

Kap. 4: Das Handlungsreisendenproblem (TSP)

4.4 Engineering Lin-Kernighan

Professor Dr. Petra Mutzel

Lehrstuhl für Algorithm Engineering, LS11

9. VO

3. Mai 2007

Experimentelle Resultate für TSP mittels Branch & Cut

Jünger, Reinelt, Rinaldi: The Traveling Salesman
Problem, TR 1994, IASI CNR Rom

www.gatech.edu

Concorde, 2003

Problem	BC- Knoten	Subtour- Relaxierung	Anzahl Cuts	Zeit (s)	BC- NEU	Zeit NEU
pr107	1	100%	111	10	1	0,37
pr226	1	100%	296	87	1	1,06
lin318	5	99,7%	1124	344	1	2,78
rat575	111	99,3%	24185	7666	17	70,30
nrv1379	615	98,5%	226518	155221	7	108,12
pr2392	3	98,8%	11301	7056	1	35,04

Überblick

+Tabellen CUTS

- 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

Literatur

- TSP-Web Site: www.tsp.gatech.edu
- NEUES BUCH: D. Applegate, R.E. Bixby, V. Chvátal und W.J. Cook: **The Traveling Salesman Problem: A Computational Study**, [Princeton University Press](#) 2007, 606 pp.

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

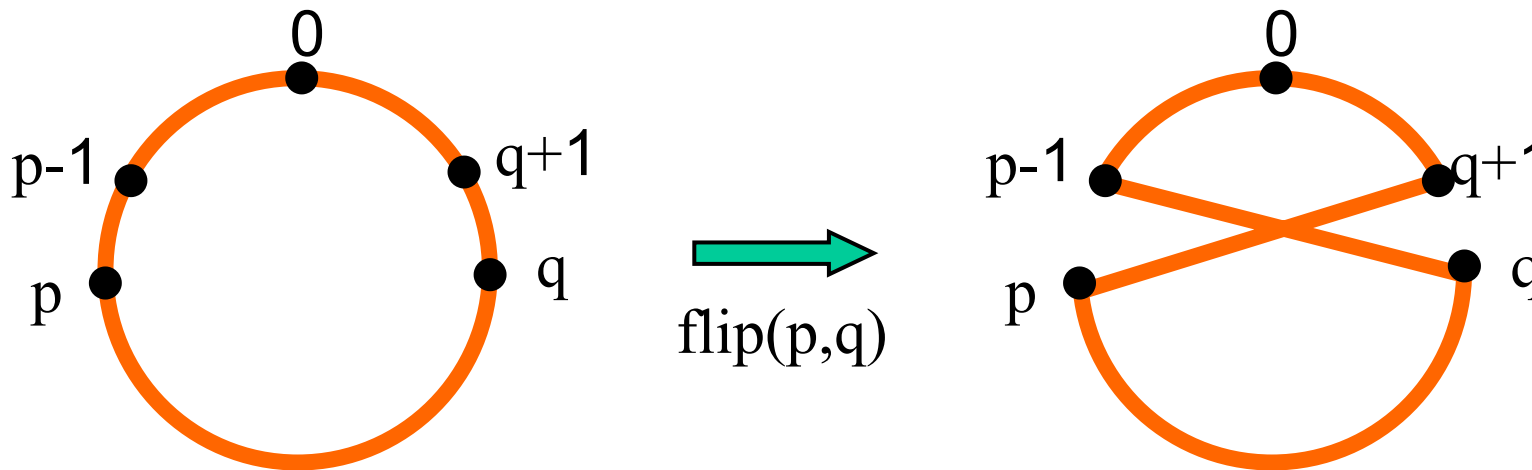
Engineering Lin-Kernighan

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

4.4.1 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: $(0, \dots, p-1, q, q-1, \dots, p+1, p, q+1, \dots, n-1)$

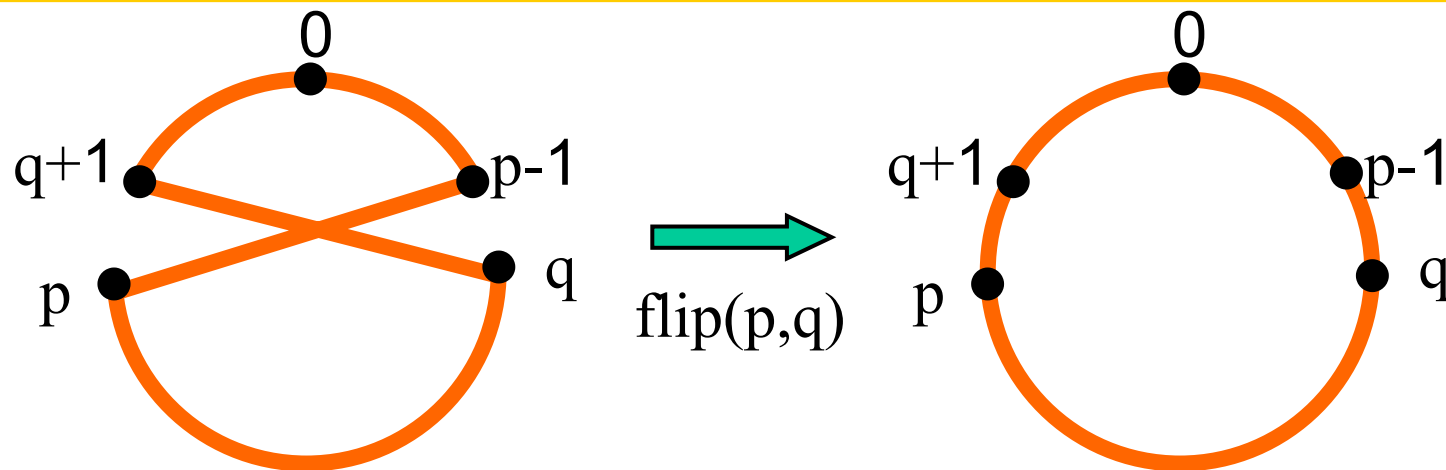


flip invertiert eine Teilsequenz einer Tour

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: $(0, \dots, p-1, p, \dots, \dots, q, q+1, \dots, n-1)$ die neue Tour: $(0, \dots, p-1, q, q-1, \dots, p+1, p, q+1, \dots, n-1)$



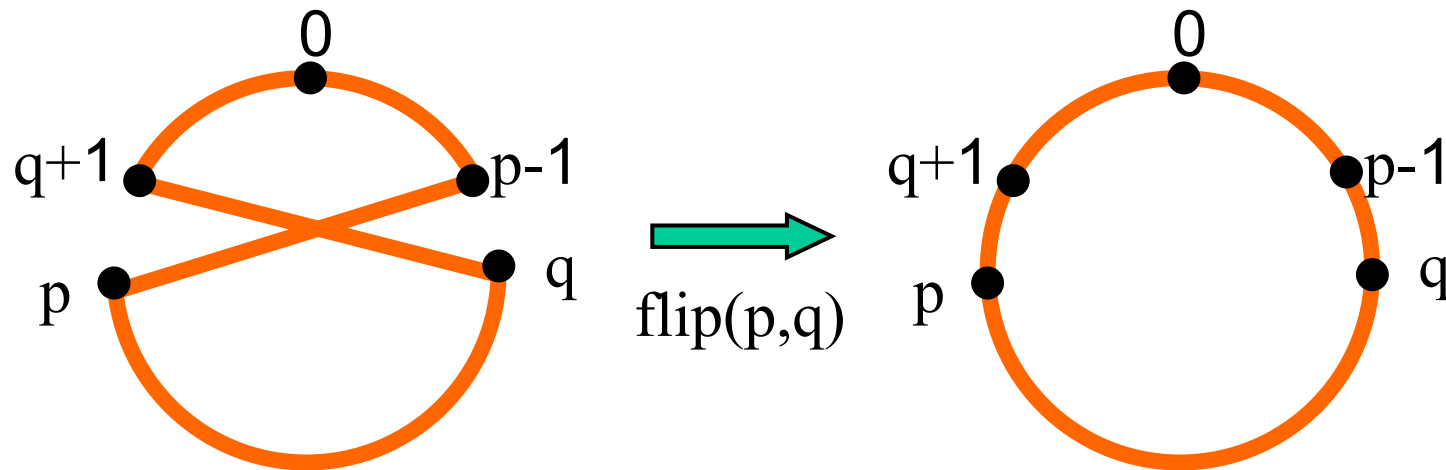
flip sinnvoll, wenn neue Tour besser ist $\Leftrightarrow c_{p-1p} + c_{qq+1} > c_{p-1q} + c_{pq+1}$

