{R}_{\geq 0}$ ,  
die die Dreiecksungleichung erfüllen,

d.h.  $\forall u, v, w \in V: c(u, w) \leq c(u, v) + c(v, w)$ .

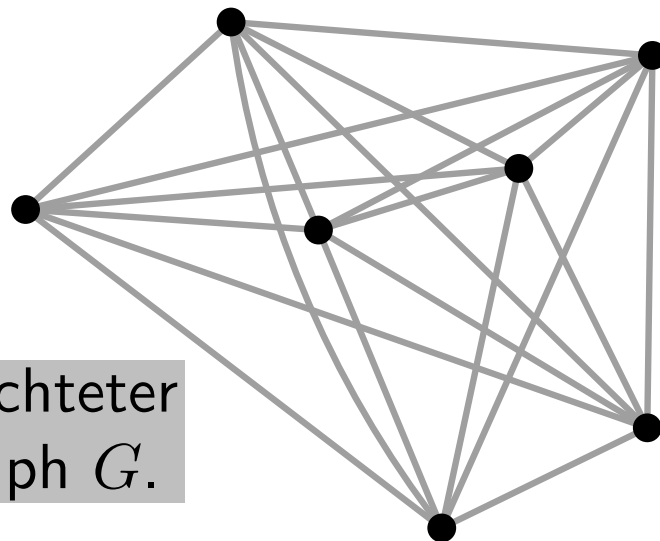


Gesucht: Hamiltonkreis in  $G$  mit minimalen Kosten.

**Satz.**

Es gibt eine 2-Approximation für  $\Delta$ -TSP.

*Beweis.*



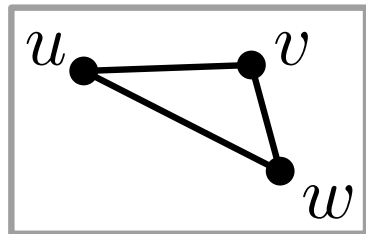
Geg. gewichteter  
vollst. Graph  $G$ .

*Algorithmus:*

Berechne min. Spannbaum **MSB**.

# Was tun? – Mach das Problem leichter!

**Problem:** *Metrisches Traveling Salesman Problem* ( $\Delta$ -TSP)

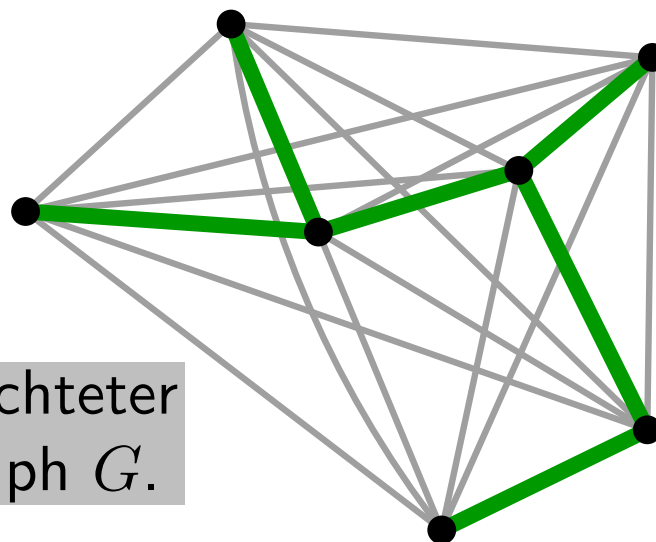


Gegeben: unger. vollständiger Graph  $G = (V, E)$   
 mit Kantenkosten  $c: E \rightarrow \mathbb{R}_{\geq 0}$ ,  
 die die Dreiecksungleichung erfüllen,  
 d.h.  $\forall u, v, w \in V: c(u, w) \leq c(u, v) + c(v, w)$ .

Gesucht: Hamiltonkreis in  $G$  mit minimalen Kosten.

**Satz.** Es gibt eine 2-Approximation für  $\Delta$ -TSP.

*Beweis.*



Geg. gewichteter  
vollst. Graph  $G$ .

*Algorithmus:*

Berechne min. Spannbaum **MSB**.

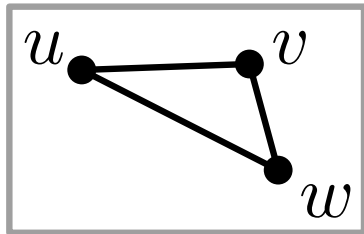
# Was tun? – Mach das Problem leichter!

**Problem:** *Metrisches Traveling Salesman Problem* ( $\Delta$ -TSP)

Gegeben: unger. vollständiger Graph  $G = (V, E)$

mit Kantenkosten  $c: E \rightarrow \mathbb{R}_{\geq 0}$ ,  
die die Dreiecksungleichung erfüllen,

d.h.  $\forall u, v, w \in V: c(u, w) \leq c(u, v) + c(v, w)$ .

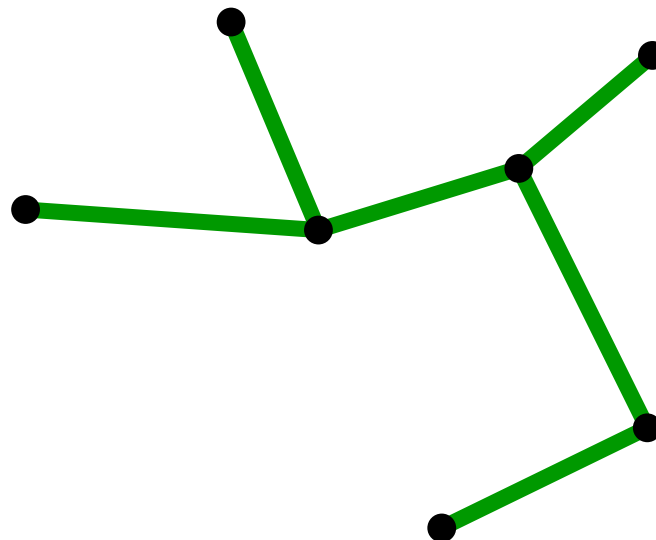


Gesucht: Hamiltonkreis in  $G$  mit minimalen Kosten.

**Satz.**

Es gibt eine 2-Approximation für  $\Delta$ -TSP.

*Beweis.*



*Algorithmus:*

Berechne min. Spannbaum **MSB**.

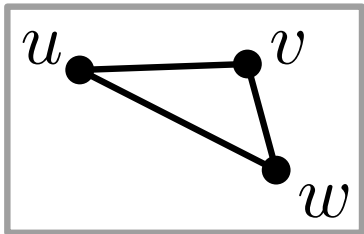
# Was tun? – Mach das Problem leichter!

**Problem:** *Metrisches Traveling Salesman Problem* ( $\Delta$ -TSP)

Gegeben: unger. vollständiger Graph  $G = (V, E)$

mit Kantenkosten  $c: E \rightarrow \mathbb{R}_{\geq 0}$ ,  
die die Dreiecksungleichung erfüllen,

d.h.  $\forall u, v, w \in V: c(u, w) \leq c(u, v) + c(v, w)$ .

