

Kap. 4: Das
Handlungsreisendenproblem
(TSP)
4.3 Branch & Cut
4.4 Engineering Lin-Kernighan

Professor Dr. Petra Mutzel

Lehrstuhl für Algorithm Engineering, LS11

8. VO Draft: noch nicht endgültig 26. April 2007

Überblick

- 4.1 Einführung
 - Einführung in TSP
- 4.2 ILP-Formulierung für TSP
- 4.3 Branch-and-Cut Algorithmus
 - Separierung der Subtour Bedingungen
 - Zusätzliche Ungleichungen
 - Branch & Cut
 - Spaltengenerierung
- 4.4 Engineering Lin-Kernighan

4.3 Branch & Cut für TSP

LP-Relaxierung: Entferne Ganzzahligkeitsbedingungen

minimiere $\sum_{e \in E} c_e x_e$

so dass $0 \leq x_e \leq 1$ für alle $e \in E$

~~x_e ganzzahlig für alle $e \in E$~~

$\sum_{e \in \delta(v)} x_e = 2$ für alle $v \in V$

$\sum_{e \in \delta(W)} x_e \geq 2$ für alle $\emptyset \neq W \subset V$

Subtour-Relaxierung: man relaxiert Ganzzahligkeit

Branch & Cut für TSP

Auch die Subtour-Relaxierung hat bereits exponentiell viele ($\Theta(2^{n-1})$) Ungleichungen

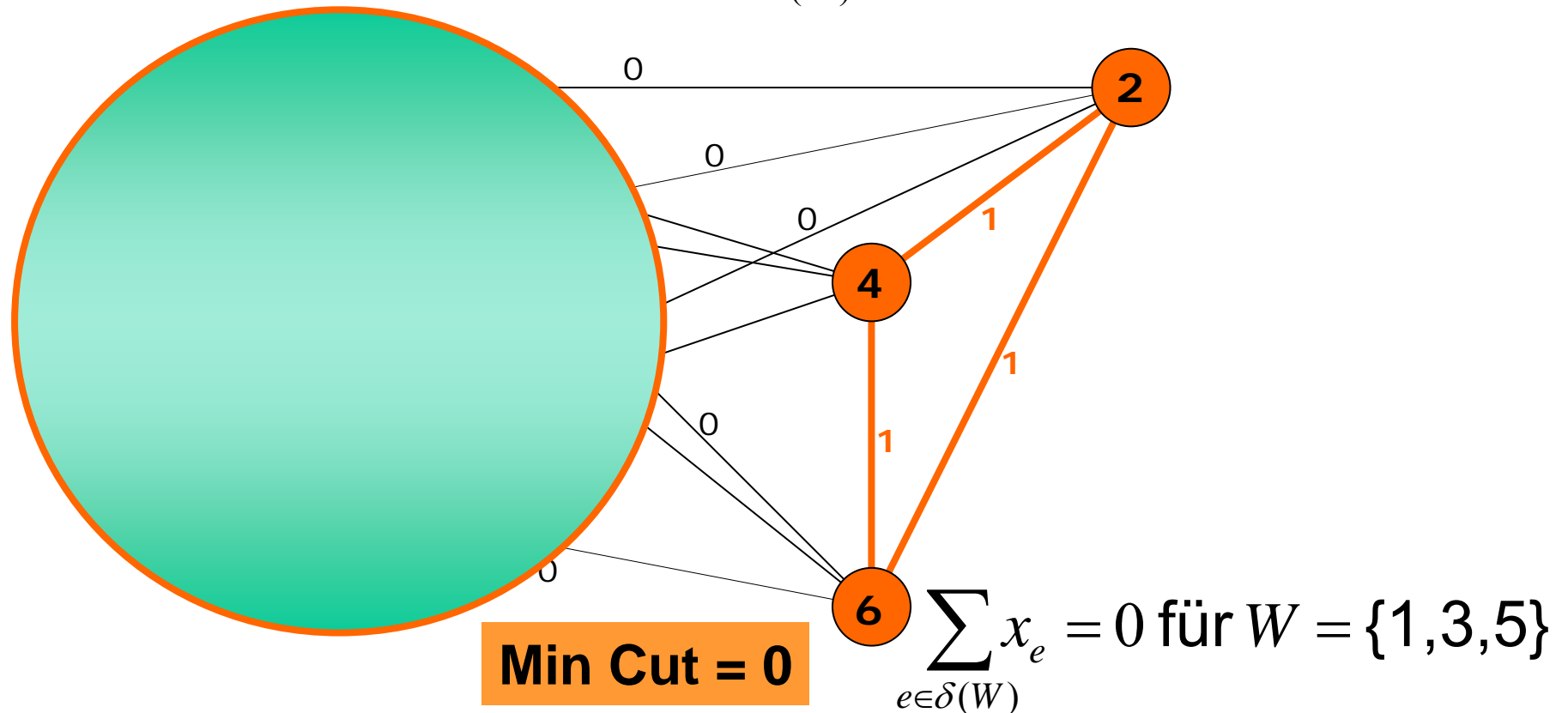
- hoffnungslos?
- Lösung: Schnittebenenverfahren

Können wir die Subtour-Relaxierung beim TSP lösen?

Berechnen eines minimalen Schnittes in $G=(V,E)$ mit Kantenkosten x_e (Kanten mit $x_e=0$ werden weggelassen)

Separierung TSP

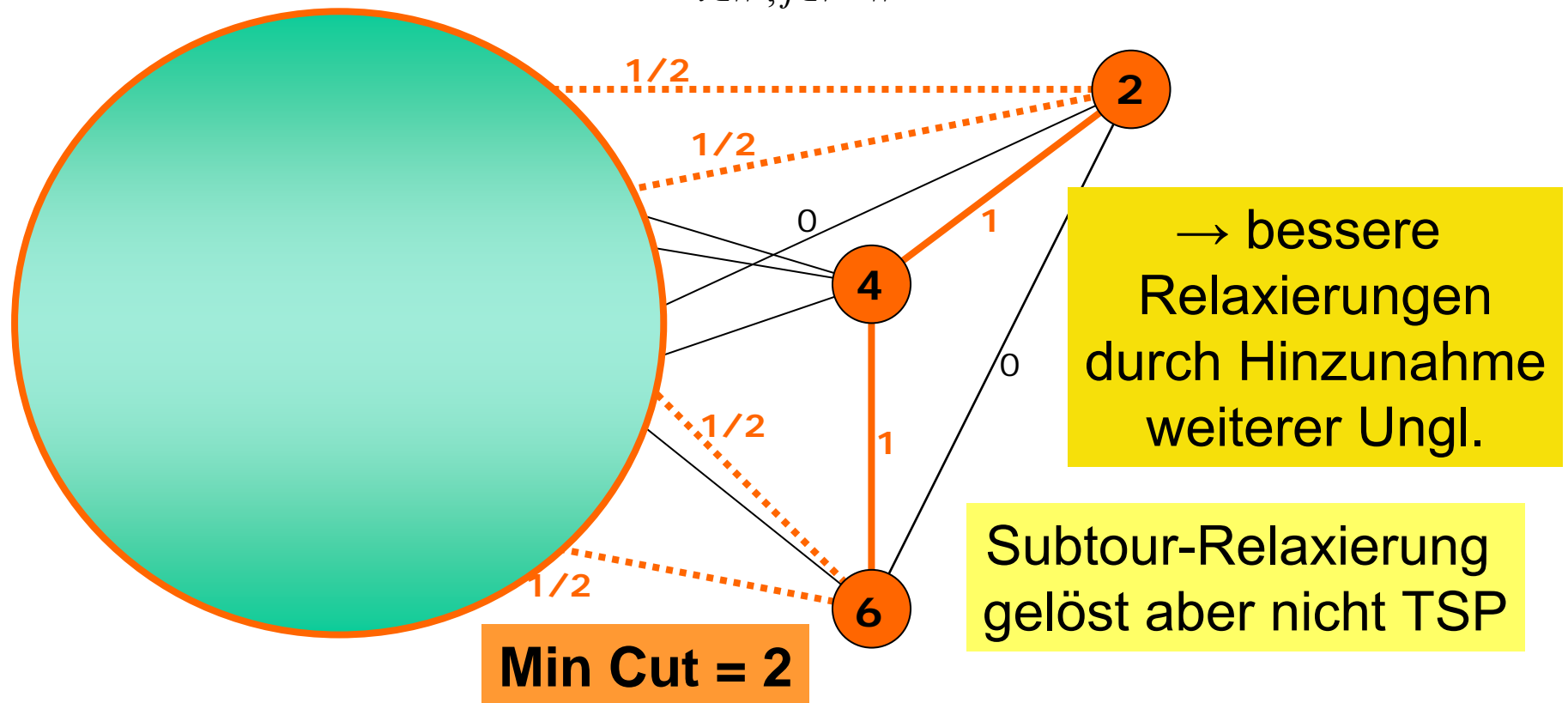
- Subtour Bedingungen $\sum_{e \in \delta(W)} x_e \geq 2$ für alle $\emptyset \neq W \subset V$



Verletzte Subtour-Ungleichung gefunden

Separierung TSP

- Subtour Bedingungen $\sum_{i \in W, j \in V-W} x_{ij} \geq 2$ für alle $\emptyset \neq W \subset V$