Einführung: 2-optimale Touren

Ein Paar (p,q) mit

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

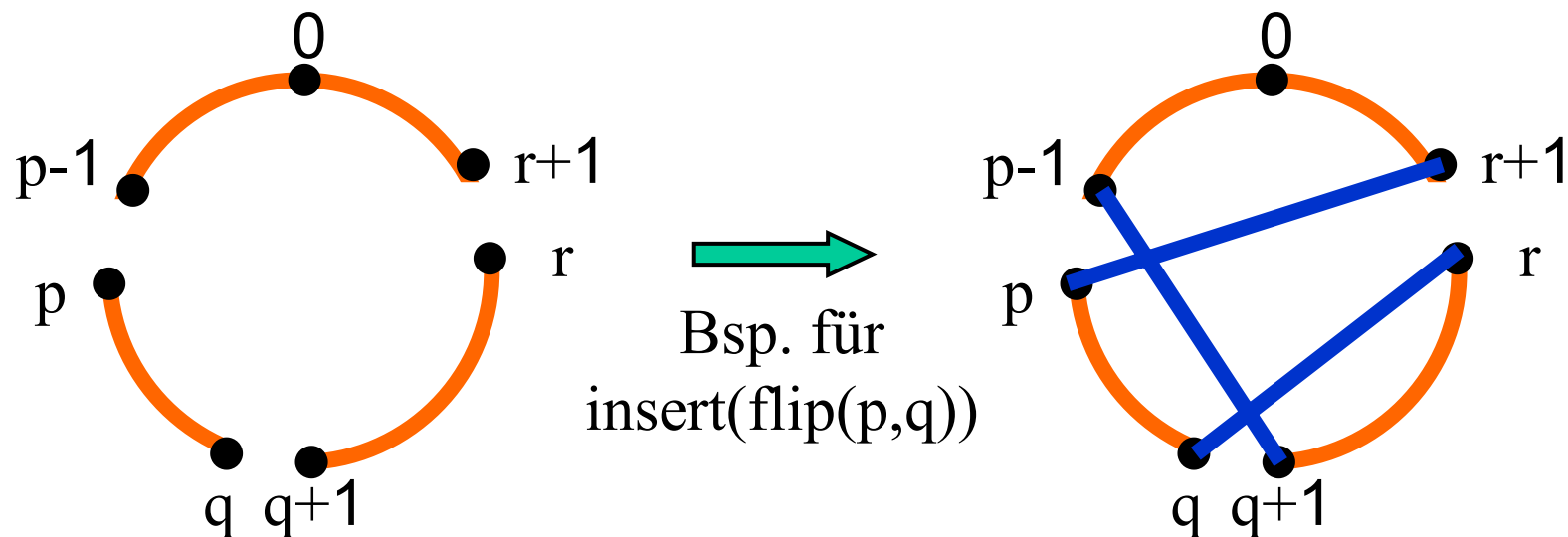
Eine Tour T ist 2-optimal $\Leftrightarrow T$ enthält kein Kreuzungspaar



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.

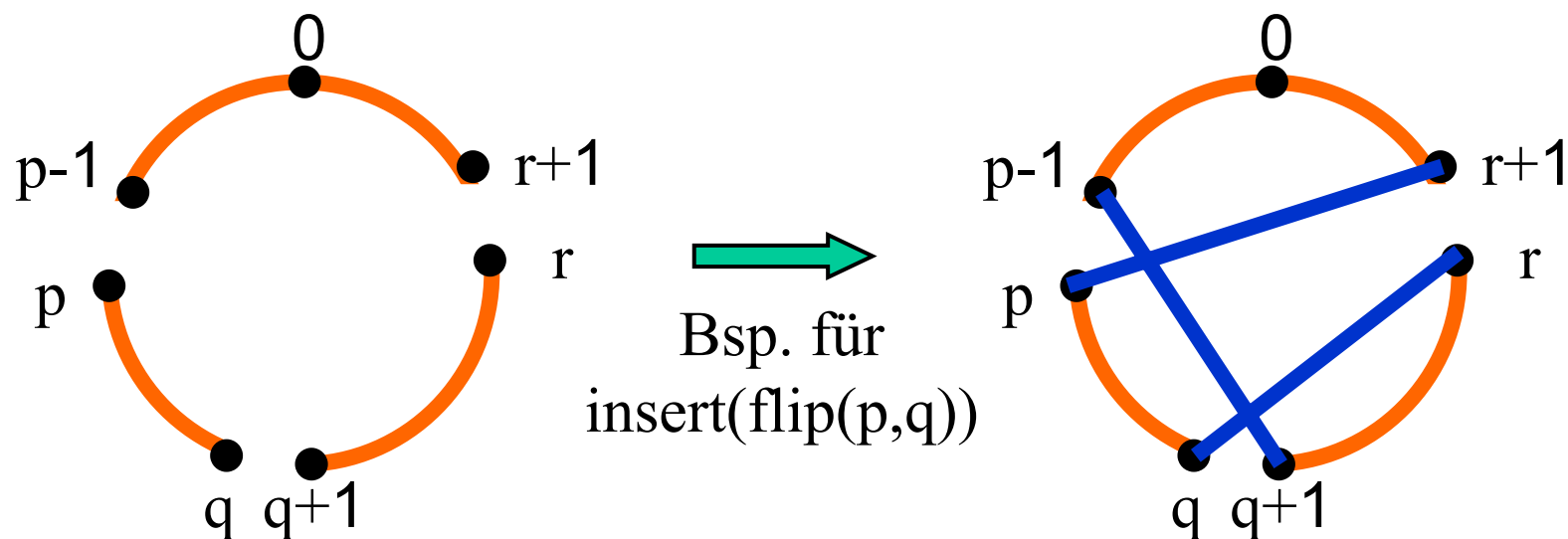
Eine Tour T ist 3-optimal $\Leftrightarrow T$ kann durch eine $\text{flip}()$, $\text{insert}()$ oder $\text{insert}(\text{flip}())$ -Operation nicht verbessert werden