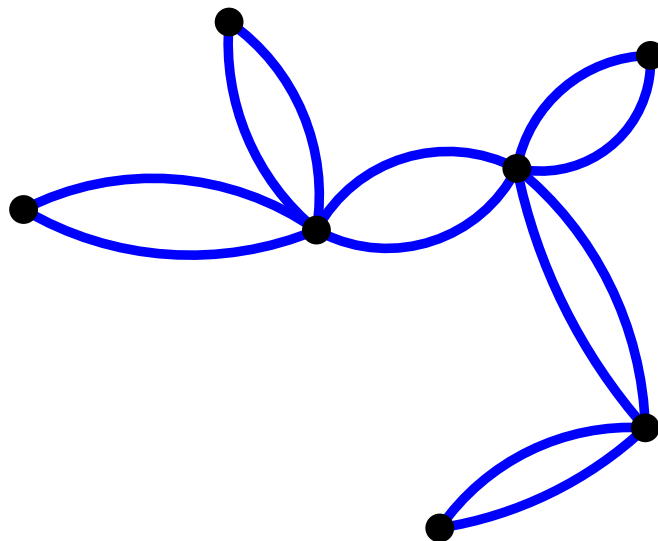


Gesucht: Hamiltonkreis in  $G$  mit minimalen Kosten.

**Satz.**

Es gibt eine 2-Approximation für  $\Delta$ -TSP.

*Beweis.*



*Algorithmus:*

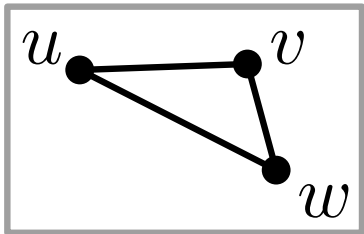
Berechne min. Spannbaum **MSB**.

Verdopple MSB  $\Rightarrow$  ergibt Kreis!

# Was tun? – Mach das Problem leichter!

**Problem:** *Metrisches Traveling Salesman Problem* ( $\Delta$ -TSP)

Gegeben: unger. vollständiger Graph  $G = (V, E)$   
 mit Kantenkosten  $c: E \rightarrow \mathbb{R}_{\geq 0}$ ,  
 die die Dreiecksungleichung erfüllen,  
 d.h.  $\forall u, v, w \in V: c(u, w) \leq c(u, v) + c(v, w)$ .

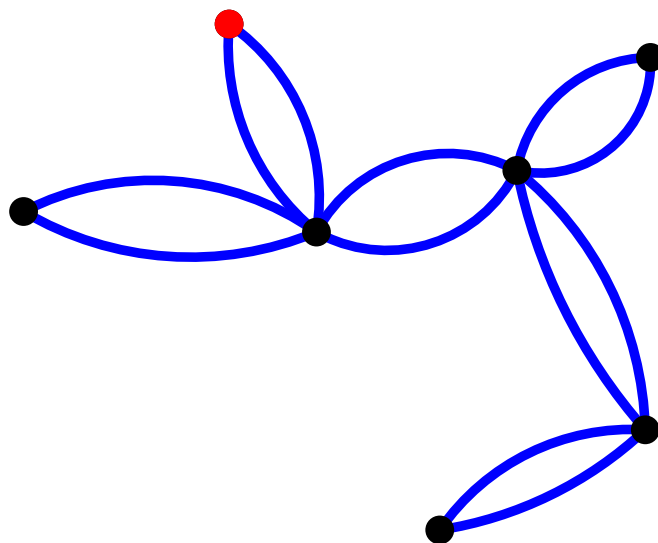


Gesucht: Hamiltonkreis in  $G$  mit minimalen Kosten.

**Satz.**

Es gibt eine 2-Approximation für  $\Delta$ -TSP.

*Beweis.*



*Algorithmus:*

Berechne min. Spannbaum **MSB**.

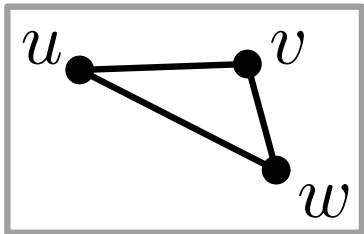
Verdopple MSB  $\Rightarrow$  ergibt Kreis!

Durchlaufe den **Kreis**.

# Was tun? – Mach das Problem leichter!

**Problem:** *Metrisches Traveling Salesman Problem* ( $\Delta$ -TSP)

Gegeben: unger. vollständiger Graph  $G = (V, E)$   
 mit Kantenkosten  $c: E \rightarrow \mathbb{R}_{\geq 0}$ ,  
 die die Dreiecksungleichung erfüllen,  
 d.h.  $\forall u, v, w \in V: c(u, w) \leq c(u, v) + c(v, w)$ .

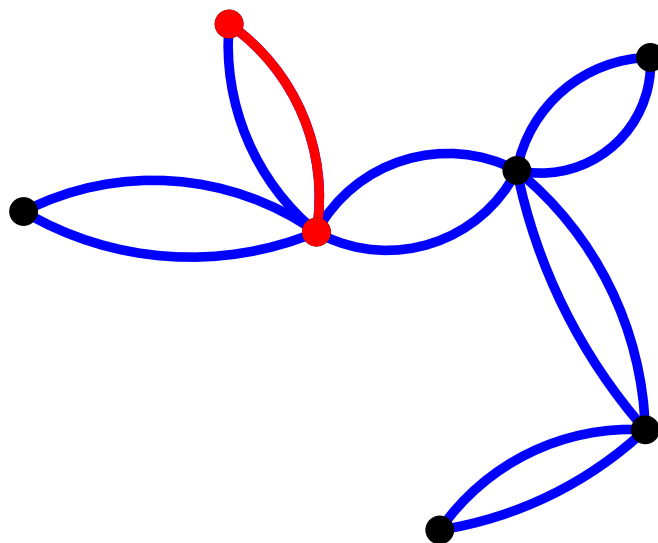


Gesucht: Hamiltonkreis in  $G$  mit minimalen Kosten.

**Satz.**

Es gibt eine 2-Approximation für  $\Delta$ -TSP.

*Beweis.*



*Algorithmus:*

Berechne min. Spannbaum **MSB**.

Verdopple MSB  $\Rightarrow$  ergibt Kreis!

Durchlaufe den **Kreis**.



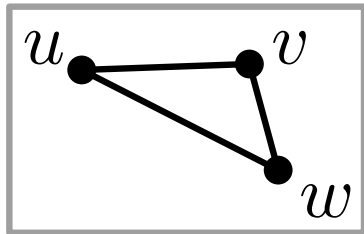
# Was tun? – Mach das Problem leichter!

**Problem:** *Metrisches Traveling Salesman Problem* ( $\Delta$ -TSP)

Gegeben: unger. vollständiger Graph  $G = (V, E)$

mit Kantenkosten  $c: E \rightarrow \mathbb{R}_{\geq 0}$ ,  
die die Dreiecksungleichung erfüllen,

d.h.  $\forall u, v, w \in V: c(u, w) \leq c(u, v) + c(v, w)$ .