Es existiert keine verletzte Subtour-Ungleichung

Alternative Relaxierung

2. Relaxierung: Entferne Subtour-Ungleichungen

minimiere $\sum_{e \in E} c_e x_e$

so dass $0 \leq x_e \leq 1$ für alle $e \in E$

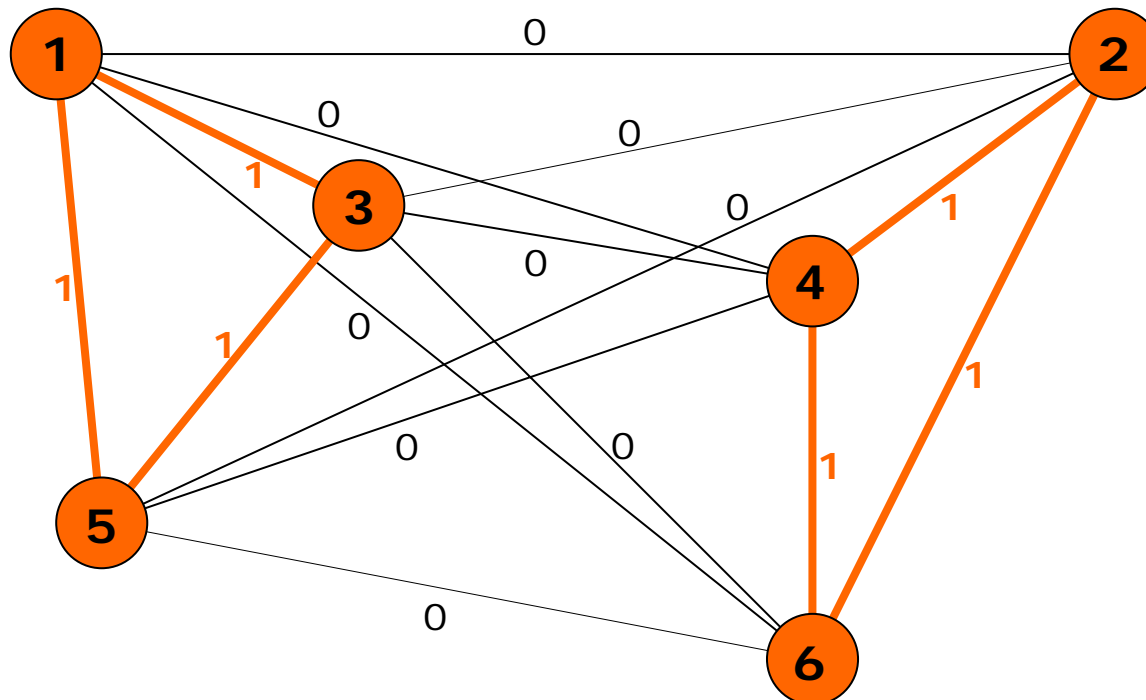
x_e ganzzahlig für alle $e \in E$

$\sum_{e \in \delta(v)} x_e = 2$ für alle $v \in V$

2-Matching Relaxierung

Branch & Cut für TSP

Erhaltene Lösung:



Problem: Lösung ist nicht zusammenhängend

Aber: gibt untere Schranke

Bekannte Ungleichungen beim TSP

- Grad Gleichungen
- Subtour Bedingungen
- 2-Matching Ungleichungen
- Blossom Ungleichungen
- Comb Ungleichungen
- Cliquenbaum Ungleichungen
- Hypergraph Ungleichungen
- ...



Separierungs-
problem
polynomiell

Separierungs-
problem
offen

Wie kommt man auf diese Ungleichungen: Polyedertheorie

oder: Rechnen und dann Studium von nicht-zulässigen Lösungen

Comb-Ungleichungen

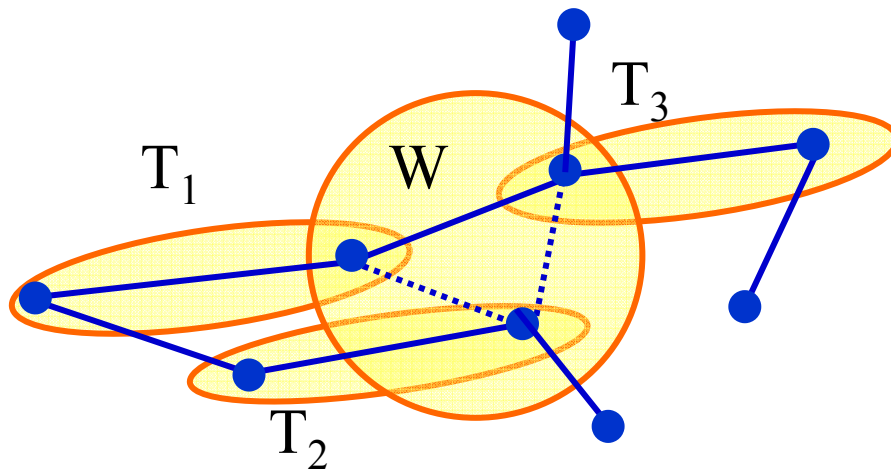
Seien $W, T_1, \dots, T_k \subseteq V$, wobei W der “Griff” (handle) und T_i die “Zinken” (teeth) sind. Es muss gelten:

$$|T_i \cap W| \geq 1 \quad \text{fuer } i = 1, \dots, k, \quad (1)$$

$$|T_i \setminus W| \geq 1 \quad \text{fuer } i = 1, \dots, k \quad (2)$$

$$T_i \cap T_j = \emptyset \quad \text{fuer } 1 \leq i < j \leq k \quad (3)$$

$$k \geq 3 \quad \text{und ungerade.} \quad (4)$$



- “2-Matching Ungleichungen”, wenn für alle i gilt: $|T_i|=2$

Comb-Ungleichungen

- Sei $S = \{W, T_1, \dots, T_k\}$, wobei W der “Griff” (handle) und T_i die “Zinken” (teeth) sind. Es muss gelten:

$$|T_i \cap W| \geq 1 \quad \text{fuer } i = 1, \dots, k, \quad (1)$$

$$|T_i \setminus W| \geq 1 \quad \text{fuer } i = 1, \dots, k \quad (2)$$

$$T_i \cap T_j = \emptyset \quad \text{fuer } 1 \leq i < j \leq k \quad (3)$$

$$k \geq 3 \quad \text{und ungerade.} \quad (4)$$

$$x(E(W)) + \sum_{i=1}^k x(E(T_i)) \leq |W| + \frac{k-1}{2}$$

- “2-Matching Ungleichungen”, wenn für alle i gilt: $|T_i|=2$

2-Matching Ungleichungen

Seien $W, T_1, \dots, T_k \subseteq V$, wobei W der "Griff" (handle) und T_i die "Zinken" (teeth) sind. Es muss gelten:

$$|T_i| = 2 \quad \text{fuer } i = 1, \dots, k, \quad (1)$$

$$|T_i \cap W| \geq 1 \quad \text{fuer } i = 1, \dots, k, \quad (2)$$

$$|T_i \setminus W| \geq 1 \quad \text{fuer } i = 1, \dots, k \quad (3)$$

$$T_i \cap T_j = \emptyset \quad \text{fuer } 1 \leq i < j \leq k \quad (4)$$

$$k \geq 3 \quad \text{und ungerade.} \quad (5)$$

$$x(E(W)) + \sum_{i=1}^k x(E(T_i)) \leq |W| + \frac{k-1}{2}$$

Der Name kommt daher, weil die Ungleichungen das 2-Matching Polytop $\{x \mid A_n x = 2, 0 \leq x \leq 1, x \text{ ganzzahlig}\}$ (mit A_n Inzidenzmatrix) zusammen mit $A_n x = 2$ und $0 \leq x \leq 1$ vollständig beschreiben (Edmonds, 1965).

2-Matching Ungleichungen

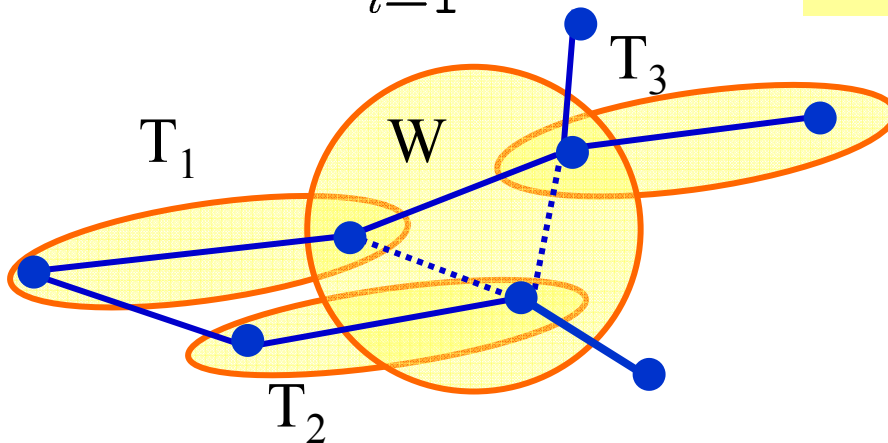
- Die 2-Matching Ungleichungen sind gültig für alle 0/1-Vektoren, die charakteristische Vektoren von Touren sind.

Beweis: Es gilt:

$$2x(E(W)) = \sum_{v \in W} x(\delta(v)) - \sum_{i=1}^k x(E(T_i)) - \sum_{e \in \delta(W) \setminus E(T_i)} x_e$$

$$+ 2 \sum_{i=1}^k x(E(T_i)) \quad + 2 \sum_{i=1}^k x(E(T_i))$$

$$2x(E(W)) + 2 \sum_{i=1}^k x(E(T_i)) = \text{[]} + \sum_{i=1}^k x(E(T_i)) - \sum_{e \in \delta(W) \setminus E(T_i)} x_e$$



$$r.S. \leq 2|W| \text{ []}$$

2-Matching Ungleichungen

- Die 2-Matching Ungleichungen sind gültig für alle 0/1-Vektoren, die charakteristische Vektoren von Touren sind.