Einführung: k-optimale Touren

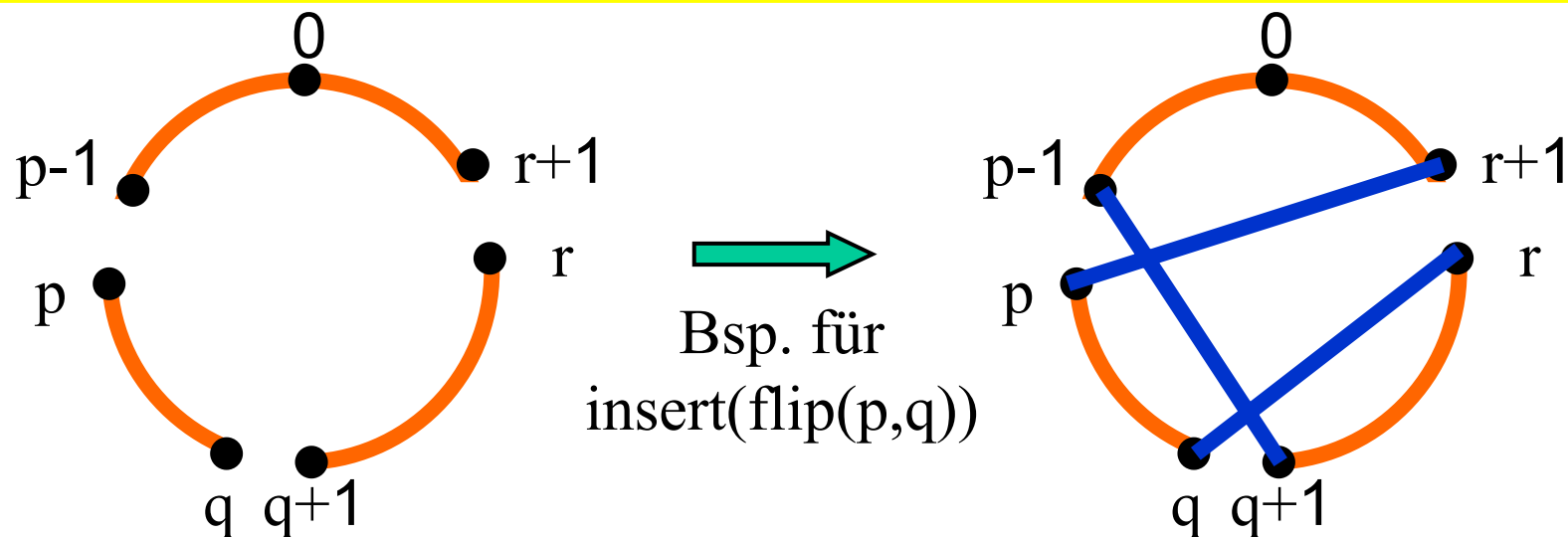
Jede Austauschoperation von k Kanten kann durch eine Sequenz von $\text{flip}()$ -Operationen simuliert werden.

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



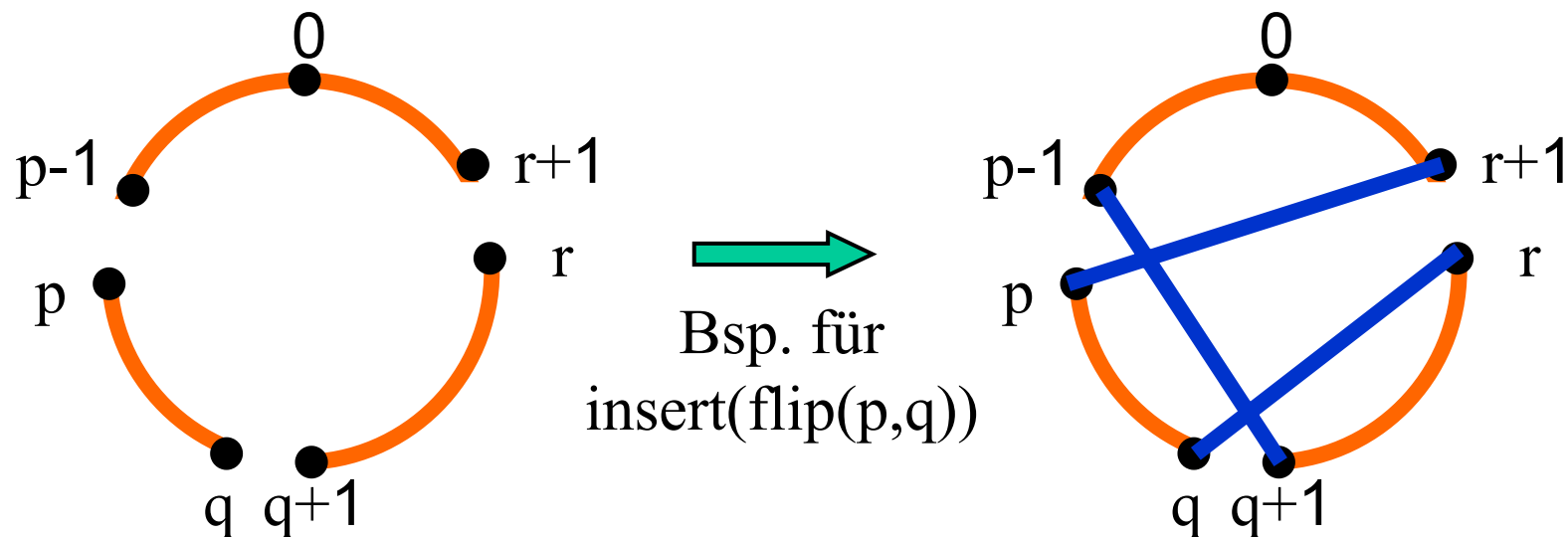
Einführung: Historie

- Croes [1958]: schlägt 2-opt Verfahren vor (local search)
- Lin [1965]: 3-opt Verfahren → 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 flip() und insert(flip())** Operationen.



4.4.2 Basisalgorithmus Lin-Kernighan

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

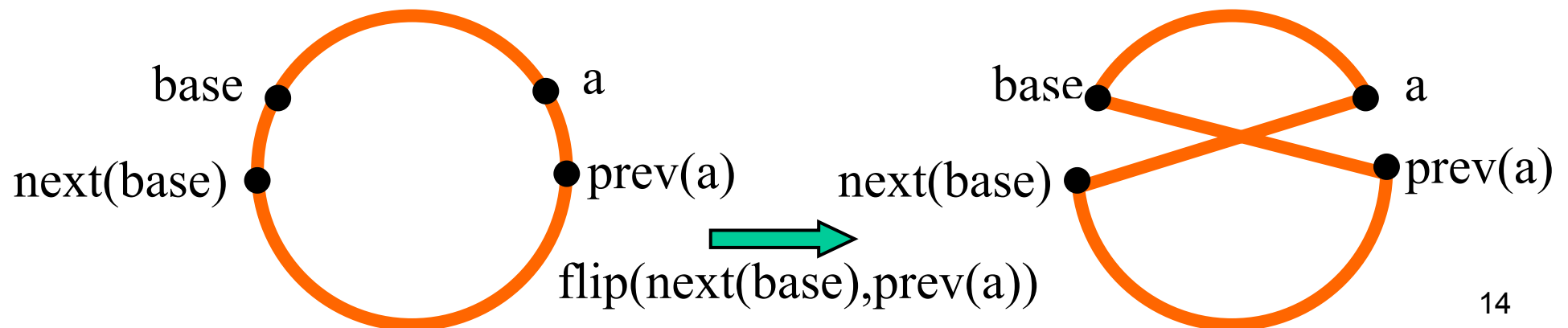


1. flip()-Operation: $\text{flip}(\text{next}(\text{base}), \text{prev}(a))$

- base: ausgewählter Knoten (Basis der flip()-Operationen)
- 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
- $c(i,j)$: Gesamtkosten von i nach j in Tour T

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

Welches a ist vielversprechend für festes base?



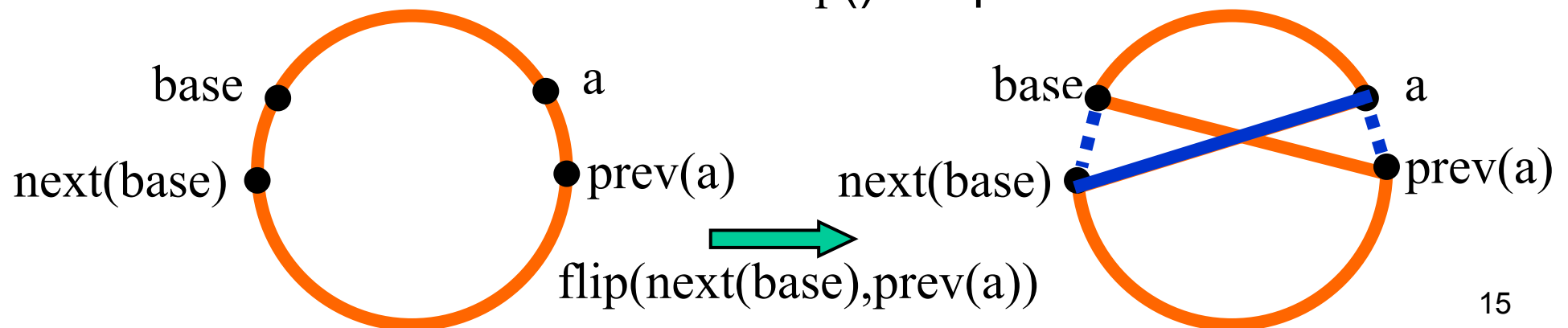
1. flip()-Operation: $\text{flip}(\text{next}(\text{base}), \text{prev}(a))$

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




Welches a ist vielversprechend für festes base ?