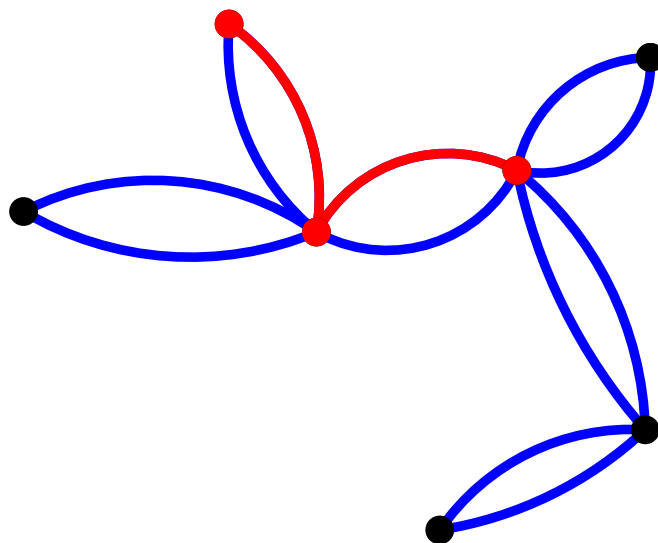


Gesucht: Hamiltonkreis in  $G$  mit minimalen Kosten.

**Satz.**

Es gibt eine 2-Approximation für  $\Delta$ -TSP.

*Beweis.*



*Algorithmus:*

Berechne min. Spannbaum **MSB**.

Verdopple MSB  $\Rightarrow$  ergibt Kreis!

Durchlaufe den **Kreis**.

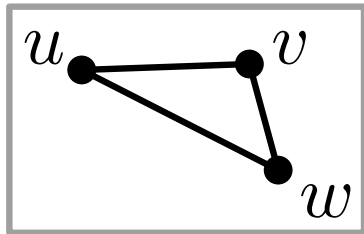
# Was tun? – Mach das Problem leichter!

**Problem:** *Metrisches Traveling Salesman Problem* ( $\Delta$ -TSP)

Gegeben: unger. vollständiger Graph  $G = (V, E)$

mit Kantenkosten  $c: E \rightarrow \mathbb{R}_{\geq 0}$ ,  
die die Dreiecksungleichung erfüllen,

d.h.  $\forall u, v, w \in V: c(u, w) \leq c(u, v) + c(v, w)$ .

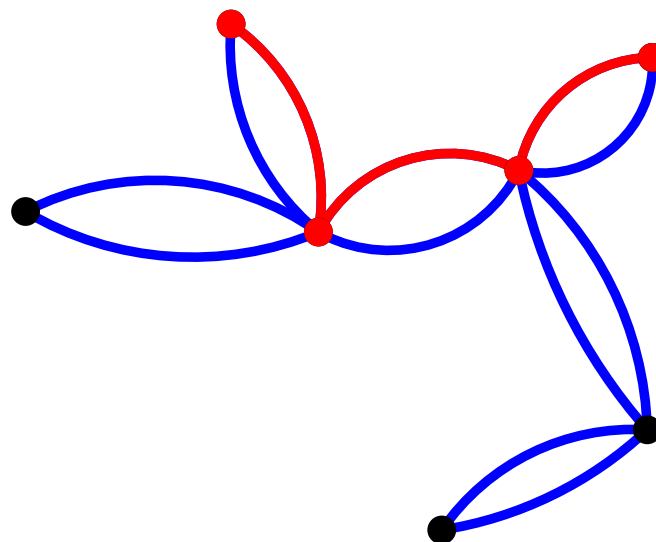


Gesucht: Hamiltonkreis in  $G$  mit minimalen Kosten.

**Satz.**

Es gibt eine 2-Approximation für  $\Delta$ -TSP.

*Beweis.*



*Algorithmus:*

Berechne min. Spannbaum **MSB**.

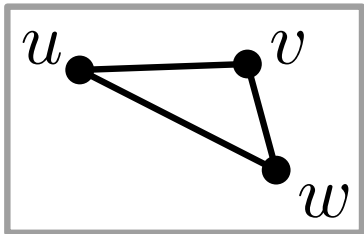
Verdopple MSB  $\Rightarrow$  ergibt Kreis!

Durchlaufe den **Kreis**.

# Was tun? – Mach das Problem leichter!

**Problem:** *Metrisches Traveling Salesman Problem* ( $\Delta$ -TSP)

Gegeben: unger. vollständiger Graph  $G = (V, E)$   
 mit Kantenkosten  $c: E \rightarrow \mathbb{R}_{\geq 0}$ ,  
 die die Dreiecksungleichung erfüllen,  
 d.h.  $\forall u, v, w \in V: c(u, w) \leq c(u, v) + c(v, w)$ .

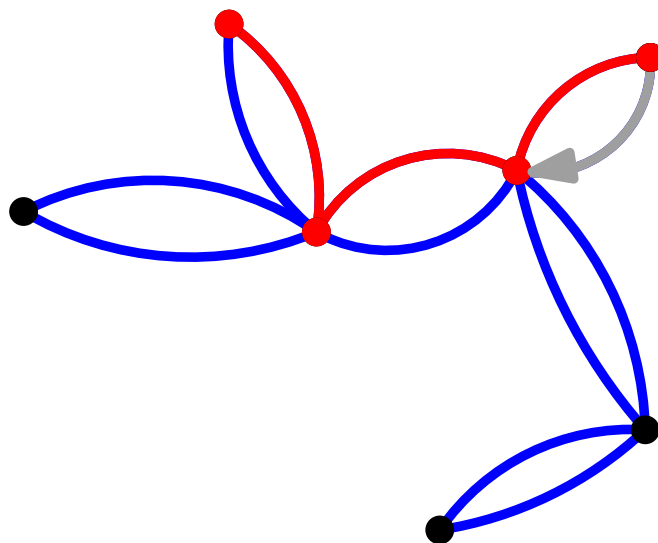


Gesucht: Hamiltonkreis in  $G$  mit minimalen Kosten.

**Satz.**

Es gibt eine 2-Approximation für  $\Delta$ -TSP.

*Beweis.*



*Algorithmus:*

Berechne min. Spannbaum **MSB**.

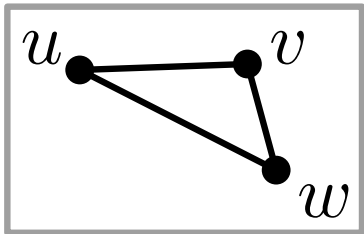
Verdopple MSB  $\Rightarrow$  ergibt Kreis!

Durchlaufe den **Kreis**.

