# Seminararbeit:
# K-Opt und die Lin-Kernighan-Heuristik für das allgemeine TSP

Tobias Boelter

28. Mai 2013

bei Prof. Dr. Rainer Schrader, Universität zu Köln

## Inhaltsverzeichnis

# 1 Einleitung

Das allgemein bekannte Traveling Salesman Problem ist stark NP-vollständig und somit praktisch nicht exakt lösbar. In der Praxis hat sich die Lin-Kernighan-Heuristik als nützlich erwiesen, um brauchbare Antworten in angemessener Zeit zu erhalten, wobei die Optimalität der Antwort natürlich nicht garantiert wird. Sie bedient sich des Konzepts der lokalen Suche, das ich zunächst vorstellen werde. Danach werden wir den allgemeinen K-Opt Algorithmus betrachten, der ebenfalls dieses Konzept nutzt, aber recht primitiv ist. Nach einigen vorbereitenden Definitionen und Sätzen widmen wir uns dann schließlich der Lin-Kernighan-Heuristik.

# 2 Lokale Suche

Ein oft verwendeter Ansatz für kombinatorische Optimierungsprobleme ist die lokale Suche:

1. Man beginnt mit einer unter Einfluss von Zufall generierten zulässigen Ausgangslösung $T$.

2. Man versucht, durch lokale Anpassungen eine verbesserte zulässige Lösung $T'$ zu finden.

3. Wenn man eine verbesserte Lösung $T'$ gefunden hat, setzt man $T := T'$ und geht wieder zu Schritt 2.

4. Sobald keine verbesserte Lösung $T'$ mehr gefunden werden kann, hat man ein lokales Optimum gefunden.

   Jetzt kann man wieder bei Schritt 1 mit einer anderen Ausgangslösung beginnen, in der Hoffnung ein anderes, besseres lokales Optimum zu finden. Dies kann man wiederholen, bis eine vorher festgelegte Abbruchbedingung erfüllt ist, z.B. eine vorher festgelegte Rechenzeit überschritten wurde.

Die Kunst besteht jetzt darin, die lokalen Anpassungen in Schritt 2 so zu machen, dass die Menge der lokalen Optima möglichst klein wird, da dadurch die Wahrscheinlichkeit, das globale Optimum zu finden entsprechend größer wird. Gleichzeitig muss man Schritt 2 auch schnell durchführen können. Die Menge der zulässigen Lösungen, die durch lokale Anpassungen von $T$ entstehen, nennt man auch *Nachbarschaft* von $T$.
Wie kann man jetzt bewerten, wie gut eine Lokale Suchheuristik ist? Allgemeingültige mathematisch präzise Aussagen zu treffen wird schwierig sein, da man im Worst-Case oft schnell bei exponentieller Laufzeit landet. In der Praxis kann man das Verfahren auf mehrere Instanzen eines Problems anwenden, von denen bereits eine optimale Lösung bekannt ist, allerdings liegt

das Einsatzgebiet dieser Heuristiken ja eher bei Instanzen, dessen optimale Lösung wir mit den uns zur Verfügung stehenden Mitteln nicht berechnen können. Dort können wir das Verfahren einige Male laufen lassen. Je öfter das gleiche lokale Optimum bei zufällig generierter Ausgangslösung gefunden wird, desto besser ist das Verfahren.
Im Folgenden bezieht sich jetzt alles auf das **TSP**, die Strategien lasen sich aber auch oft auf andere kombinatorische Optimierungsprobleme anwenden.
Eine Möglichkeit, den Schritt 2 in Bezug auf das TSP zu gestalten, ist für einen vorher fest gewählten Parameter $k$ alle Touren $T'$ zu betrachten, die sich in höchstens $k$ Kanten von $T$ unterscheiden. Dies führt zu dem k-Opt Algorithmus, den ich jetzt vorstellen werde.

# 3 K-Opt Algorithmus

**Algorithmus 1** K-Opt Algorithmus

Input:
- Instanz $(K_n, c)$ des TSP
  *Hier ist $K_n$ der vollständige Graph mit $n$ Knoten und $c : E(K_n) \to \mathbb{R}_+$ die Kostenfunktion*

Output:
- Eine Tour $T$, also der zu einem geschlossener Pfad, der jeden Knoten *genau* einmal enthält (Hamilton-Kreis) gehörender Teilgraph von $K_n$