- ein a mit möglichst kleinem Wert $c(\text{next}(\text{base}), a)$
- und mit möglichst großem Wert $c(\text{prev}(a), a)$

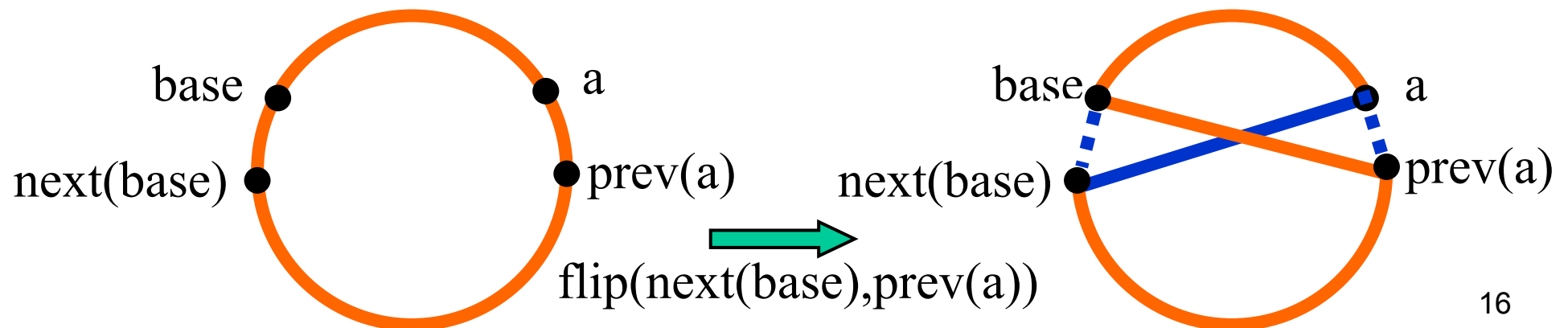
verbinde dies mit flip()-Sequenzen



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

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

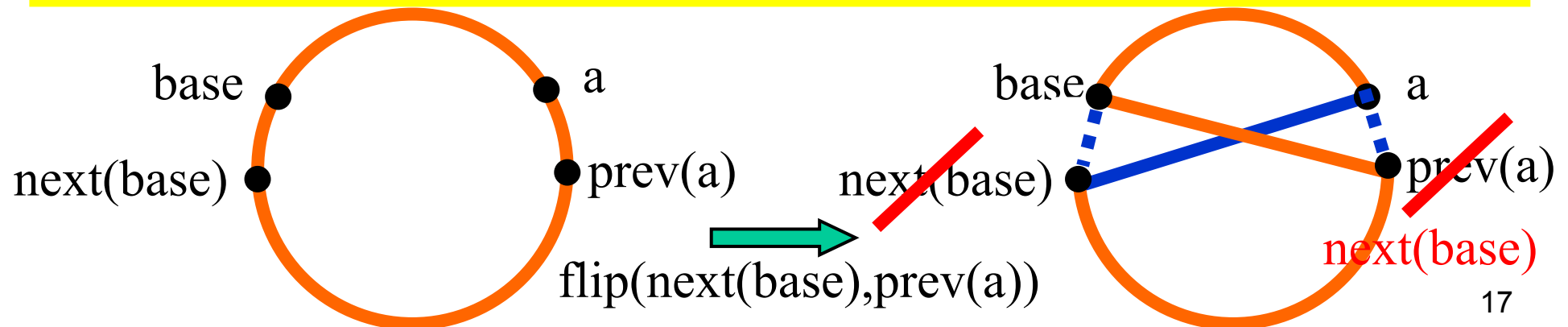
- ein a mit möglichst kleinem Wert $c(\text{next}(\text{base}), a)$
- und mit möglichst großem Wert $c(\text{prev}(a), a)$



Fragment 1 Lin-Kernighan

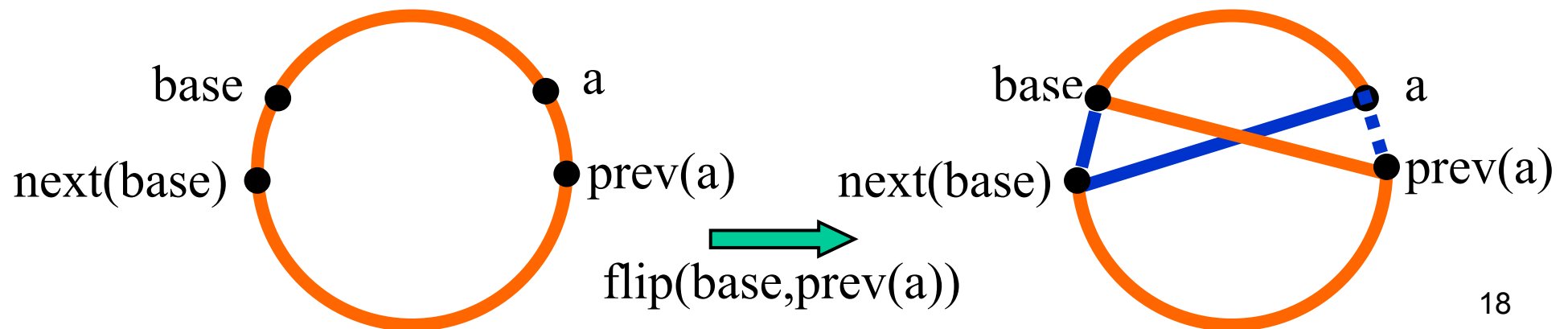
- Sortiere die promising Nachbarn von $\text{next}(\text{base})$ absteigend nach $c(\text{prev}(a),a) - c(\text{next}(\text{base}),a)$ → 1k-Ordnung

1. Setze $\text{delta} := 0$
2. **While** ein unbehandelter Nachbar in 1k-Ordnung von $\text{next}(\text{base})$ existiert **do**
3. Sei a der nächste Knoten in der Liste 1k-Ordnung
4. $\text{delta} := \text{delta} + c(\text{base},\text{next}(\text{base})) - c(\text{next}(\text{base}),a)$
5. $+ c(\text{prev}(a),a) - c(\text{prev}(a),\text{base})$
6. addiere $\text{flip}(\text{next}(\text{base}),\text{prev}(a))$ zur $\text{flip}()$ -Sequenz




Hinzufügung von Backtracking

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



STEP(level,delta) (vorläufig)