# Was tun? – Mach das Problem leichter!

**Problem:** *Metrisches Traveling Salesman Problem* ( $\Delta$ -TSP)

Gegeben: unger. vollständiger Graph  $G = (V, E)$   
 mit Kantenkosten  $c: E \rightarrow \mathbb{R}_{\geq 0}$ ,  
 die die Dreiecksungleichung erfüllen,  
 d.h.  $\forall u, v, w \in V: c(u, w) \leq c(u, v) + c(v, w)$ .

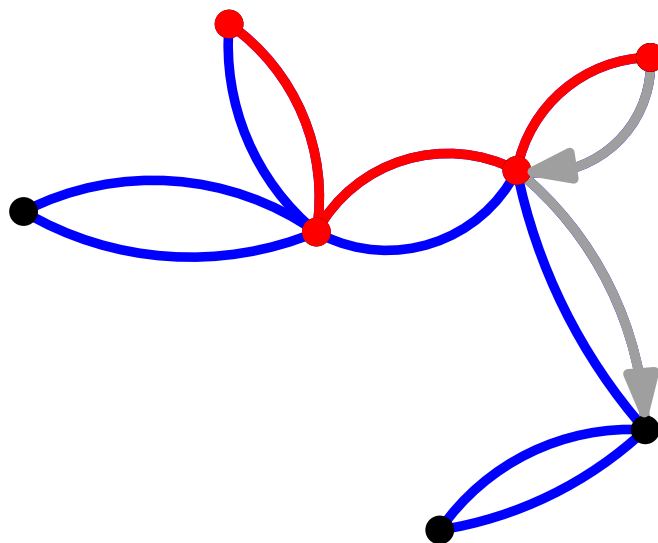


Gesucht: Hamiltonkreis in  $G$  mit minimalen Kosten.

**Satz.**

Es gibt eine 2-Approximation für  $\Delta$ -TSP.

*Beweis.*



*Algorithmus:*

Berechne min. Spannbaum **MSB**.

Verdopple MSB  $\Rightarrow$  ergibt Kreis!

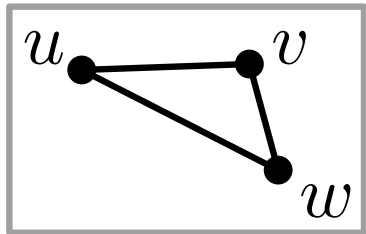
Durchlaufe den **Kreis**.

Überspringe besuchte Knoten.

# Was tun? – Mach das Problem leichter!

**Problem:** *Metrisches Traveling Salesman Problem* ( $\Delta$ -TSP)

Gegeben: unger. vollständiger Graph  $G = (V, E)$   
 mit Kantenkosten  $c: E \rightarrow \mathbb{R}_{\geq 0}$ ,  
 die die Dreiecksungleichung erfüllen,  
 d.h.  $\forall u, v, w \in V: c(u, w) \leq c(u, v) + c(v, w)$ .

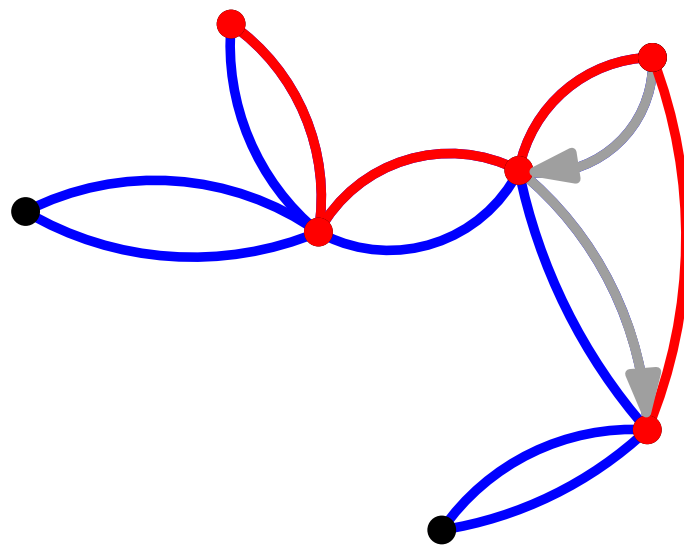


Gesucht: Hamiltonkreis in  $G$  mit minimalen Kosten.

**Satz.**

Es gibt eine 2-Approximation für  $\Delta$ -TSP.

*Beweis.*



*Algorithmus:*

Berechne min. Spannbaum **MSB**.

Verdopple MSB  $\Rightarrow$  ergibt Kreis!

Durchlaufe den **Kreis**.

Überspringe besuchte Knoten.

Füge „Abkürzungen“ ein.

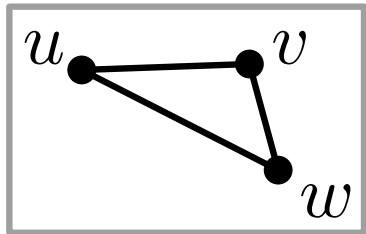
# Was tun? – Mach das Problem leichter!

**Problem:** *Metrisches Traveling Salesman Problem* ( $\Delta$ -TSP)

Gegeben: unger. vollständiger Graph  $G = (V, E)$

mit Kantenkosten  $c: E \rightarrow \mathbb{R}_{\geq 0}$ ,  
die die Dreiecksungleichung erfüllen,

d.h.  $\forall u, v, w \in V: c(u, w) \leq c(u, v) + c(v, w)$ .

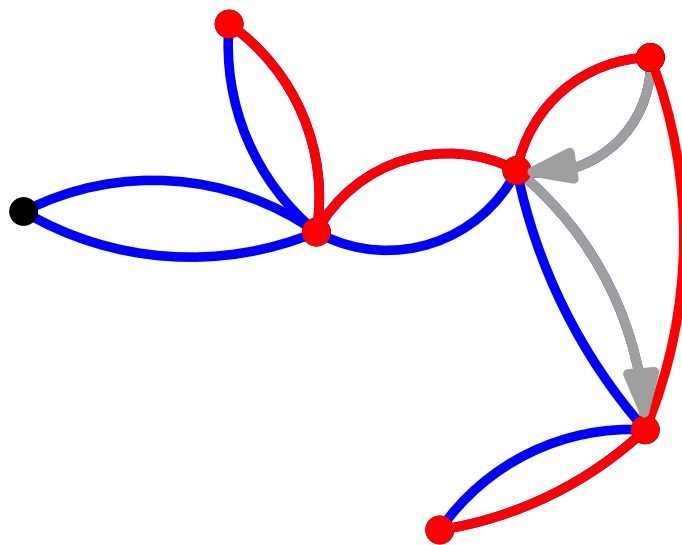


Gesucht: Hamiltonkreis in  $G$  mit minimalen Kosten.

**Satz.**

Es gibt eine 2-Approximation für  $\Delta$ -TSP.

*Beweis.*



*Algorithmus:*

Berechne min. Spannbaum **MSB**.

Verdopple MSB  $\Rightarrow$  ergibt Kreis!

Durchlaufe den **Kreis**.

Überspringe besuchte Knoten.

Füge „Abkürzungen“ ein.