$$2x(E(W)) + 2 \sum_{i=1}^k x(E(T_i)) \leq 2|W| + k$$

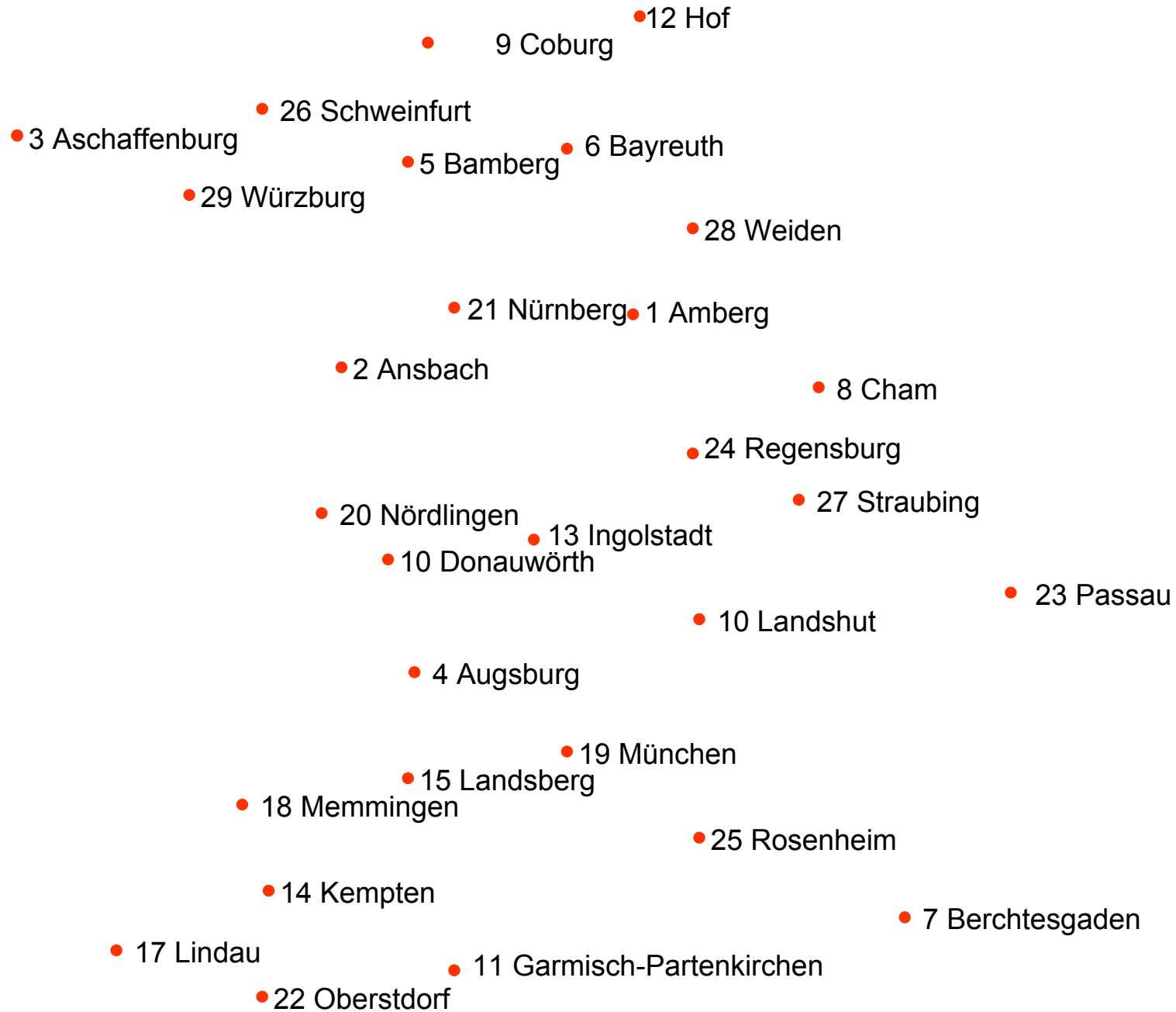
$$x(E(W)) + \sum_{i=1}^k x(E(T_i)) \leq |W| + \frac{k}{2}$$

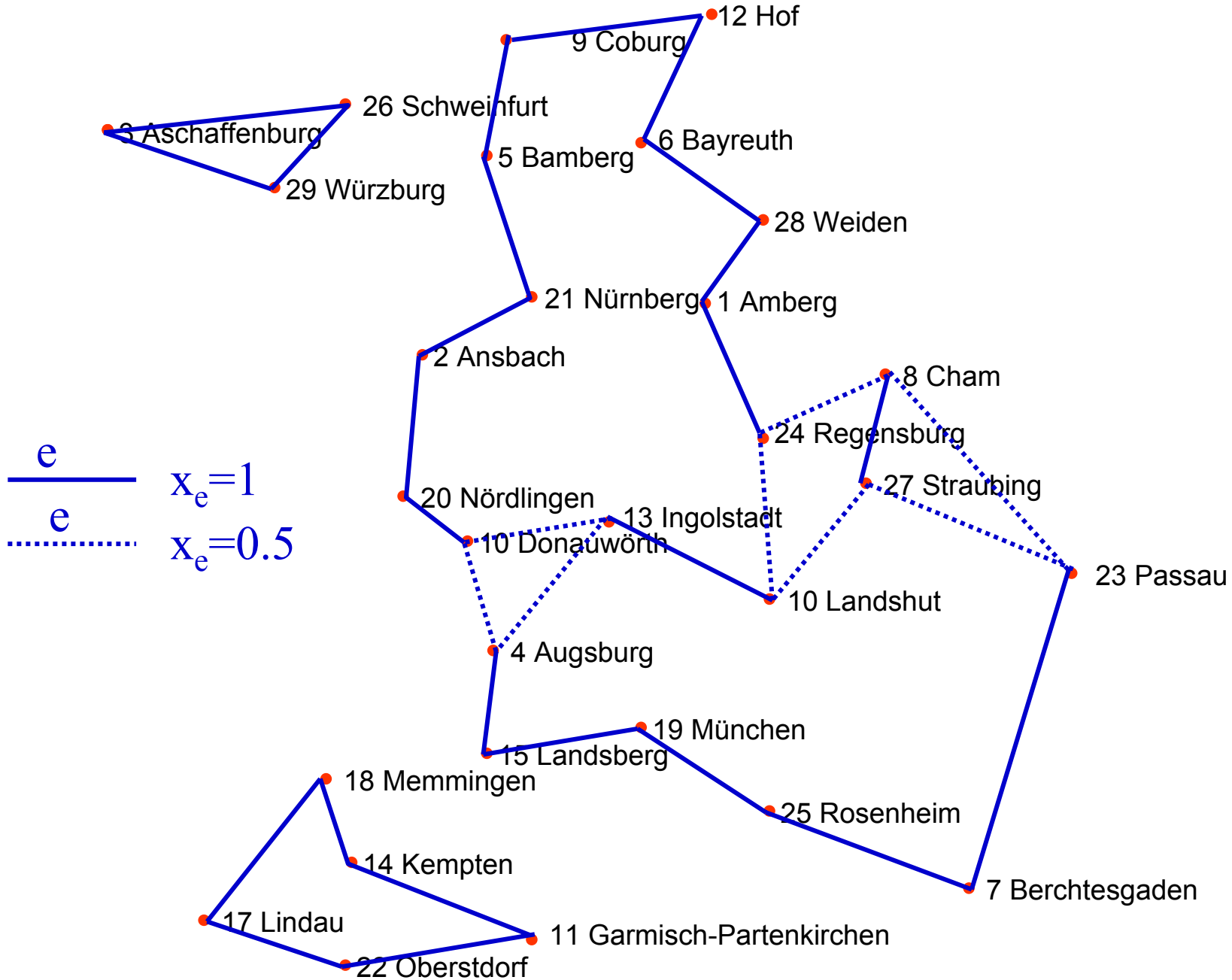
l.S. ganzzahlig \Rightarrow r.S. ganzzahlig

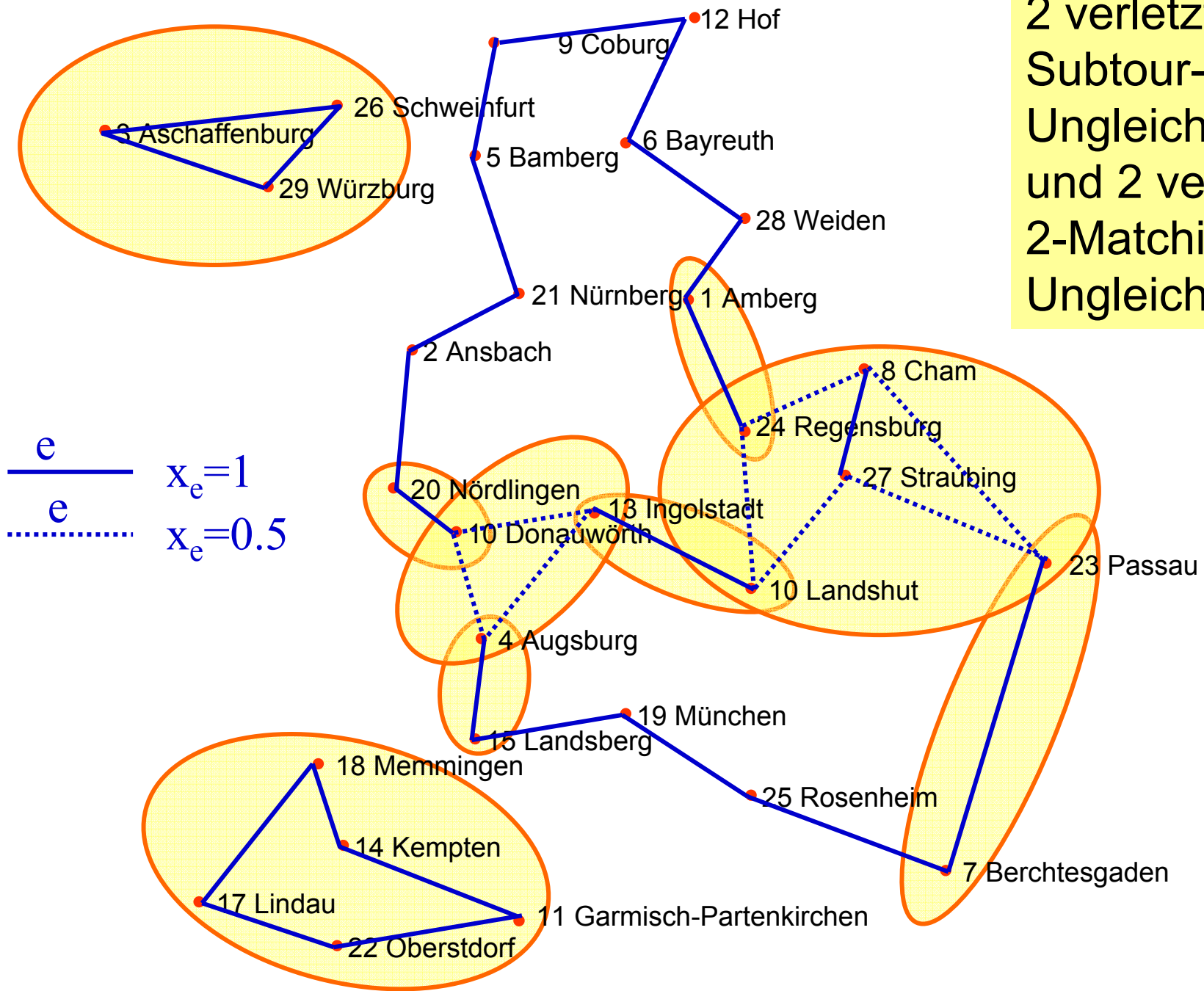
weil k ungerade \Rightarrow $\leq |W| + \frac{k-1}{2}$