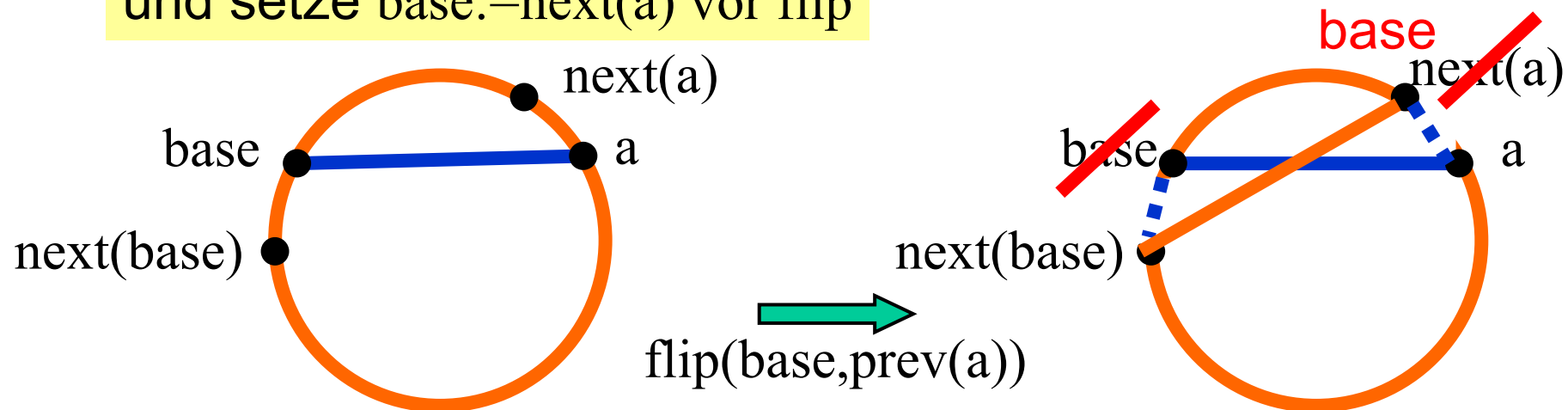
1. Erzeuge eine 1k-Ordnung der Nachbarn von next(base) 
2. Setze $i:=1$
3. **While** ein neuer promising Knoten in 1k-Ordnung existiert und $i \leq \text{breadth}(\text{level})$ do
4. Sei a der nächste promising Nachbar in 1k-Ordnung
5.
$$g = c(\text{base}, \text{next}(\text{base})) - c(\text{next}(\text{base}),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

2. flip() Operation: flip(next(a),base)



Mak & Morton [1993] für alle $a \neq \text{base}, \text{next}(\text{base}), \text{prev}(\text{base})$

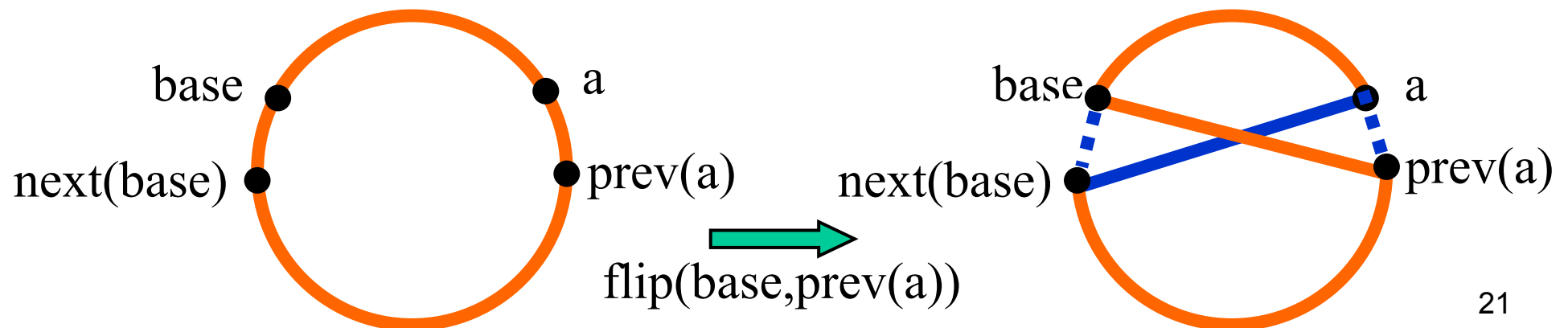
- führe einen Mak & Morton flip() nur dann aus, falls $\text{delta} + c(\text{base}, \text{next}(\text{base})) - c(\text{base}, a) > 0$
- sortiere Nachbarn von base absteigend nach $c(a, \text{next}(a)) - c(\text{base}, a)$ ☾
- nach dem flip() erhöhe delta um $c(\text{base}, \text{next}(\text{base})) - c(\text{base}, a) + c(a, \text{next}(a)) - c(\text{next}(a), \text{next}(\text{base}))$

und setze $\text{base} := \text{next}(a)$ vor flip



Verbindung der beiden flip()-Operationen

- Sortiere sowohl die Nachbarn von `base` als auch die Nachbarn von `next(base)` nach  bzw.  und füge sie in die gleiche Liste ein: lk-ordering
- Manche Knoten können doppelt in dieser Liste auftauchen
- In jedem Schritt des Algorithmus werden die Knoten in dieser Ordnung betrachtet
- Aufruf: `STEP(1,0)`



STEP(level,delta)

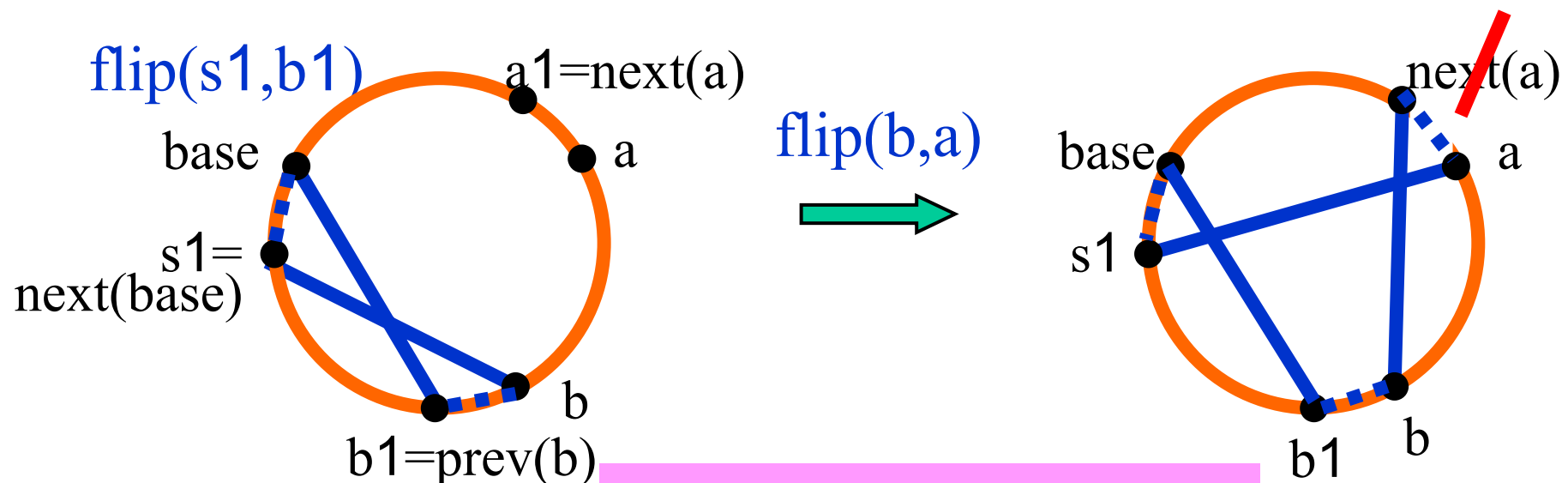
1. Erzeuge eine lk-Ordnung der Nachbarn von base und next(base) und setze $i:=1$
2. **While** ein neuer Knoten in lk-Ordnung existiert und $i \leq \text{breadth}(\text{level})$ do
3. Sei a der nächste Nachbar in lk-Ordnung
4. **Falls** a ein Mak & Morton move ist, dann
5. Bearbeite Mak & Morton move: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