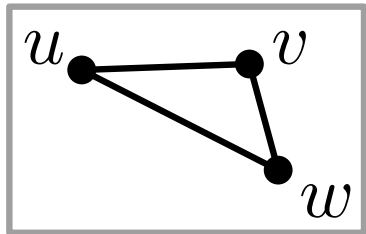
# Was tun? – Mach das Problem leichter!

**Problem:** *Metrisches Traveling Salesman Problem* ( $\Delta$ -TSP)

Gegeben: unger. vollständiger Graph  $G = (V, E)$

mit Kantenkosten  $c: E \rightarrow \mathbb{R}_{\geq 0}$ ,  
die die Dreiecksungleichung erfüllen,

d.h.  $\forall u, v, w \in V: c(u, w) \leq c(u, v) + c(v, w)$ .

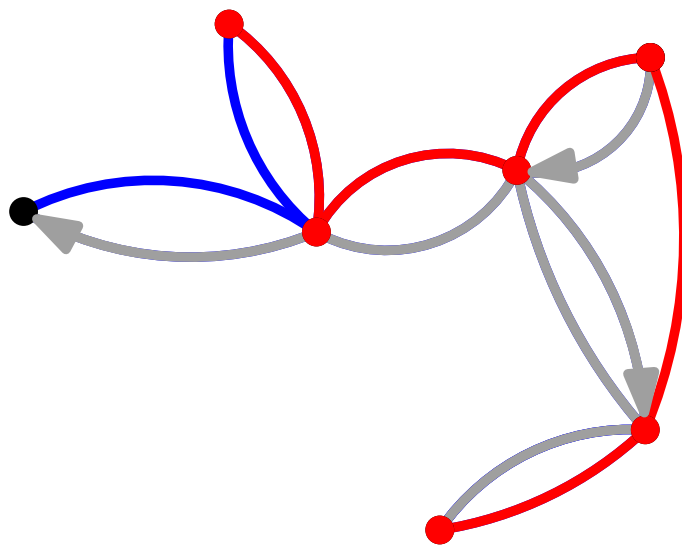


Gesucht: Hamiltonkreis in  $G$  mit minimalen Kosten.

**Satz.**

Es gibt eine 2-Approximation für  $\Delta$ -TSP.

*Beweis.*



*Algorithmus:*

Berechne min. Spannbaum **MSB**.

Verdopple MSB  $\Rightarrow$  ergibt Kreis!

Durchlaufe den **Kreis**.

Überspringe besuchte Knoten.

Füge „Abkürzungen“ ein.

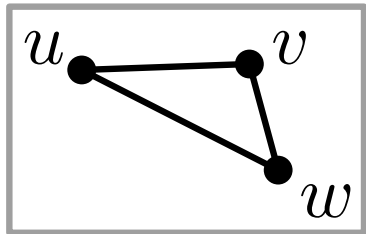
# Was tun? – Mach das Problem leichter!

**Problem:** *Metrisches Traveling Salesman Problem* ( $\Delta$ -TSP)

Gegeben: unger. vollständiger Graph  $G = (V, E)$

mit Kantenkosten  $c: E \rightarrow \mathbb{R}_{\geq 0}$ ,  
die die Dreiecksungleichung erfüllen,

d.h.  $\forall u, v, w \in V: c(u, w) \leq c(u, v) + c(v, w)$ .

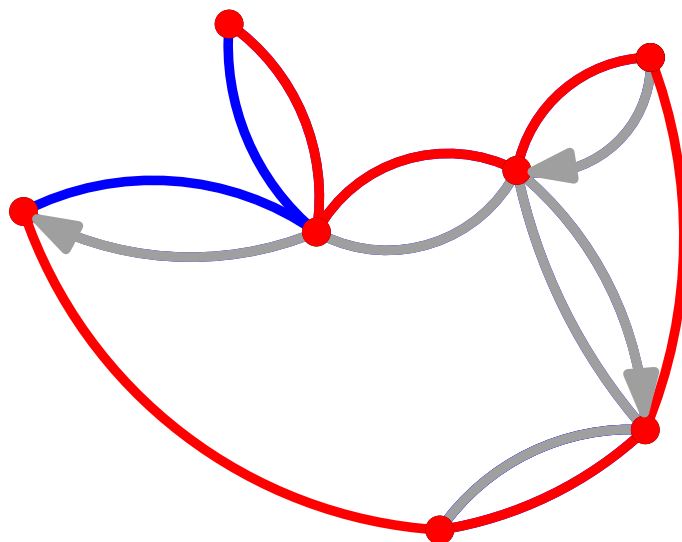


Gesucht: Hamiltonkreis in  $G$  mit minimalen Kosten.

**Satz.**

Es gibt eine 2-Approximation für  $\Delta$ -TSP.

*Beweis.*



*Algorithmus:*

Berechne min. Spannbaum **MSB**.

Verdopple MSB  $\Rightarrow$  ergibt Kreis!

Durchlaufe den **Kreis**.

Überspringe besuchte Knoten.

Füge „Abkürzungen“ ein.



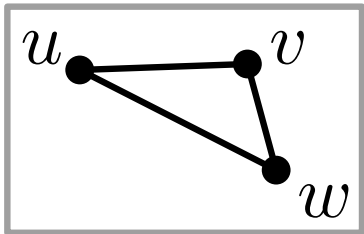
# Was tun? – Mach das Problem leichter!

**Problem:** *Metrisches Traveling Salesman Problem* ( $\Delta$ -TSP)

Gegeben: unger. vollständiger Graph  $G = (V, E)$

mit Kantenkosten  $c: E \rightarrow \mathbb{R}_{\geq 0}$ ,  
die die Dreiecksungleichung erfüllen,

d.h.  $\forall u, v, w \in V: c(u, w) \leq c(u, v) + c(v, w)$ .

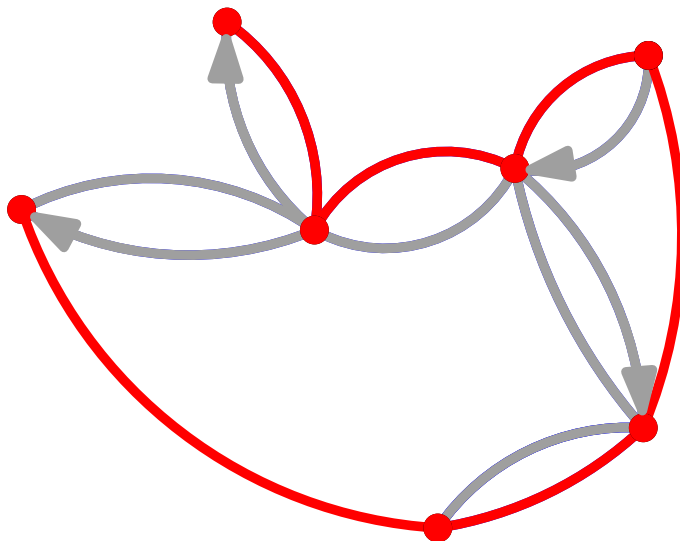


Gesucht: Hamiltonkreis in  $G$  mit minimalen Kosten.