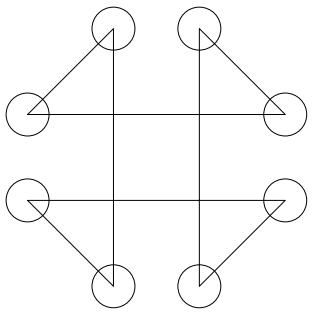
Sei $T$ eine beliebige Tour // ①
Sei $S$ die Familie der k-elementigen Teilmengen von E(T), also // ②
$S := \{X \subseteq E(T) : |X| = k\}$
// Enumeration aller $T'$, die sich in bis zu k Kanten von $T$ unterscheiden.
**for all** $X \in S$ **and all** tours $T'$ with $E(T') \supseteq E(T) \setminus X$ **do**
  **if** $c(E(T')) < c(E(T))$ **then** // ③
    $T := T'$
    goto 2
  **end if**
**end for**

**Definition 3.1** (k-opt)**.** Eine Tour heißt *k-opt*, falls sie mit dem k-Opt Algorithmus nicht weiter verbessert werden kann.

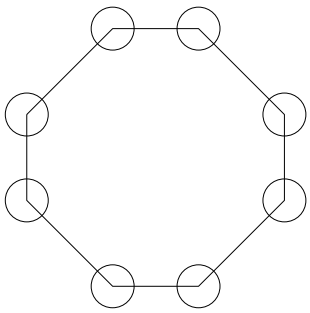
**Bemerkung 3.1.** Für $k = n$ liefert der k-Opt Algorithmus eine optimale Lösung, da dann $S = \{E(T)\}$ und damit alle Touren betrachtet werden.

**Beispiel 3.1.** Als Beispiel betrachte die Graphen 1 und 2 mit euklidischen Distanzen. Die Tour in Graph 1 ist offensichtlich 1-opt, 2-opt und 3-opt.

Durch austauschen von 4 Kanten erhält man aber die Tour in Graph 2, die offensichtlich optimal ist.

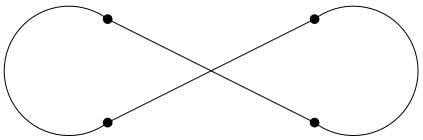


Graph 1: 3-opt

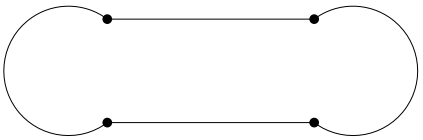


Graph 2: 4-opt

**Beispiel 3.2.** Die 2-opt Touren sind kreuzungsfrei im euklidischen TSP. Betrachte hierzu Graph 3 und Graph 4. 2-opt wird jede Kreuzung sicherlich gegen die direkte Verbindung austauschen, da Sie im euklidischen TSP kürzer ist.



Graph 3



Graph 4

**Bemerkung 3.2.** Für jedes feste k gibt es Instanzen des TSP und k-opt Touren, die nicht (k+1)-opt sind.

**Satz 3.1** (Güte, Beweis siehe [3])**.**

- *Eine von k-Opt Algorithmus erzeugte Tour kann für ein festes k ab einem bestimmten n um den Faktor* $\frac{1}{4}n^{\frac{1}{2k}}$ *länger sein. D.h. beliebig viel länger.*
- *Eine 2-opt Tour ist nicht schlechter als* $4\sqrt{n}$ *mal das Optimum.*

**Satz 3.2** (Laufzeit, Beweis siehe [4])**.** *Die Worst-Case Laufzeit von k-Opt ist exponentiell für alle k.*

**Bemerkung 3.3.** Laut Lin (1965) ist $k = 3$ häufig der Fall, der die besten Ergebnisse im Verhältnis zum Zeitaufwand erzielt.

Der k-Opt Algorithmus hat den Nachteil, dass man das k im Voraus bestimmen muss. S. Lin und B.W. Kernighan [2] haben deshalb die Lin-Kernighan-Heuristik entworfen, die in der Praxis sehr gute Ergebnisse liefert und dabei $k$ adaptiv anpasst.

# 4 Lin-Kernighan-Heuristik

## 4.1 Alternierende Spaziergänge

**Definition 4.1** (alternierender Spaziergang)**.** Gegeben sei eine Instanz $(K_n, c)$ des TSP und eine Tour $T$. Ein **alternierender Spaziergang** ist eine Knotenfolge $P = (x_0, x_1, \ldots, x_{2m})$, sodass keine Kante $\{x_i, x_{i+1}\}$ doppelt vorkommt (Spaziergang) und

$$\forall i \in \{0, \ldots, 2m-1\} : \{x_i, x_{i+1}\} \in E(T) \iff i \equiv 0 \ mod\ 2.$$

Die Kanten sind also abwechseln in $E(T)$ und $E \setminus E(T)$.