STEP(level,delta) ff

4. **Falls** a ein Mak-Morton move ist, dann
5. $g = c(\text{base}, \text{next}(\text{base})) - c(\text{base}, a)$
6. $+ c(a, \text{next}(a)) - c(\text{next}(a), \text{next}(\text{base}))$
7. $\text{newbase} = \text{next}(a); \text{oldbase} = \text{base}$
8. addiere $\text{flip}(\text{newbase}, \text{base})$ zur $\text{flip}()$ -Sequenz
9. $\text{base} = \text{newbase}$
10. Aufruf von $\text{step}(\text{level}+1, \text{delta}+g)$
11. $\text{base} = \text{oldbase}$
12. **Sonst:** siehe Zeile 12 vorige Folie
13. **Falls** eine bessere Tour gefunden wurde \rightarrow return
14. **Sonst:** entferne die hinzugefügte $\text{flip}()$ Operation vom Ende der Sequenz und erhöhe i um +1

Dritte flip()-Sequenz: 3er Austausch

- Wähle einen Nachbarn b von $\text{next}(a)$ entfernt $(a, \text{next}(a))$
- **1. Fall:** b liegt zwischen $\text{next}(\text{base})$ und a :
 - Betrachte 2 Alternativen für $\text{flip}()$ -Sequenzen:
 - **1.1:** $\langle \text{flip}(a1, \text{base}), \text{flip}(\text{base}, b), \text{flip}(b, s1) \rangle \rightarrow$ entfernt zusätzlich $(b, \text{next}(b))$
 - **1.2:** $\langle \text{flip}(s1, b1), \text{flip}(b, a) \rangle$, wobei $b1 = \text{prev}(b)$ zu Beginn \rightarrow entfernt zusätzlich $(\text{prev}(b), b)$

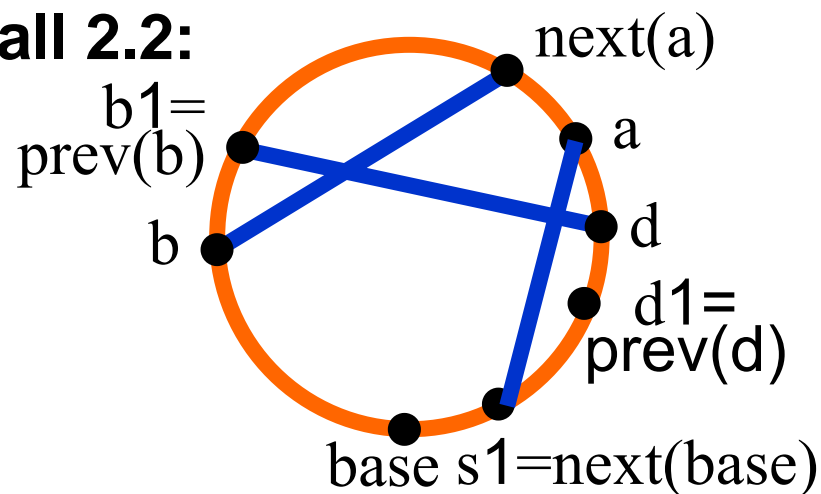


NEU (Achtung Fehler im Buch)

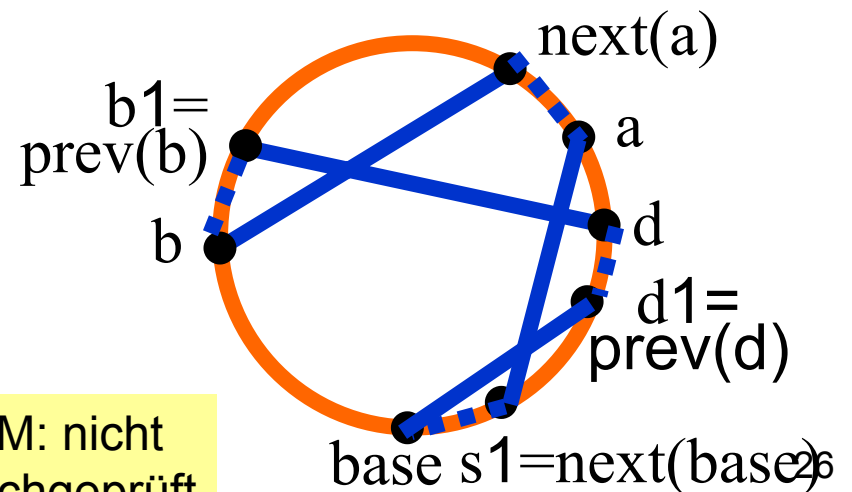
Dritte flip()-Sequenz: 4er Austausch

- **2. Fall:** b liegt nicht zwischen $\text{next}(\text{base})$ und a :
 - Betrachte Nachbarn d von $\text{prev}(b)$ mit d liegt zwischen $\text{next}(\text{base})$ und a entfernt $(a, \text{next}(a))$
 - Betrachte 2 Alternativen für flip()-Sequenzen:
 - **2.1:** $\langle \text{flip}(s1, b1), \text{flip}(b1, d1), \text{flip}(a1, s1) \rangle$, wobei $b1 = \text{prev}(b)$
 → entfernt zusätzlich $(d, \text{next}(d))$
 - **2.2:** $\langle \text{flip}(s1, d1), \text{flip}(d, a), \text{flip}(a1, b1) \rangle$, wobei $d1 = \text{prev}(d)$ zu Beginn
 → entfernt zusätzlich $(\text{prev}(d), d)$

Fall 2.2:



PM: nicht nachgeprüft



Dritte flip()-Sequenz: Backtracking

- Sei breadth_A die max. Anzahl an Knoten a , die ausprobiert werden
- A-ordering def. durch: sortiere die promising Nachbarn von $\text{next}(\text{base})$ absteigend nach $c(\text{next}(a),a)-c(\text{base},a)$
- Sei breadth_B die größte Anzahl an Paaren $(b,b1)$, die ausprobiert werden (wobei $b1$ entweder $\text{prev}(b)$ oder $\text{next}(b)$), ebenso breadth_D
- Für Wahl von $(b,b1)$ betrachten wir Nachbarn von $\text{next}(a)$, die ungleich base , $\text{next}(\text{base})$ und a sind, und erfüllen:
- $c(\text{next}(a),b) < c(a,\text{next}(a))+c(\text{base},\text{next}(\text{base}))-c(\text{base},a)$
- B-ordering def. durch: sortiere die Paare absteigend nach $c(b1,b)-c(\text{next}(a),b)$
- Für Wahl von $(d,d1)$ betrachten wir Nachbarn von $b1$, die ungleich base , $\text{next}(\text{base}),a,\text{next}(a),b$ sind, und erfüllen:
- $c(b1,d) < c(b,b1)+c(\text{base},\text{next}(\text{base}))-c(\text{next}(\text{base}),a)+c(a,\text{next}(a))-c(\text{next}(a),b)$
- D-ordering def. durch: sortiere die Paare absteigend nach $c(d1,d)-c(b1,d)$

alternate_step

1. $s1 := \text{next}(\text{base})$
2. Erzeuge die A-Ordnung der Nachbarn von $\text{next}(\text{base})$ und setze $i := 1$
3. **While** ein neuer Knoten in A-Ordnung existiert und $i \leq \text{breadth}_A$ **do**
4. Sei a der nächste Nachbar in A-Ordnung und setze $a1 := \text{next}(a)$
5. Erzeuge die B-Ordnung der Nachbarn von $\text{next}(a)$ und setze $j := 1$
6. **While** ein neuer Knoten in B-Ordnung existiert und $j \leq \text{breadth}_B$ **do**
7. Sei $(b, b1)$ das nächste Paar in der B-Ordnung
8. **Falls** b auf dem Segment von $\text{next}(b)$ nach a liegt
9. addiere die $\text{flip}()$ -Sequenz zu Fall 1
10. aktualisiere delta
11. Aufruf von $\text{STEP}(3, \text{delta})$
12. **Falls** eine bessere Tour gefunden wurde \rightarrow return
13. **Sonst:** entferne die hinzugefügten $\text{flip}()$ Operationen
14. **Sonst**

alternate_step ff

14. **Sonst** erzeuge die D-Ordnung der Nachbarn von b_1 , setze $k:=1$
15. **While** neue Paare in D-Ordnung existieren und $k \leq \text{breadth}_D$ **do**
16. sei (d, d_1) das nächste Paar in der D-Ordnung
17. füge die $\text{flip}()$ -Sequenz (**) hinzu und aktualisiere delta
18. Aufruf von $\text{STEP}(4, \text{delta})$
19. **Falls** eine bessere Tour gefunden wurde \rightarrow return
20. **Sonst:** entferne die hinzugefügten $\text{flip}()$ Operationen und
21. erhöhe k
22. erhöhe j
23. erhöhe i .