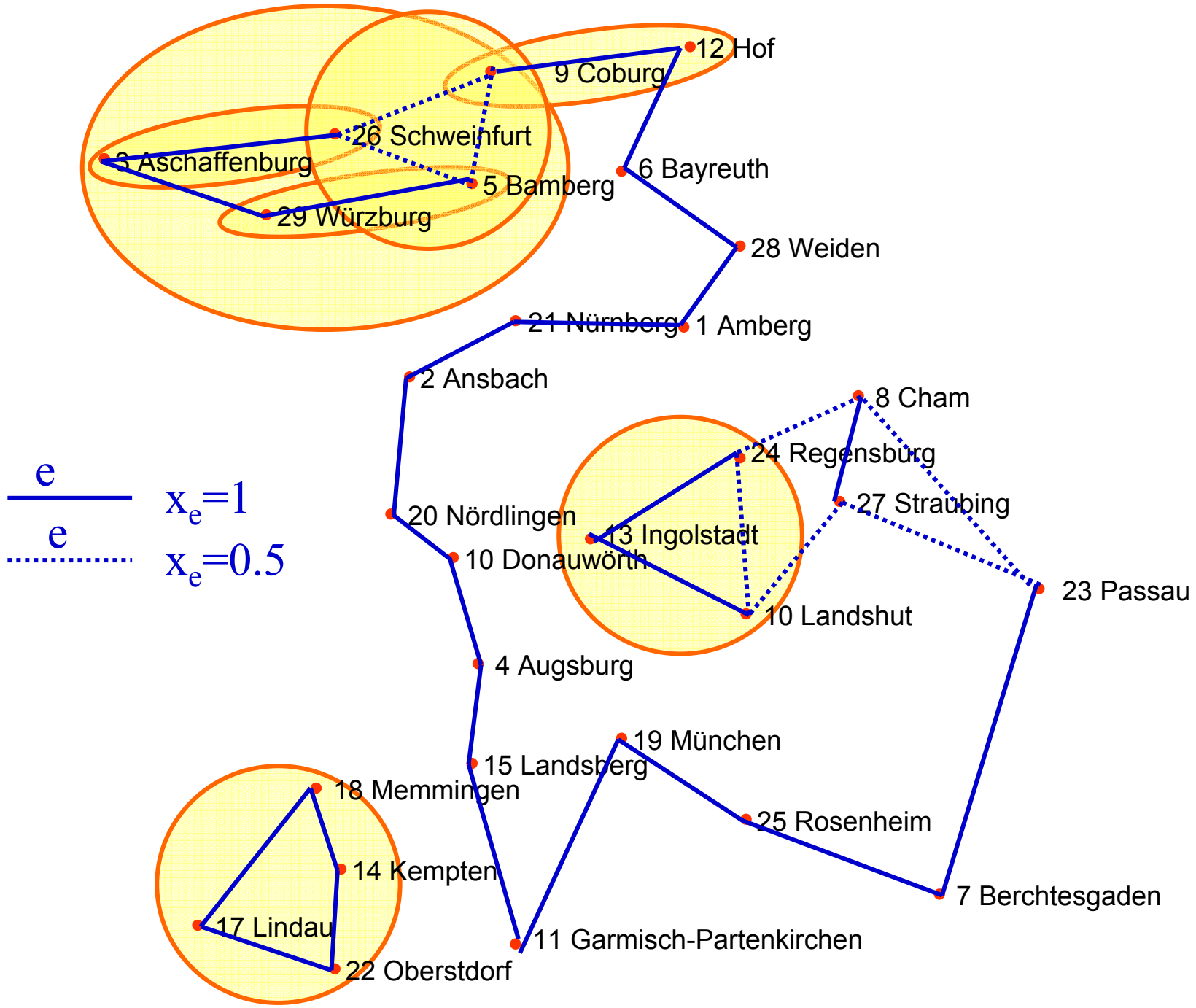
Beispiel: Bayern

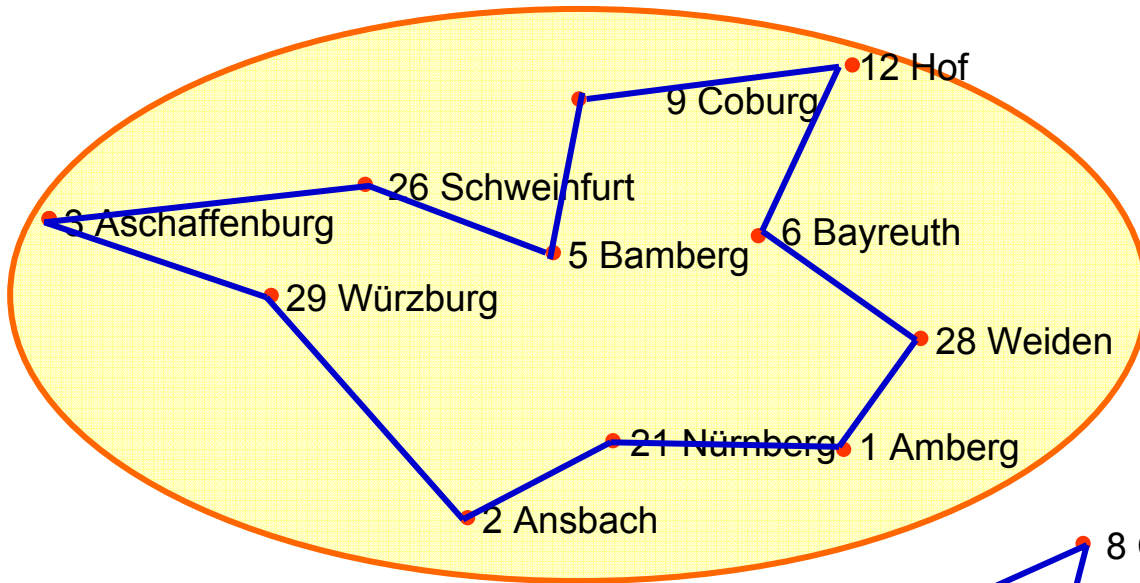
- 29 Städte in Bayern
- TSPLIB/bayg29.tsp