**Bemerkung 4.1.** In einem alternierenden Spaziergang können Knoten mehrfach vorkommen.

**Definition 4.2.** P heißt **geschlossen**, falls zusätzlich $x_0 = x_{2m}$.

**Definition 4.3.** Der **Gewinn** $g(P)$ wird definiert als

$$g(P) := \sum_{i=0}^{m-1} c(\{x_{2i}, x_{2i+1}\}) - c(\{x_{2i+1}, x_{2i+2}\}).$$

**Definition 4.4** (geeignet)**.** Ein geschlossener alternierender Spaziergang P heißt **geeignet**, falls

$$g(x_0, \ldots, x_{2i}) > 0 \quad \forall i \in \{1, \ldots, m\}.$$

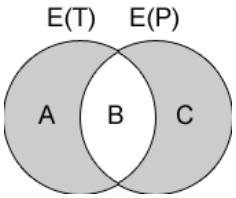
Von besonderem Interesse sind jetzt die geschlossenen alternierenden Spaziergänge $P$, sodass $(V(T), E(T) \triangle E(P))$ wieder eine Tour ist. Denn was macht $E(T) \triangle E(P)$ anschaulich? Die geraden Kanten in P (aus E(T)) werden gegen die ungerade (außerhalb von E(T)) ausgetauscht. In Bezug auf die k-Opt Strategie hieße das dann $k = \frac{|P|-1}{2}$

**Satz 4.1** (Lin, Kernighan (1973))**.** *Sei $P$ ein geschlossener alternierender Spaziergang mit $g(P) > 0$, dann gilt:*

**(a)** $c(E(T) \triangle E(P)) = c(E(T)) - g(P)$ ***und***

**(b)** *Es existiert ein geeigneter geschlossener alternierender Spaziergang $Q$ mit $E(Q) = E(P)$, also nur andere Reihenfolge, in der die Kanten durchlaufen werden.*

**Beweis.** **(a)** folgt sofort aus der Definition:




Man sieht direkt:
$g(P) = c(B) - c(C)$
und damit
$c(E(T)) - g(P) = c(A) + c(B) - c(B) - c(C)$
$= c(A) + c(C) = c(E(T) \triangle E(P))$ □

**(b)** Sei $P = (x_0, x_1, \ldots, x_{2m})$ (Bem: $x_{2m} = x_0$) und sei k der größte Index, sodass $g(x_0, \ldots, x_{2k})$ minimal ist.
Sei dann $Q := (x_{2k}, x_{2k+1}, \ldots, x_{2m-1}, x_0, x_1, \ldots, x_{2k})$.
Für $i = k+1, \ldots, m$ gilt dann:

$$g(x_{2k}, x_{2k+1}, \ldots, x_{2i}) = g(x_0, x_1, \ldots, x_{2i}) - g(x_0, x_1, \ldots, x_{2k}) > 0$$

nach Wahl von k.
Und für $i = 1, \ldots, k$ gilt:

$$\begin{aligned} & g(x_{2k}, x_{2k+1}, \ldots, x_{2m-1}, x_0, x_1, \ldots, x_{2i}) \\ = \; & g(x_{2k}, x_{2k+1}, \ldots, x_{2m}) + g(x_0, x_1, \ldots, x_{2i}) \\ \geq \; & g(x_{2k}, x_{2k+1}, \ldots, x_{2m}) + g(x_0, x_1, \ldots, x_{2k}) \\ = \; & g(P) > 0 \end{aligned}$$

Also ist $Q$ geeignet. □

## 4.2 Der Algorithmus

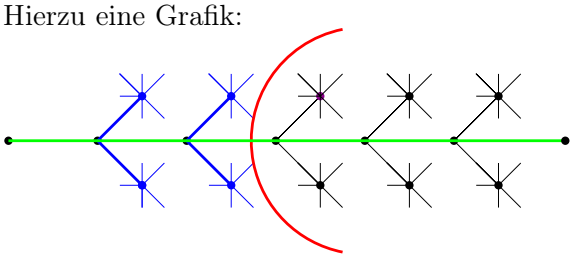
Der Algorithmus ist etwas länger und ein bisschen technisch. Zum besseren Verständnis werden erst die verwendeten Variablen vorgestellt und dann wird die allgemeine Funktionsweise grob erklärt. Das ist natürlich nicht mathematisch präzise, wird aber sicherlich von Nutzen sein.