**Satz.**

Es gibt eine 2-Approximation für  $\Delta$ -TSP.

*Beweis.*



*Algorithmus:*

Berechne min. Spannbaum **MSB**.

Verdopple MSB  $\Rightarrow$  ergibt Kreis!

Durchlaufe den **Kreis**.

Überspringe besuchte Knoten.

Füge „Abkürzungen“ ein.

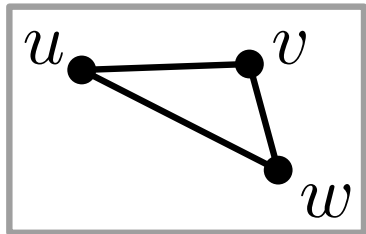
# Was tun? – Mach das Problem leichter!

**Problem:** *Metrisches Traveling Salesman Problem* ( $\Delta$ -TSP)

Gegeben: unger. vollständiger Graph  $G = (V, E)$

mit Kantenkosten  $c: E \rightarrow \mathbb{R}_{\geq 0}$ ,  
die die Dreiecksungleichung erfüllen,

d.h.  $\forall u, v, w \in V: c(u, w) \leq c(u, v) + c(v, w)$ .

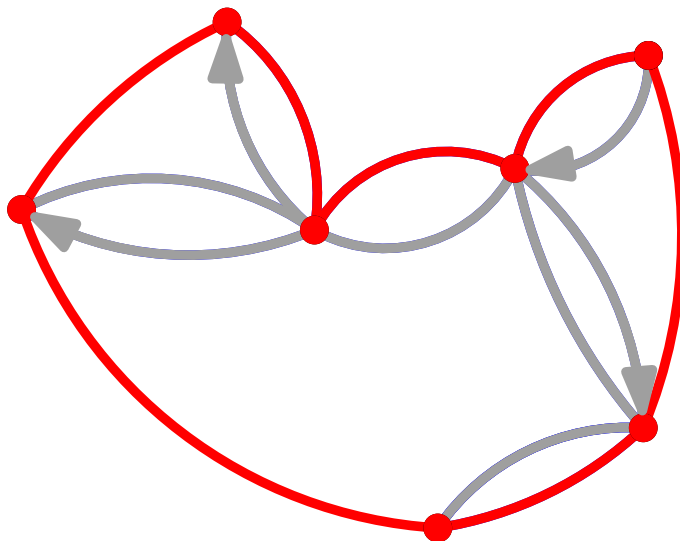


Gesucht: Hamiltonkreis in  $G$  mit minimalen Kosten.

**Satz.**

Es gibt eine 2-Approximation für  $\Delta$ -TSP.

*Beweis.*



*Algorithmus:*

Berechne min. Spannbaum **MSB**.

Verdopple MSB  $\Rightarrow$  ergibt Kreis!

Durchlaufe den **Kreis**.

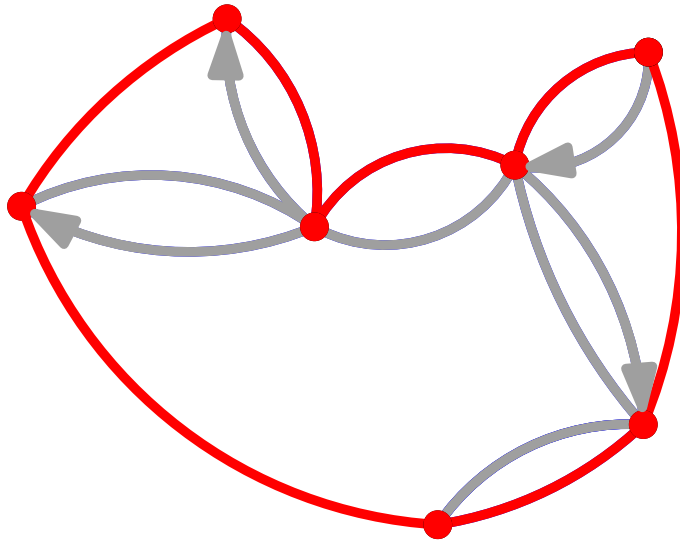
Überspringe besuchte Knoten.

Füge „Abkürzungen“ ein.

# Analyse

**Satz.** Es gibt eine 2-Approximation für  $\Delta$ -TSP.

*Beweis.*



## 1. Algorithmus

Berechne **MSB** von  $G$ .

Verdopple MSB  $\Rightarrow$  ergibt Kreis!

Durchlaufe den **Kreis**.

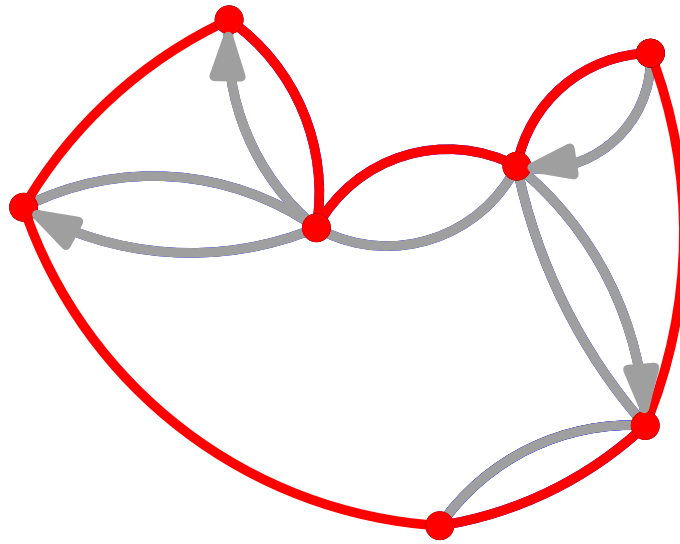
Überspringe besuchte Knoten.

Füge „Abkürzungen“ ein.

# Analyse

**Satz.** Es gibt eine 2-Approximation für  $\Delta$ -TSP.

*Beweis.*



## 1. Algorithmus

Berechne **MSB** von  $G$ .

Verdopple MSB  $\Rightarrow$  ergibt Kreis!

Durchlaufe den **Kreis**.

Überspringe besuchte Knoten.

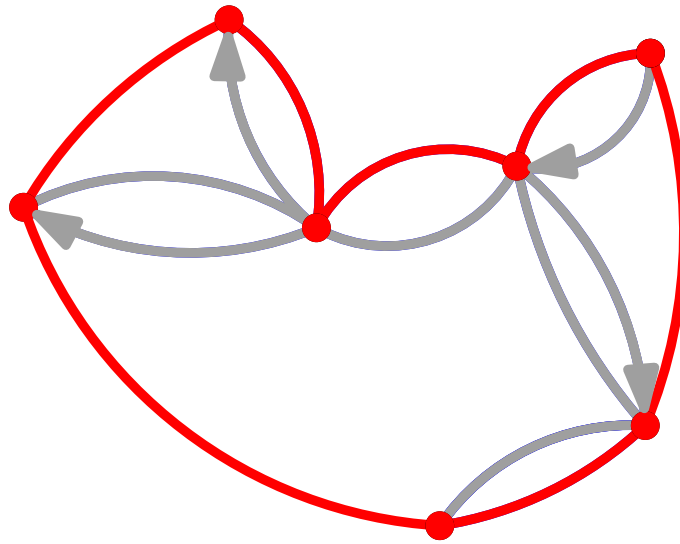
Füge „Abkürzungen“ ein.