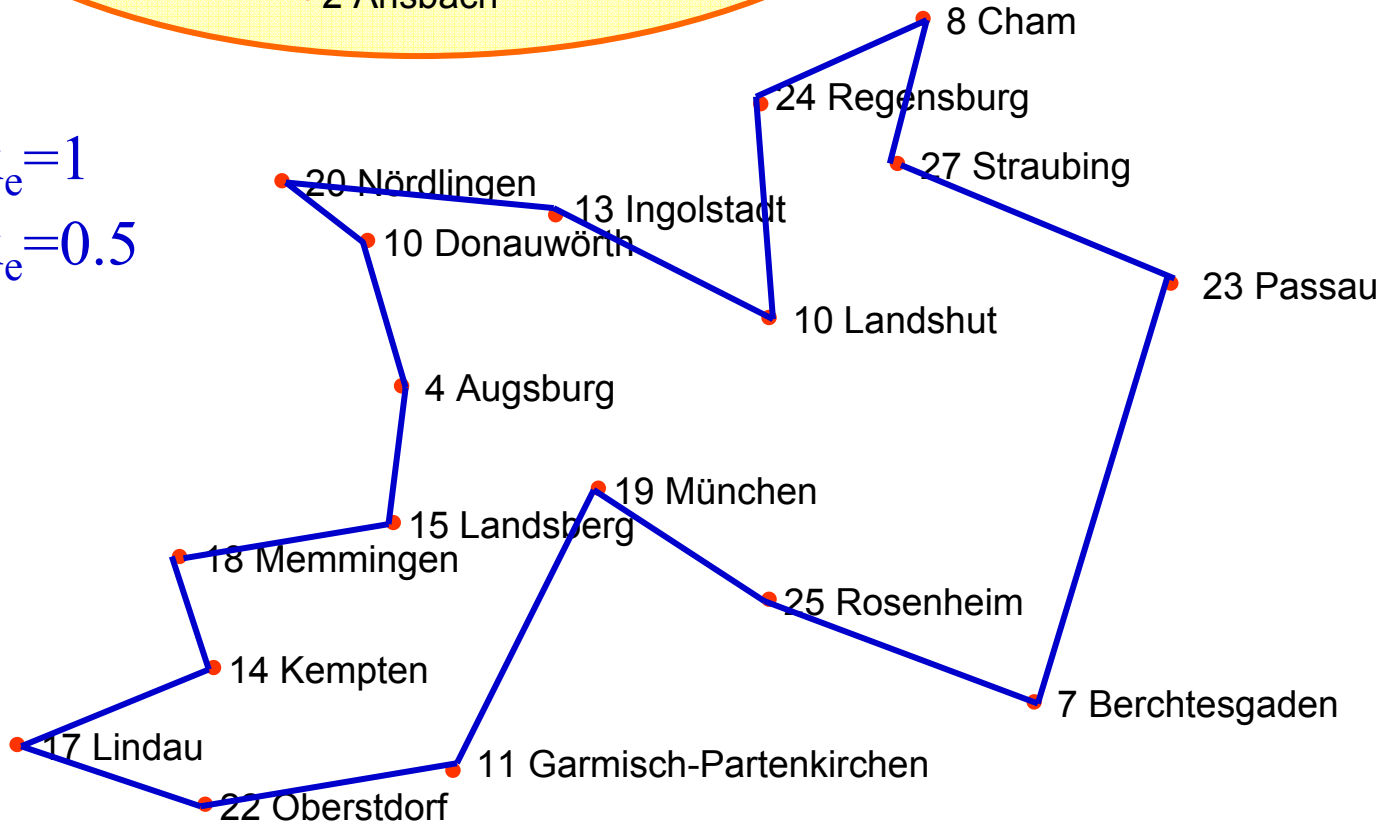


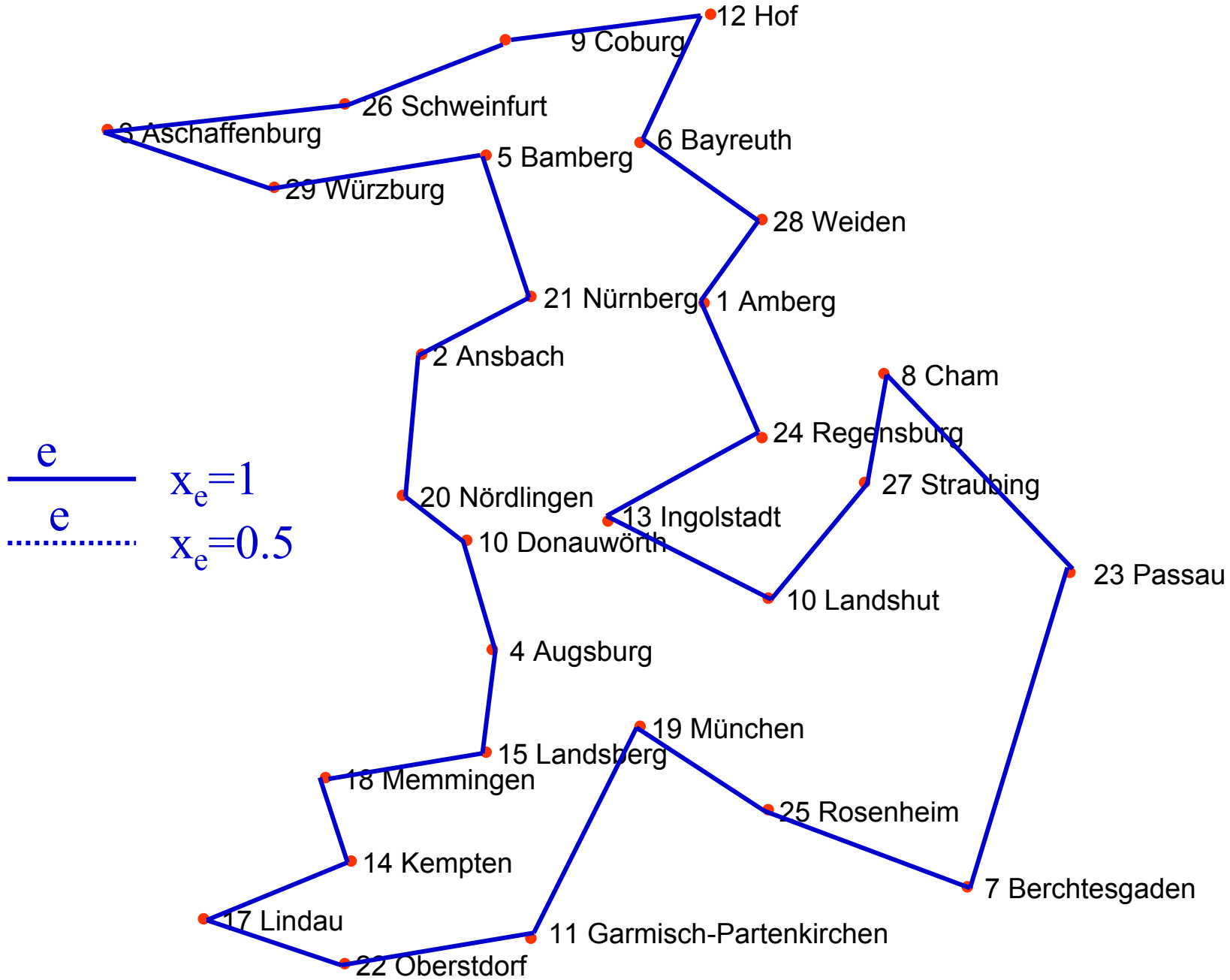






$\frac{e}{e} \quad x_e = 1$
 $\frac{e}{\dots} \quad x_e = 0.5$





Spaltengenerierung für TSP

Idee:

1. Starte nicht mit allen Variablen, sondern nur mit einer Teilmenge $E' \subseteq E$ (dies entspricht einer Teilmenge der Spalten des ILPs).
2. Bewährt hat sich Sparse Graph \subseteq Reserve Graph \subseteq ganz G.
3. z.B. als Sparse Graph den 5-Nächsten Nachbar Graphen, als Reserve Graph den 10-Nächsten Nachbarn
4. Löse die LP-Relaxierung LPRel bzgl. E'
5. Nun muss man testen, ob diese Lösung auch eine gültige Lösung der LP-Relaxierung bzgl. E ist.

Spaltengenerierung für TSP

Idee:

1. Starte nicht mit allen Variablen, sondern nur mit einer Teilmenge $E' \subseteq E$ (dies entspricht einer Teilmenge der Spalten des ILPs).
2. Löse die LP-Relaxierung LPRel bzgl. E'
3. Nun muss man testen, ob diese Lösung auch eine gültige Lösung der LP-Relaxierung bzgl. E ist.
4. Falls dies nicht der Fall ist, müssen zusätzliche Spalten hinzugefügt werden → Re-Optimierung über dem neuen $E' \rightarrow$ gehe zu 3.
5. Sonst: Falls die Lösung von LPRel ganzzahlig ist → Optimallösung gefunden
6. Sonst: Branching

„Branch&Cut&Price“

Pricing-Problem

Geg. ist eine Lösung x^* von LPRel bzgl. $E' \subseteq E$. Ist diese optimal auch bzgl. E ?

Eine neue Spalte $t \in E \setminus E'$ kann den Lösungswert nur dann verbessern, wenn die reduzierten Kosten der dazugehörigen Variablen kleiner als 0 sind.

Reduzierte Kosten für neue Variable t : $c_t - \pi A_t$, wobei

- A_t die Spalte t der Matrix bezeichnet
- c_t der dazugehörige Kostenkoeffizient, und
- π der Lösungswert der dualen Variablen zu Zeile i .

Suche also eine Spalte t , für die der Wert $c_t - \pi A_t < 0$.

Bei TSP: probiere zunächst alle Kanten aus Reserve Graph, wenn nicht erfolgreich, dann alle restlichen Kanten aus G .

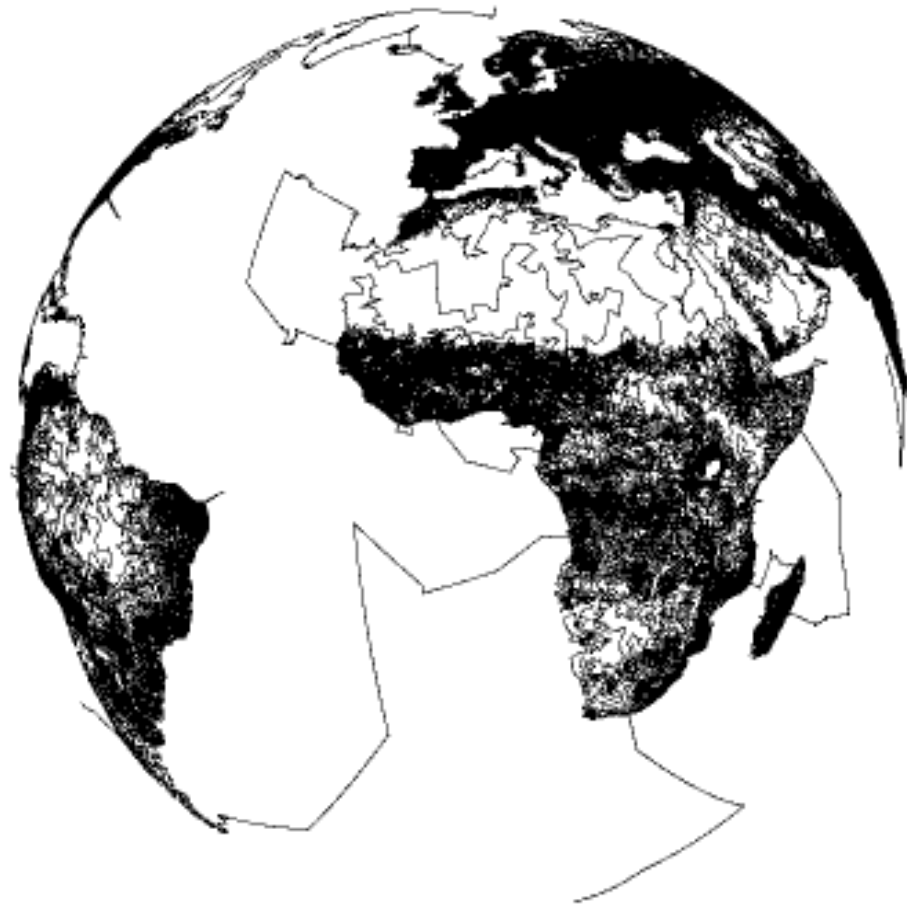
Pricing-Problem

Bemerkung: Für alle Spalten, die bereits in E' enthalten sind gilt: die reduzierten Kosten von Lösung x^* sind ≥ 0 , sie werden also nicht wieder ausgewählt.

und jetzt: auf zur neuen Challenge:

World Tour: 1.904.711 Städte

aktuelle Challenge



4.4 Engineering Lin-Kernighan

Johnson & McGeoch: „For over a decade and a half, from 1973 to about 1989, the world champion heuristic for the TSP was generally recognized to be the local search algorithm of Lin and Kernighan [1973].“

Applegate et al. 2007: At the heart of the most successful tour-finding approaches to date lies the simple and elegant algorithm of Lin and Kernighan. This is remarkable, given the wide range of attacks that have been made on the TSP in the past three decades...“

Es existieren viele Varianten davon.
Wir: TSP-Buch 2007

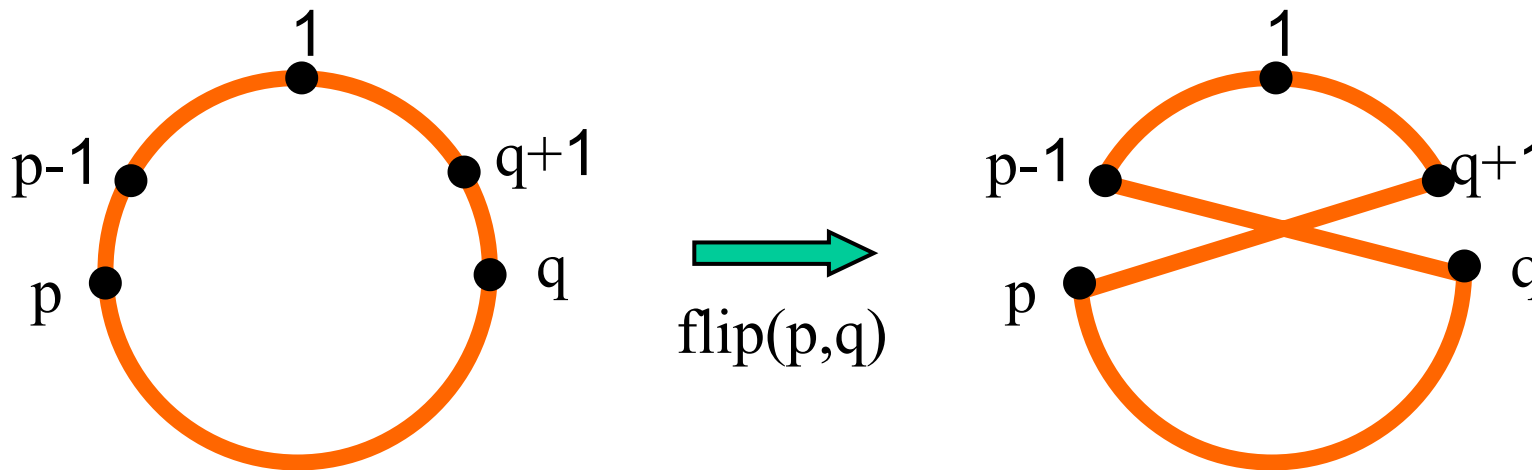
Engineering Lin-Kernighan

- Einführung
 - k -optimale Touren und *flips*
 - Grundoperationen von Lin-Kernighan
- Basisalgorithmus Lin-Kernighan
- Engineering Lin-Kernighan
 - Iterated Lin-Kernighan
 - Chained Lin-Kernighan
 - Lin-Kernighan-Helsgaun (LKH)
- Experimentelle Resultate

Einführung: k-optimale Touren

Def. Eine Tour T ist **k-optimal**, wenn das Entfernen einer k -elementigen Kantenmenge sowie das Einfügen einer neuen k -elementigen Kantenmenge nicht zu einer besseren Tour führen kann.