### 4.2.1 Bedeutung der Variablen

| Name | Bedeutung |
|---|---|
| $p_1$ | Parameter (konstant): Backtrackingtiefe. $p_1 \in \mathbb{N}$ |
| $p_2$ | Parameter (konstant): Unzulässigkeitstiefe. $p_2 \in \mathbb{N}$ |
| $i$ | Laufvariable. Wird am Ende von ③ um 1 erhöht und in ④ sobald der Pfad nicht mehr verlängert werden kann auf $min\{i-1, p_1\}$ gesetzt. |
| $X_i$ | Menge der Knoten, die für eine Tour an der Stelle $i$ infrage kommen |
| $P^*$ | Bisher bester (in Bezug auf den Gewinn) gefundener geschlossener alternierender Spaziergang, sodass $(V(T), E(T) \triangle E(P^*))$ eine Tour ist |
| $g^*$ | $g(P^*)$ |

### 4.2.2 Funktionsweise

Hier sei kurz die Idee des Algorithmus vorgestellt. Wir bauen $(x_0, x_1, ..., x_i)$ sukzessive auf. Wenn es einen geschlossenen alternierenden Spaziergang $Q$ mit $g(Q) > 0$ gibt (genau das wollen wir haben vgl. Satz 4.1 a), wissen wir nach Satz 4.1 b), dass wir auf diese Weise auch einen geschlossenen alternierenden Spaziergang finden können, bei dem in jedem Schritt mit $i \equiv 0 \, mod \, 2$ gilt: $g(x_0, x_1, ..., x_i) > 0$.
Sobald $(V(T), E(T) \triangle E \underbrace{(x_0, x_1, ..., x_i, x_0)}_{P'})$ eine Tour ist und $g(P') > g(P^*)$, setzen wir $P^* := P'$. Wir verfahren dann so weiter, bis wir kein $x_{i+1}$ mehr wählen können. Wenn dann $g^* > 0$, update die Tour: $T := (V(T), E(T) \triangle E(P^*))$. Damit wird $T$ um $g^*$ verkürzt. In dem anderen Fall, also $g^* = 0$ unterscheiden wir 2 Fälle: Wenn $i = 0$ beenden wir das Verfahren, sonst setzen wir $i := min\{i-1, p_1\}$. Bemerke, dass $p_1$ die Backtrackingtiefe ist. Mit niedrigem $p_1$ werden Chancen vertan, hohe $p_1$ erhöhen den Rechenaufwand immens. Hierzu eine Grafik:



Wir betrachten also gerade den grünen Pfad. Wenn jetzt $g^* = 0$, wird an der Stelle $p_1$, die durch die rote Trennlinie symbolisiert wird, abgeschnitten und alles rechts davon nicht mehr betrachtet.

Und was bewirkt die Unzulässigkeitstiefe $p_2$?
$p_2$ verhindert, dass wir endlos weitersuchen, obwohl es vielleicht gar nicht mehr möglich ist, durch den Tausch der Kanten eine Tour aufzubauen. Ab $i > p_2$ werden deshalb nur noch alternierende Spaziergänge erlaubt, die beim Flippen der Kanten mit der aktuellen Tour wieder eine Tour ergeben.
(Bemerkung: Für $p_2$ machen eigentlich nur ungerade Werte Sinn. Ist $p_2$ gerade, beobachtet man den gleichen Effekt wie bei $p_2 + 1$. )

### 4.2.3 Pseudocode

Input:
- Instanz $(K_n, c)$ des TSP
- Ein aufspannender Teilgraph $G$ von $K_n$
- Eine beliebige Tour $T$
- Zwei Parameter
  - $p_1 \in \mathbb{N}$: Backtrackingtiefe
  - $p_2 \in \mathbb{N}$: Unzulässigkeitstiefe

Output:
- Eine Tour $T$

① Initialisierung

$X_0 := V(K_n)$
$i \quad := 0$
$g^* \ := 0$

② Tourbildung
Wähle $x_i$ aus $X_i$ beliebig
Entferne $x_i$ aus $X_i$
**if** $i$ ungerade AND $i \geq 3$ AND $(V(T), E(T) \triangle E(x_0, x_1, ..., x_{i-1}, x_i, x_0))$ ist eine Tour AND $g(x_0, x_1, ..., x_{i-1}, x_i, x_0) > g^*$ **then**
$P^* := (x_0, x_1, ..., x_{i-1}, x_i, x_0)$
$g^* := g(P^*)$
**end if**
③ Bestimmung von den möglichen Nachfolgern $X_i$
**if** $i$ ungerade **then**

$$X_{i+1} := \{x_{new} \in \{\text{Nachbarn von } x_i \text{ in } G\} \setminus \{x_0\} : \{x_i, x_{new}\} \notin E(T) \bigwedge \{x_i, x_{new}\} \notin E(x_0, \ldots, x_i) \bigwedge g(x_0, x_1, ..., x_{i-1}, x_i, x_{new}) > g^*\}$$