Aufruf von $\text{STEP}(4, \text{delta})$, da alternate_step nur einmal auf Rekursionsstufe 1 gestartet wird; 4 entspricht also tatsächlich der korrekten Rekursionsstufe nach dem Aufruf von $\text{STEP}()$

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()
8. **Falls** eine verbesserte current_tour gefunden wurde
9. **Dann** return die flip()-Sequenz
10. **Sonst** return mit flag „erfolglos“

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. **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 lk_tour

4.4.3 Engineering LIN-KERNIGHAN

- Bereits der Basisalgorithmus LK liefert Touren guter Qualität für viele Probleminstanzen
- Noch bessere Touren können durch wiederholte Anwendung von LK erreicht werden.

- Idee für **Iterated LK** (bereits von Lin und Kernighan 1973 vorgeschlagen)
- Solange die Zeit erlaubt:
 - Konstruiere neue Basistour T
 - Wende LK darauf an

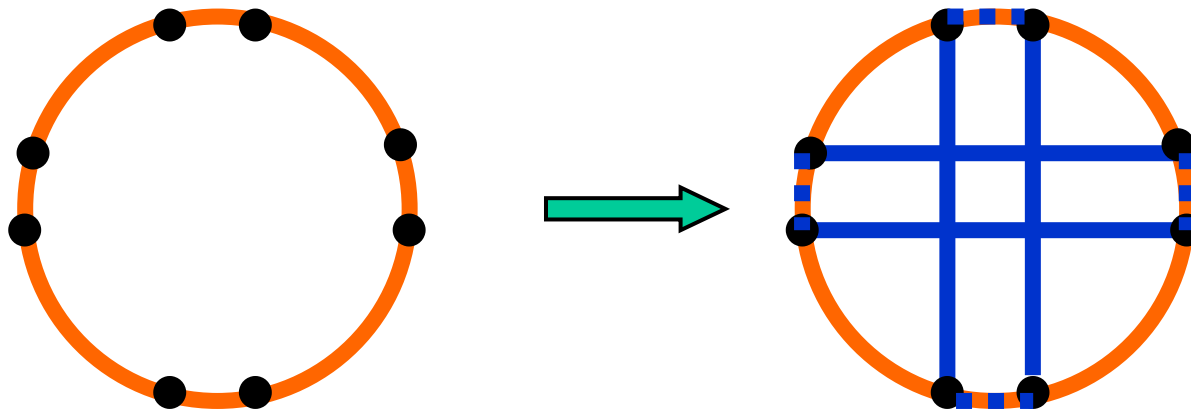
- Diese Strategie war ca. 15 Jahre lang die Standardmethode um sehr gute Touren zu erhalten.

CHAINED LIN-KERNIGHAN

- Idee von Martin, Otto und Felten 1992:
- Idee: statt immer wieder auf völlig neue Touren zu bauen, nutze die Resultate von LK:

- Perturbiere die von LK erhaltene Tour ein wenig („kick“)
- Wende LK auf die neue Tour an

- Kick von Martin, Otto und Felten:
- flip()-Sequenz, die einen speziellen Typ von 4-opt Austausch erzeugt: **double-bridge**



CHAINED LIN-KERNIGHAN

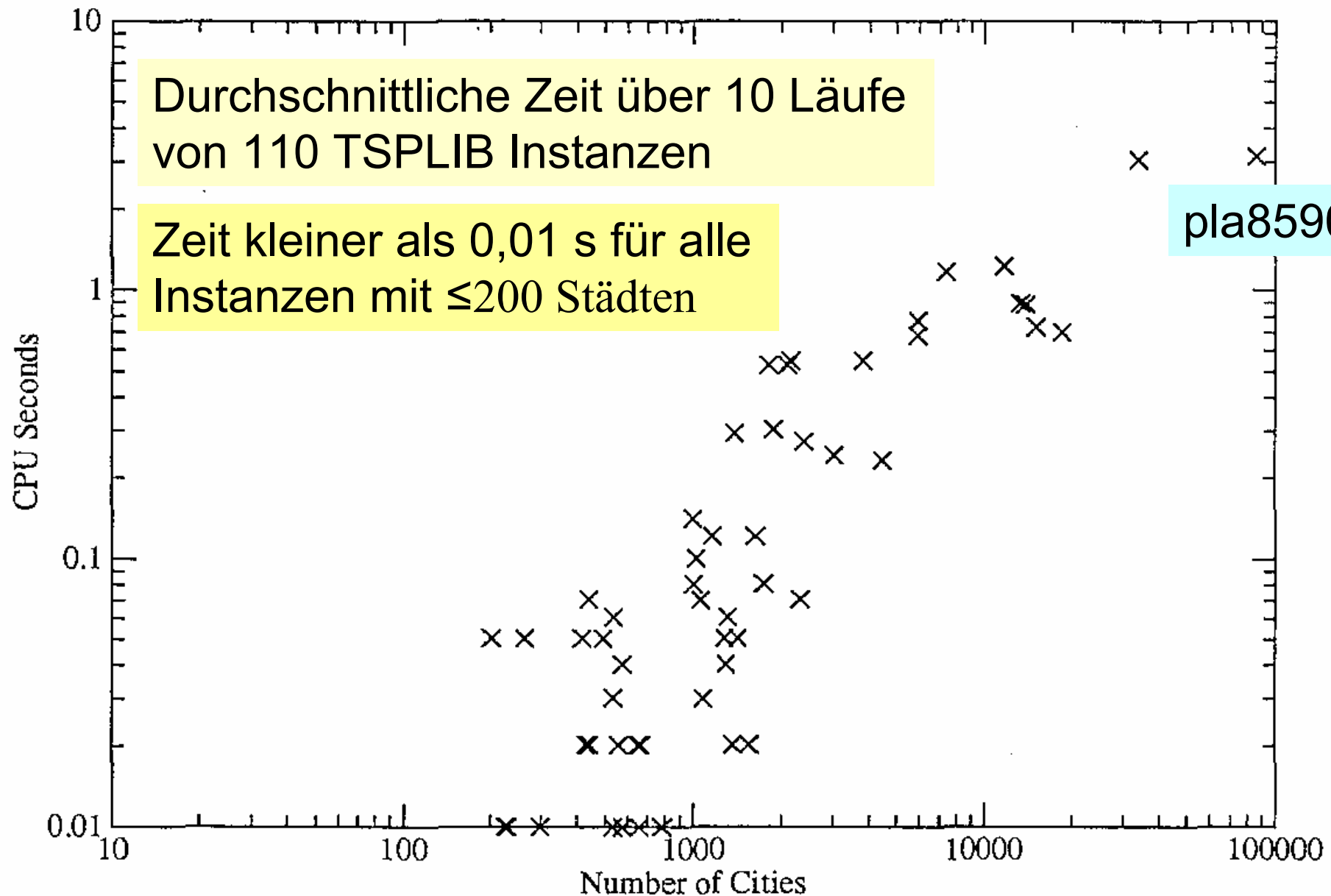
1. Aufruf von LIN_KERNIGHAN(S) → Tour T
2. **While** die maximale Zeit noch nicht erreicht ist **do**
3. Wähle eine „kicking“ flip()-Sequenz
4. Wende diese auf T an
5. Aufruf von LIN_KERNIGHAN(T) → Tour T'
6. **Falls** T' billiger als T ist,
7. **Dann** ersetze T durch T'
8. **Sonst** benutze die reverse kicking flip()-Sequenz
um die alte Tour zu erhalten
9. return T