Operation: $\text{flip}(p,q)$ macht aus einer Tour: $(0, \dots, p-1, p, \dots, q, q+1, \dots, n-1)$ die neue Tour: $(0, \dots, p-1, q, q-1, \dots, p+1, p, q+1, \dots, n-1)$

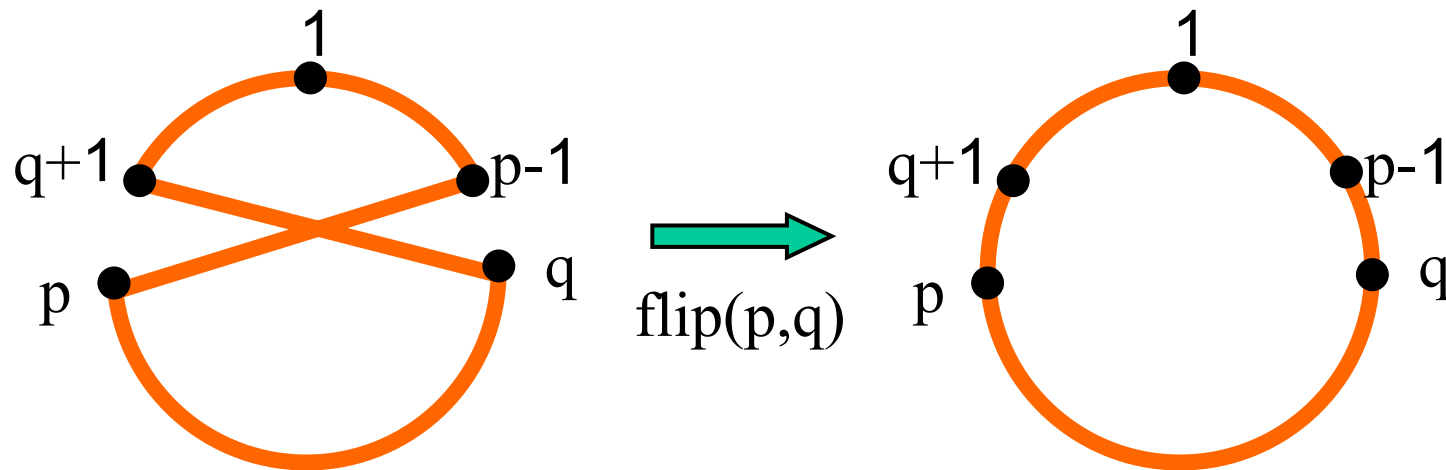


flip invertiert eine Teilsequenz einer Tour.

Einführung: k-optimale Touren

Def. Eine Tour T ist **k-optimal**, wenn das Entfernen einer k -elementigen Kantenmenge sowie das Einfügen einer neuen k -elementigen Kantenmenge nicht zu einer besseren Tour führen kann.

Operation: $\text{flip}(p,q)$ macht aus einer
Tour: $(0, \dots, p-1, p, \dots, q, q+1, \dots, n-1)$ die neue
Tour: $(0, \dots, p-1, q, q-1, \dots, p+1, p, q+1, \dots, n-1)$



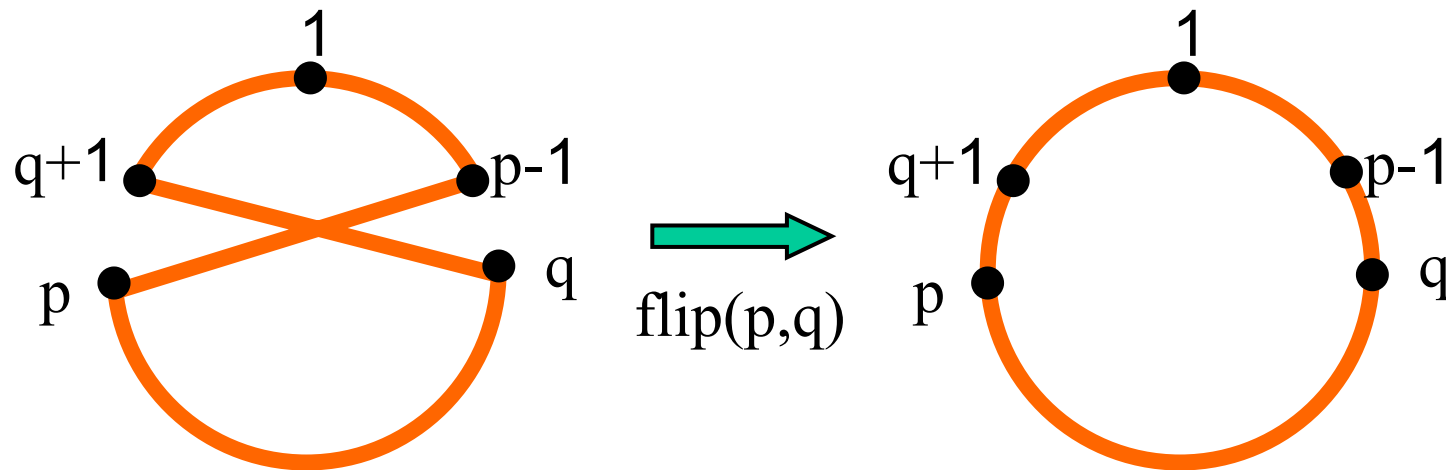
flip sinnvoll, wenn neue Tour besser ist $\Leftrightarrow c_{p-1p} + c_{qq+1} > c_{p-1q} + c_{pq+1}$

Einführung: 2-optimale Touren

Ein Paar (p,q) mit

$c_{p-1p} + c_{qq+1} > c_{p-1q} + c_{pq+1}$ heißt **Kreuzungspaar**.

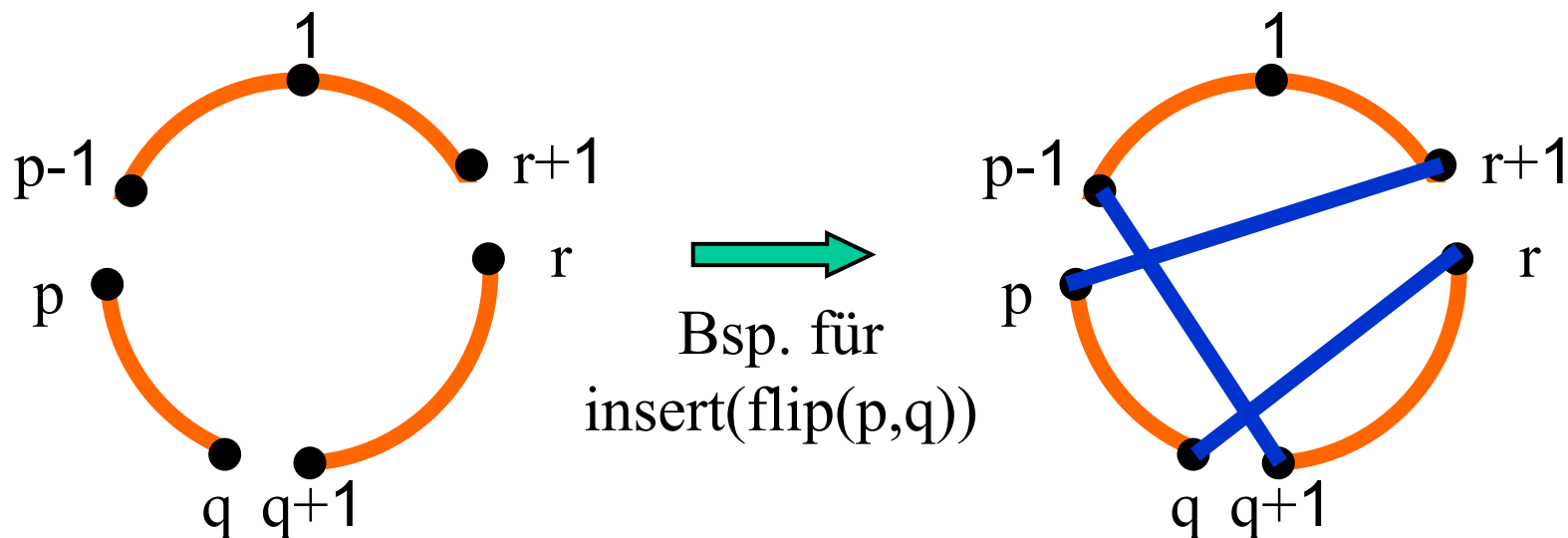
Eine Tour T ist 2-optimal $\Leftrightarrow T$ enthält kein Kreuzungspaar



Einführung: 3-optimale Touren

- $\text{flip}(p,q)$ invertiert die Teilsequenz $[p..q]$ einer Tour.
- $\text{insert}(p,q)$ fügt die Teilsequenz $[p..q]$ an anderer Stelle in die Tour ein
- $\text{insert}(\text{flip}(p,q))$ invertiert die Teilsequenz $[p..q]$ und fügt sie an anderer Stelle in die Tour wieder ein.