## 2. Analyse

# Analyse

**Satz.** Es gibt eine 2-Approximation für  $\Delta$ -TSP.

*Beweis.*



## 1. Algorithmus

Berechne **MSB** von  $G$ .

Verdopple MSB  $\Rightarrow$  ergibt Kreis!

Durchlaufe den **Kreis**.

Überspringe besuchte Knoten.

Füge „Abkürzungen“ ein.

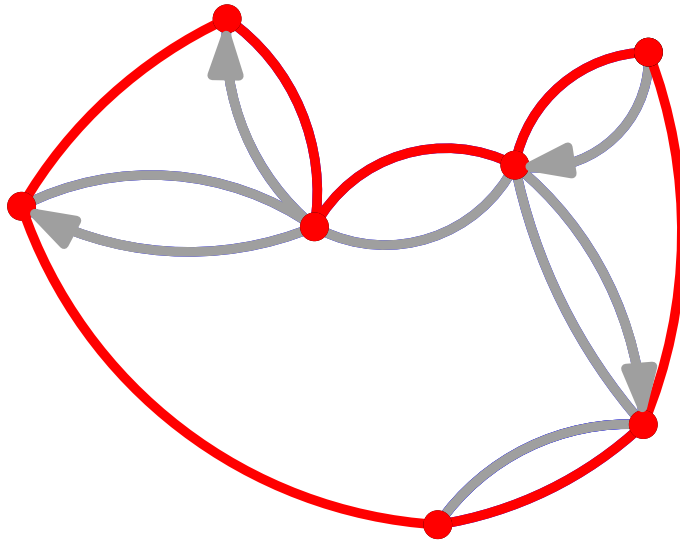
## 2. Analyse

$$c(\text{ALG}) \leq$$

# Analyse

**Satz.** Es gibt eine 2-Approximation für  $\Delta$ -TSP.

*Beweis.*



## 1. Algorithmus

Berechne **MSB** von  $G$ .

Verdopple MSB  $\Rightarrow$  ergibt Kreis!

Durchlaufe den **Kreis**.

Überspringe besuchte Knoten.

Füge „Abkürzungen“ ein.

## 2. Analyse

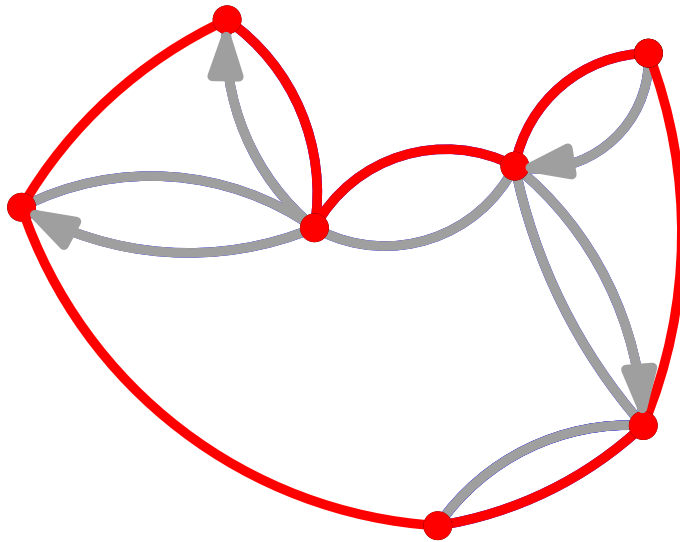
$$c(\text{ALG}) \leq$$

Dreiecksungleichung

# Analyse

**Satz.** Es gibt eine 2-Approximation für  $\Delta$ -TSP.

*Beweis.*



## 1. Algorithmus

Berechne **MSB** von  $G$ .

Verdopple MSB  $\Rightarrow$  ergibt Kreis!

Durchlaufe den **Kreis**.

Überspringe besuchte Knoten.

Füge „Abkürzungen“ ein.

## 2. Analyse

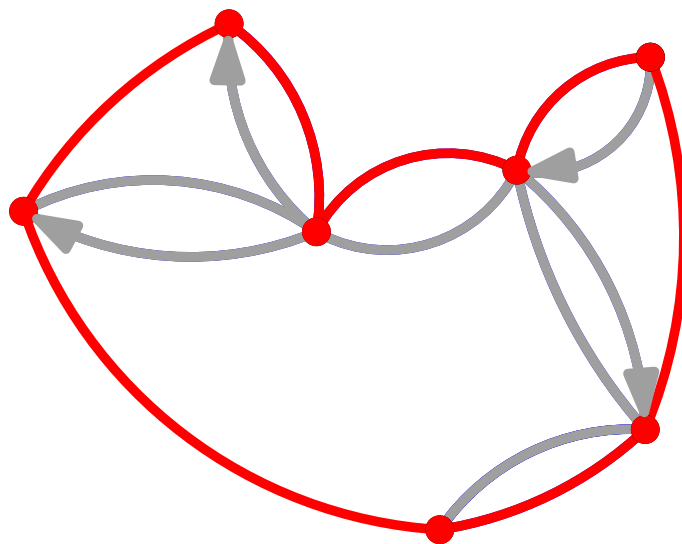
$$c(\text{ALG}) \leq c(\text{Kreis}) =$$

Dreiecksungleichung

# Analyse

**Satz.** Es gibt eine 2-Approximation für  $\Delta$ -TSP.

*Beweis.*



## 1. Algorithmus

Berechne **MSB** von  $G$ .

Verdopple MSB  $\Rightarrow$  ergibt Kreis!

Durchlaufe den **Kreis**.

Überspringe besuchte Knoten.

Füge „Abkürzungen“ ein.

## 2. Analyse

$$c(\text{ALG}) \leq c(\text{Kreis}) = 2 \cdot c(\text{MSB}) \leq$$

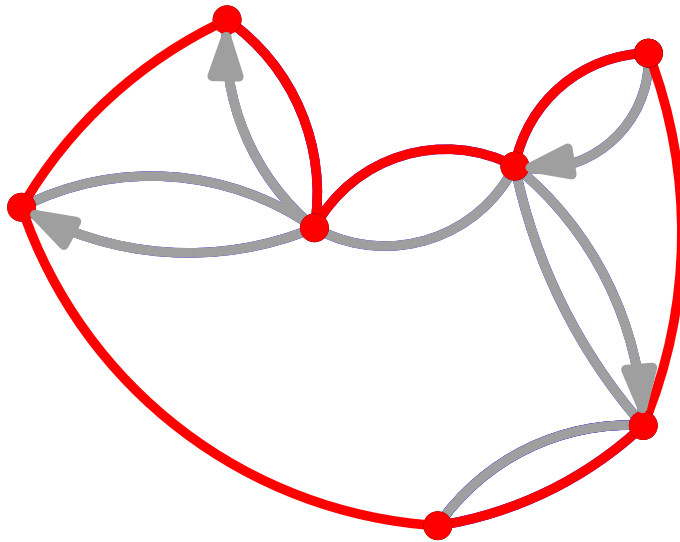
Dreiecksungleichung



# Analyse

**Satz.** Es gibt eine 2-Approximation für  $\Delta$ -TSP.

*Beweis.*



## 1. Algorithmus

Berechne **MSB** von  $G$ .