Verbindung mit Simulated Annealing Schema: Akzeptiere
zwischen durch auch mal schlechtere Touren
→ Deutliche Verbesserung gegenüber iterated Lin-Kernighan ⁴

Experimental Results CHAINED LIN-KERNIGHAN

aus: TSP 2007 Buch

Notwendige Zeit für Güte $<1\%$ für Chained LK



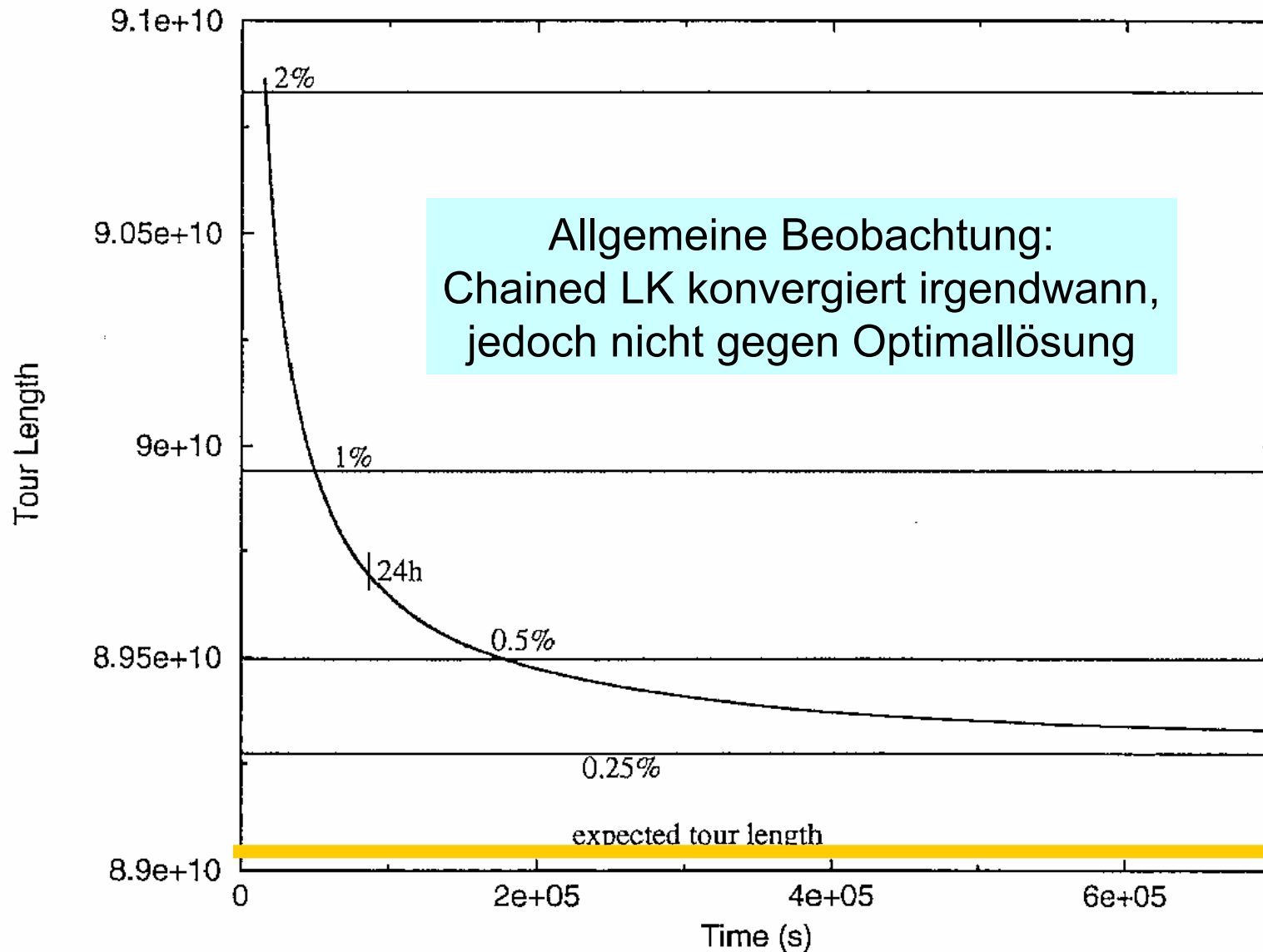
Experimental Results CHAINED LIN-KERNIGHAN

- 49 Instanzen aus TSPLIB mit ≤ 200 Knoten
 - 10 unabhängige Chained Lin-Kernighan Aufrufe pro Instanz mit Zeitlimit von 1 Sekunde
 - Von den 490 Läufen wurde nur 3 Mal die Optimallösung nicht gefunden
- 27 Instanzen bis 1000 Knoten: hier hat iterated LK Schwierigkeiten das Optimum zu finden
 - 1000 unabhängige Läufe pro Instanz mit Zeitlimit 5 Sek.
 - Optimallösungen konnten für alle Instanzen gefunden werden, aber die Laufzeit war teilweise höher als die exakte Lösung mit Branch & Cut zu berechnen.

Experimental Results CHAINED LIN-KERNIGHAN

- 17 Instanzen aus TSPLIB zwischen 1000 und 2000 Knoten
 - 100 unabhängige Chained Lin-Kernighan Aufrufe pro Instanz mit Zeitlimit 25 Sek.
 - Für 16 Instanzen Optimallösungen gefunden, für 3 davon nur in einem Lauf
- 6 Instanzen bis 4000 Knoten:
 - 100 unabhängige Läufe pro Instanz mit Zeitlimit 100 Sek.
 - Optimallösungen konnten für 3 Instanzen gefunden werden, davon in 2 Fällen nur in einem Lauf.

Chained LK für ein 25,000,000 Städte TSP



Helsgaun's LKH Algorithmus

- Helsgaun fiel auf, dass LK durch die Forderung nach „promising“ Nachbarn, nur beschränkte 2er Austausche machen kann
- Idee: statt 2er auch direkt 5er Austausche anzusehen
- Idee: zusätzlich auch erweiterte 5er Austausche, die aus speziellen nicht-zulässigen 2er und 3er Austauschungen bestehen; nicht-zulässig in dem Sinne, dass diese zwei Subtours produzieren dürfen
- Nachbargraph: Approximation der reduzierten Kosten der Subtour-Relaxation → nimmt die 5 Nachbarn mit den geringsten reduzierten Kosten
- Verbiete einen 2er Tausch, bei dem eine Kante entfernt wird, die in der bisher besten erhaltenen Tour ist.

Experimentelle Resultate für Helsgaun's LKH Algorithmus

- Helsgaun publizierte, dass er mittels LKH für **alle** in der TSPLIB bis zum Jahr 2000 enthaltenen Instanzen (inklusive usa13509) **optimale** Lösungen gefunden hat (die optimalen Lösungen wurden bereits vorher mittels Branch & Cut berechnet und bewiesen)
- Implementierung erhältlich unter www.akira.ruc.dk/~keld/research/LKH
- Tests mit usa13509: MinCost bei 10 Versuchen: 19983681 (0,0042% nahe am Optimum) bei mittlerer Laufzeit 7,391 Sekunden nach Preprocessing mit 673 Sekunden
- → deutlich besser als Chained Lin-Kernighan

Experimentelle Resultate für Helsgaun's LKH Algorithmus

Zeit in Sekunden bis zum Erreichen einer Güte von
0,1% Abweichung von der Optimallösung

Instanz	Pre-Processing in Sek.	Suchzeit
dsj1000	2	7
vm1084	2	2
nrw1379	4	1
pr2392	13	1
d15112	1047	3
pla33810	4914	1902

Für größere Graphen: Dekomposition