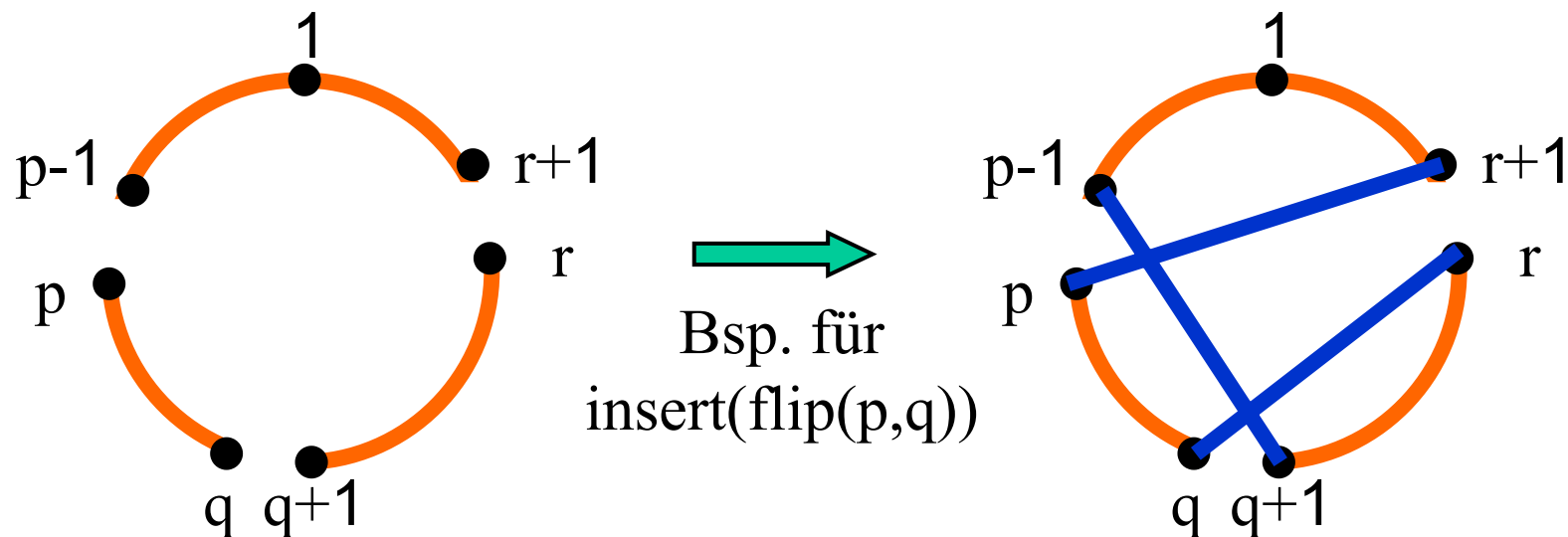
Eine Tour T ist 3-optimal $\Leftrightarrow T$ kann durch eine $\text{flip}()$, $\text{insert}()$ oder $\text{insert}(\text{flip}())$ -Operation nicht verbessert werden



Einführung: k-optimale Touren

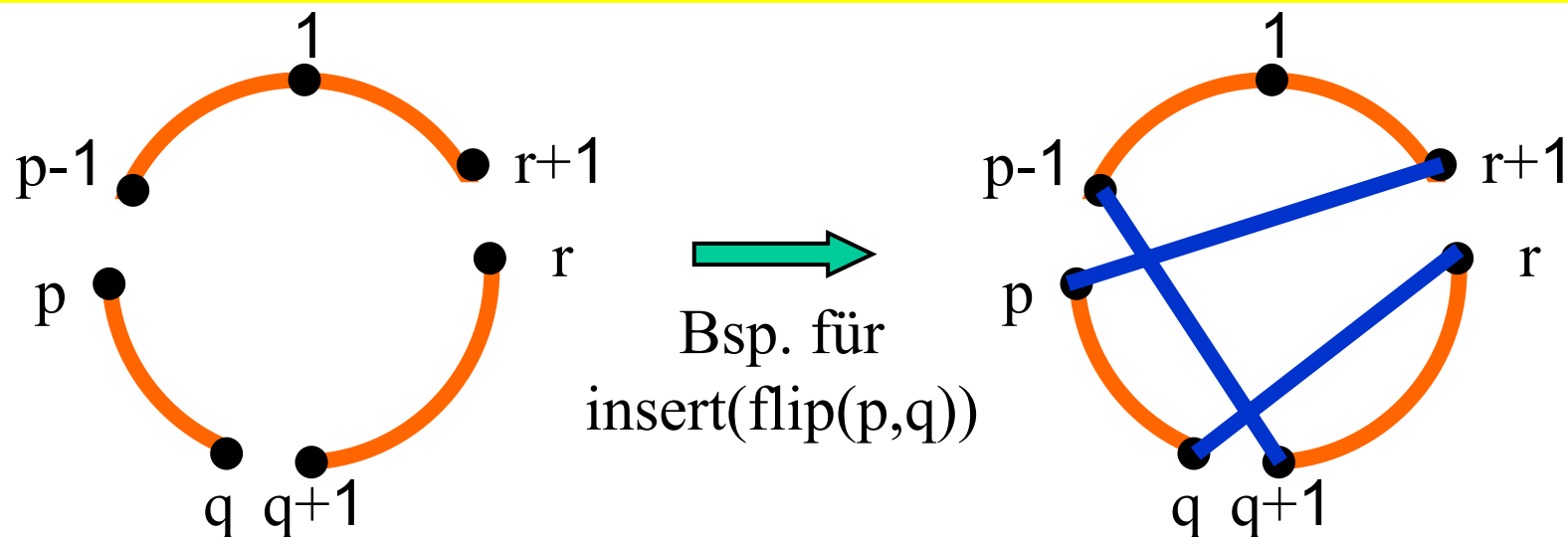
Jede Austauschoperation von k Kanten kann durch eine Sequenz von $\text{flip}()$ -Operationen simuliert werden.

Jede k -optimale Tour kann durch eine Sequenz von $\text{flip}()$ -Operationen erreicht werden.



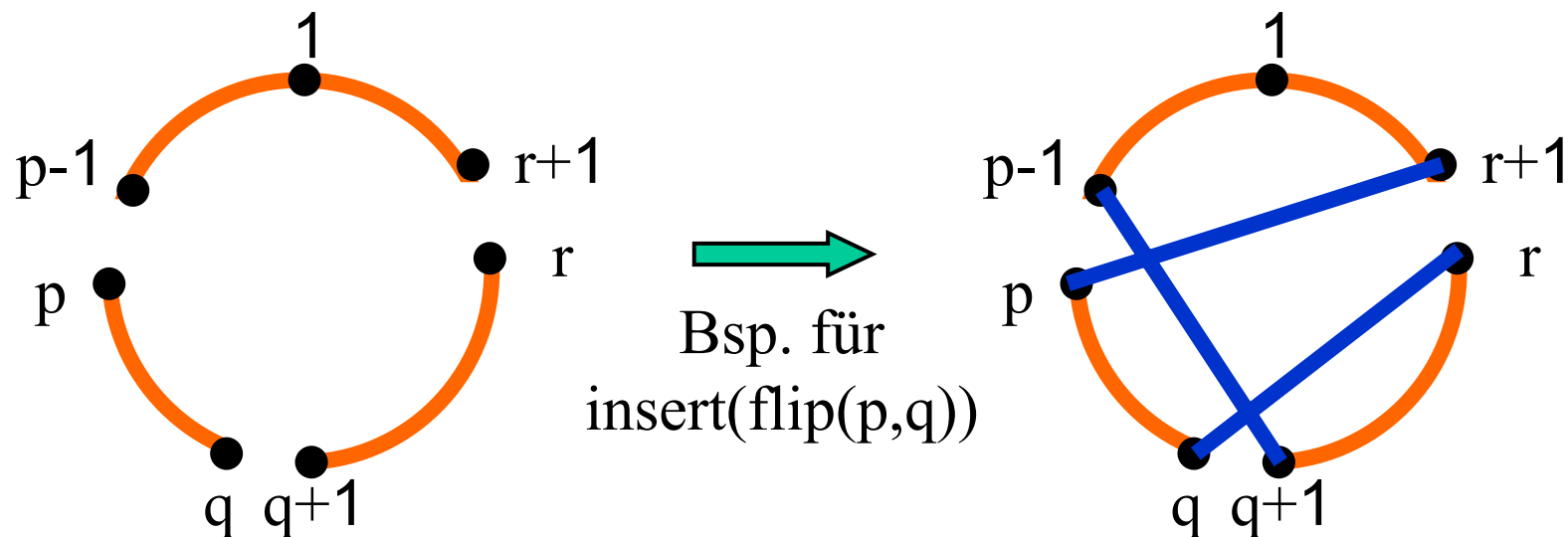
Einführung: Historie

- Croes [1958]: schlägt 2-opt Verfahren vor (local search)
- Lin [1965]: 3-opt Austausch vor ($\text{insert}(\text{flip}())$) → ergab deutlich bessere Touren
- Lin testete 4-opt, aber zu viel Zeitaufwand und kaum bessere Ergebnisse (bis 100 Knoten Graphen)
- Lin und Kernighan [1973]: „variable k-opt“: spezielle Sequenzen von $\text{flip}()$ und $\text{insert}(\text{flip}())$ Operationen.



Basisalgorithmus Lin-Kernighan

- LK erlauben Sequenzen von insgesamt drei verschiedenen Austauschoperationen:
 - 2 verschiedene flip()-Operationen
 - 2x2 verschiedene flip(insert())-Operationen

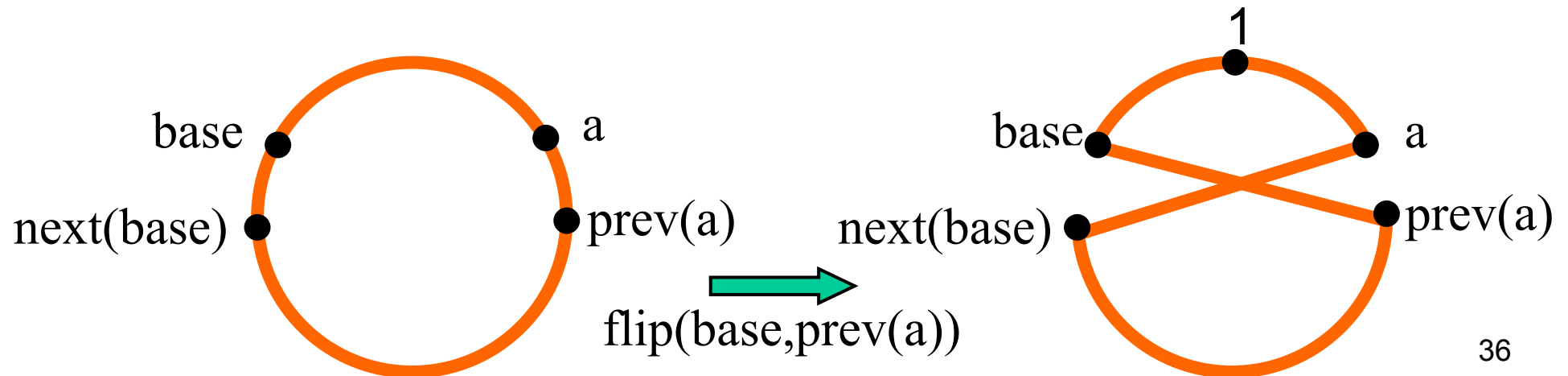


Basisalgorithmus Lin-Kernighan

- $c(i,j)$: Gesamtkosten von i nach j in Tour T
- $base$: ausgewählter Knoten
- $current_tour$: Tour nach flip-Sequenz auf T
- $next(v)$: Knoten direkt nach v in $current_tour$
- $prev(v)$: Knoten direkt vor v in $current_tour$