Verdopple MSB  $\Rightarrow$  ergibt Kreis!

Durchlaufe den **Kreis**.

Überspringe besuchte Knoten.

Füge „Abkürzungen“ ein.

## 2. Analyse

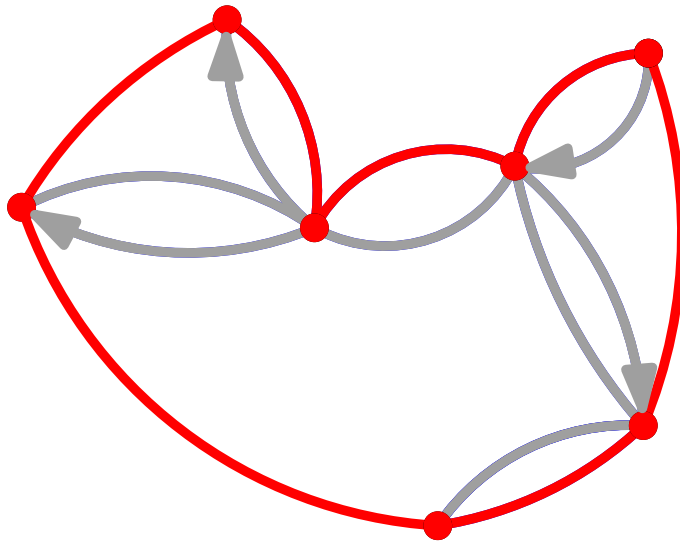
$$c(\text{ALG}) \leq c(\text{Kreis}) = 2 \cdot c(\text{MSB}) \leq 2 \cdot \text{[ ]}$$

Dreiecksungleichung

# Analyse

**Satz.** Es gibt eine 2-Approximation für  $\Delta$ -TSP.

*Beweis.*



## 1. Algorithmus

Berechne **MSB** von  $G$ .

Verdopple MSB  $\Rightarrow$  ergibt Kreis!

Durchlaufe den **Kreis**.

Überspringe besuchte Knoten.

Füge „Abkürzungen“ ein.

## 2. Analyse

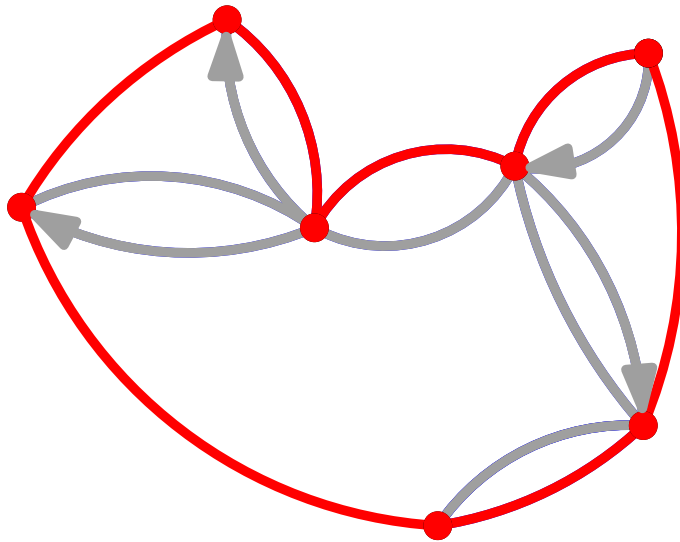
$$c(\text{ALG}) \leq c(\text{Kreis}) = 2 \cdot c(\text{MSB}) \leq 2 \cdot \text{OPT}$$

Dreiecksungleichung

# Analyse

**Satz.** Es gibt eine 2-Approximation für  $\Delta$ -TSP.

*Beweis.*



## 1. Algorithmus

Berechne **MSB** von  $G$ .

Verdopple MSB  $\Rightarrow$  ergibt Kreis!

Durchlaufe den **Kreis**.

Überspringe besuchte Knoten.

Füge „Abkürzungen“ ein.

## 2. Analyse

$$c(\text{ALG}) \leq c(\text{Kreis}) = 2 \cdot c(\text{MSB}) \leq 2 \cdot \text{OPT}$$

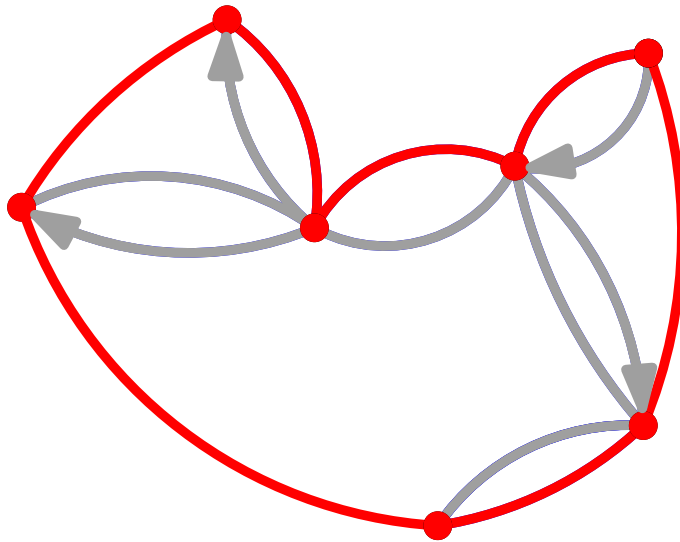
Dreiecksungleichung

Optimale TSP-Tour minus eine Kante ist (i.A. nicht minimaler) Spannbaum!!

# Analyse

**Satz.** Es gibt eine 2-Approximation für  $\Delta$ -TSP.

*Beweis.*



## 1. Algorithmus

Berechne **MSB** von  $G$ .

Verdopple MSB  $\Rightarrow$  ergibt Kreis!

Durchlaufe den **Kreis**.

Überspringe besuchte Knoten.

Füge „Abkürzungen“ ein.

## 2. Analyse

$$c(\text{ALG}) \leq c(\text{Kreis}) = 2 \cdot c(\text{MSB}) \leq 2 \cdot \text{OPT}$$

Dreiecksungleichung

Optimale TSP-Tour minus eine Kante ist (i.A. nicht minimaler) Spannbaum!!

*Die „Kunst“ der unteren Schranke:*  $c(\text{min. Spannbaum}) \leq c(\text{TSP-Tour})$