1. $flip()$ -Operation: $flip(next(base),prev(a))$

für alle $prev(a) \neq base, next(base), prev(base)$

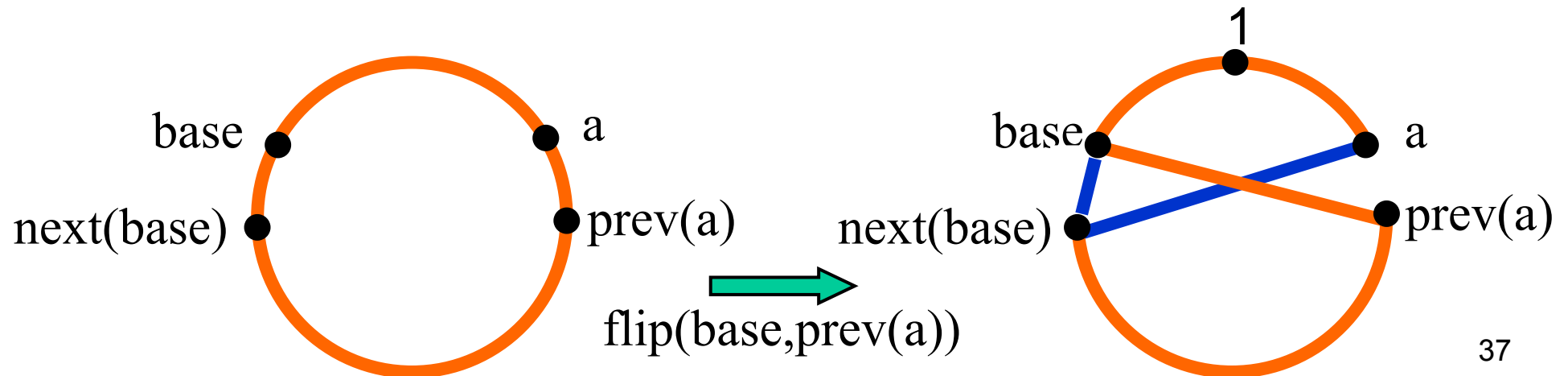


Basisalgorithmus Lin-Kernighan

- 2-opt würde den $\text{flip}(\text{next}(\text{base}), \text{prev}(a))$ nur durchführen, wenn
$$c(\text{base}, \text{next}(\text{base})) - c(\text{next}(\text{base}), a) + c(\text{prev}(a), a) - c(\text{base}, \text{prev}(a)) > 0$$
- LK führt ihn jedoch auch durch, wenn gilt:
$$c(\text{base}, \text{next}(\text{base})) - c(\text{next}(\text{base}), a) > 0$$

1. flip()-Operation: $\text{flip}(\text{next}(\text{base}), \text{prev}(a))$

für alle $\text{prev}(a) \neq \text{base}, \text{next}(\text{base}), \text{prev}(\text{base})$

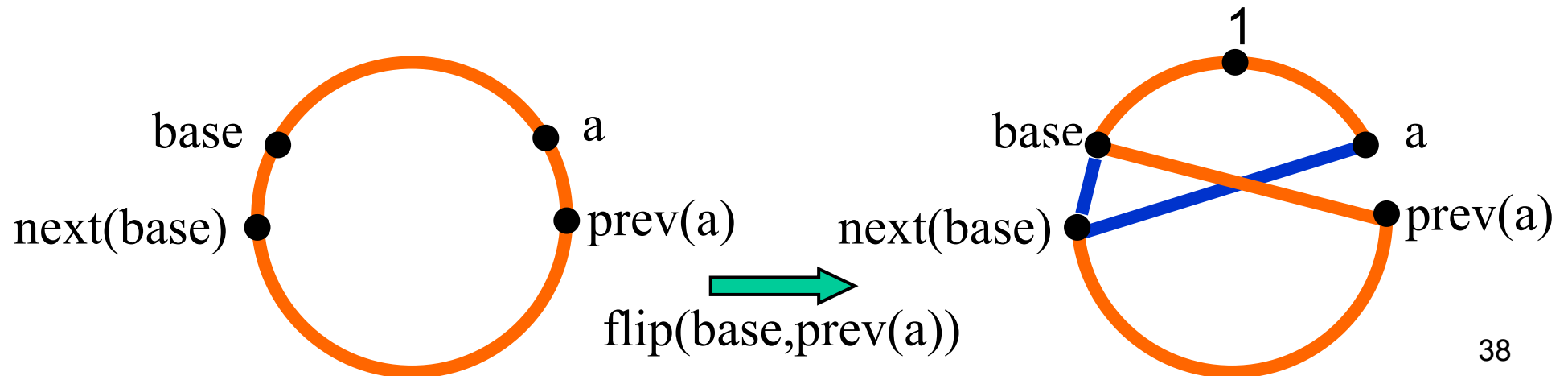


Basisalgorithmus Lin-Kernighan

- Sei $\delta := \text{Wert der Verbesserung einer flip() Sequenz}$
- Dann: $\text{cost}(\text{current_tour}) = \text{cost}(\text{starting_tour}) - \delta$
- LK führt einen neuen flip() nur dann aus, falls $\delta + c(\text{base}, \text{next}(\text{base})) - c(\text{next}(\text{base}), a) > 0$
- Wir nennen Knoten a **promising**, falls Bedingung (*) gilt

1. flip()-Operation: $\text{flip}(\text{next}(\text{base}), \text{prev}(a))$

für alle $\text{prev}(a) \neq \text{base}, \text{next}(\text{base}), \text{prev}(\text{base})$



Fragment 1 Lin-Kernighan

1. Setze $\text{delta}:=0$
2. **While** ein promising Nachbar von $\text{next}(\text{base})$ existiert do
3. Sei a promising Nachbar von $\text{next}(\text{base})$ der den Wert von $c(\text{prev}(a),a)-c(\text{next}(\text{base}),a)$ maximiert
4.
$$\text{delta} := \text{delta} + c(\text{base},\text{next}(\text{base})) - c(\text{next}(\text{base}),a) + c(\text{prev}(a),a) - c(\text{prev}(a),\text{base})$$
5. addiere $\text{flip}(\text{next}(\text{base}),\text{prev}(a))$ zur $\text{flip}()$ -Sequenz

Zweite flip() Operation

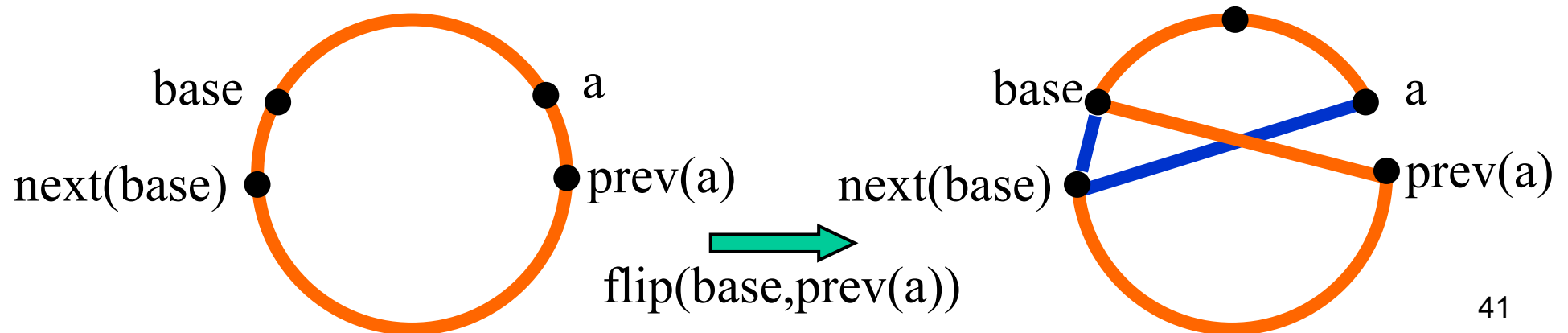
Mak & Morton [1993]

2. flip()-Operation: $\text{flip}(\text{prev}(a), \text{base})$
für alle $\text{prev}(a) \neq \text{base}, \text{next}(\text{base}), \text{prev}(\text{base})$

nächste VO

Hinzufügung von Backtracking

- Sei $\text{breadth}(k)$ die maximale Anzahl von promising Nachbarn, die auf level k getestet werden.
- Lin und Kernighan: $\text{breadth}(1)=5$, $\text{breadth}(2)=5$, $\text{breadth}(k)=1$ für alle $k \geq 2$
- $\text{breadth}(k)=0$ begrenzt die Länge der betrachteten flip()-Sequenzen



Fragment 2 LK: STEP(level,delta)

1. Erzeuge eine lk-Ordnung der Nachbarn von base
2. Setze $i:=1$
3. **While** ein neuer Knoten in lk-Ordnung existiert und $i \leq \text{breadth}(\text{level})$ do
4. Sei a der nächste Nachbar in lk-Ordnung
5. **Falls** a ein Mak-Morton move ist, dann
6. Bearbeite Mak-Morton move (nächstes Mal)
7. **Sonst** $g = c(\text{base}, \text{next}(\text{base})) - c(\text{next}(\text{base}), a)$
 $+ c(\text{prev}(a), a) - c(\text{prev}(a), \text{base})$
5. addiere $\text{flip}(\text{next}(\text{base}), \text{prev}(a))$ zur $\text{flip}()$ -Sequenz
6. Aufruf von $\text{step}(\text{level}+1, \text{delta}+g)$
7. **Falls** eine bessere Tour gefunden wurde \rightarrow return
8. **Sonst:** entferne die hinzugefügte $\text{flip}()$ Operation vom Ende der Sequenz und erhöhe i um +1

LK-SEARCH(v,T)

1. Initialisiere current_tour als T
2. Initialisiert die leere flip()-Sequenz
3. base = v
4. Aufruf von STEP(1,0)
5. **Falls** eine bessere current_tour gefunden wurde
6. **Dann** return die flip()-Sequenz
7. **Sonst** Aufruf von alternate_step() (nächstes Mal)
8. **Falls** eine verbesserte current_tour gefunden wurde
9. **Dann** return die flip() Sequenz
10. **Sonst** return with unsuccessful flag

LIN-KERNIGHAN(T)

1. Initialisiere lk-tour als T und markiere alle Knoten
2. **While** markierte Knoten existieren **do**
3. Wähle einen markierten Knoten v
4. Aufruf von LK_SEARCH(v ,lk_tour)
5. **Falls** eine verbesserte flip() Sequenz gefunden wurde
6. **Dann While** die flip() Sequenz nicht-leer ist **do**
7. sei flip(x,y) der nächste flip in der Sequenz
8. wende flip(x,y) auf lk_tour an \rightarrow neue lk_tour
9. markiere Knoten x und y
10. entferne flip(x,y) aus der flip() Sequenz
11. **Sonst** unmark v
12. return tour lk_tour

mehr: s. nächste VO