**end if**
**if** $i$ gerade AND $i \leq p_2$ **then**

$$X_{i+1} := \{x_{new} \in V(K_n) : \{x_i, x_{new}\} \in E(T) \bigwedge \{x_i, x_{new}\} \notin E(x_0, \ldots, x_i)\}$$

**end if**
**if** $i$ gerade AND $i > p_2$ **then**

$$X_{i+1} := \{x_{new} \in V(K_n) : \{x_i, x_{new}\} \in E(T) \bigwedge \{x_i, x_{new}\} \notin E(x_0, \ldots, x_i) \bigwedge (V(T), E(T) \triangle E(x_0, \ldots, x_i, x_{new}, x_0)) \text{ ist eine Tour}\}$$

**end if**
$i++$;
④ Abbruchbedingungen
**if** $\overline{X_i == \varnothing \; AND \; g^* > 0}$ **then**
  $T := (V(T), E(T) \triangle E(P^*))$
  Beginne von vorne
**end if**
**if** $X_i == \varnothing \; AND \; g^* == 0$ **then**
  **if** $i == 0$ **then**
    stop
  **else**
    $i := min\{i-1, p_1\}$
    goto ④
  **end if**
**end if**
goto ②

### 4.2.4 Weitere Eigenschaften des Algorithmus

**Bemerkung 4.2** (Bedeutung von $G$)**.** $G$ ist neben $p_1$ und $p_2$ noch ein zusätzliches Feature, mit dem wir die empirische Laufzeit drücken können, da wir damit die Größe der $X_{i+1}$ für ungerade $i$ extrem verkleinern können. Mit $G = K_n$ können wir dieses Feature aber auch einfach ungenutzt lassen. Beim euklidischen TSP hat sich herausgestellt, dass die Delaunay-Triangulation als Wahl von G gute Ergebnisse liefert. Die Delaunay-Triangulation Maximiert den kleinsten Winkel in jedem Dreieck.

**Satz 4.2** (Laufzeit)**.** *Die Laufzeit der Lin-Kernighan-Heuristik ist für* $p_1 = 5$, $p_2 = 2$ *im Worst-Case wahrscheinlich exponentiell. Empirisch wurde von Lin und Kernighan beobachtet, dass die Average-Case Laufzeit in* $O(n^{2,2})$ *liegt.*

**Satz 4.3** (Korrektheit des Algorithmus)**.**

**(a)** *Für* $G = K_n$ *und* $p_1 = p_2 = \infty$ *findet die Lin-Kernighan-Heuristik einen geeigneten alternierenden Spaziergang P, sodass* $(V(T), E(T) \triangle E(P))$ *eine Tour ist, falls er existiert.*

**(b)** *Für* $G = K_n$, $p_1 = 5$, $p_2 = 2$ *gibt die Lin-Kernighan-Heuristik eine 3-opt Tour zurück.*

**Beweis. (a)** Sei $T$ die Tour mit der der Algorithmus endet. Dann war seit dem letzten Update von $T$ immer $g^* = 0$. Mit $G = K_n$ und $p_1 = p_2 = \infty$ wurden dann alle geeigneten alternierenden Spaziergänge betrachtet aber keiner gefunden. □

**(b)** Für $G = K_n$, $p_1 = 5$, $p_2 = 2$ hat der Algorithmus alle geeigneten geschlossenen alternierenden Spaziergänge der Länge 4 oder 6 Kanten gefunden. Wo liegt nämlich die Einschränkung, wenn die Unzlässigkeitstiefe $p_2 = 2$ ist? Im Schritt $i = 4$ muss dann $(V(T), E(T) \triangle E(x_0, x_1, x_2, x_3, x_4, x_5, x_0))$ eine Tour sein. $x_0, \ldots, x_5, x_0$ sind aber schon 6 Kanten. Auch die Backtrackingtiefe $p_1 = 5$ ist keine weitere Einschränkung mehr.
Angenommen $T$ wäre jetzt nicht 3-opt. Dann existiert eine Möglichkeit, 3 Kanten auszutauschen, sodass die resultierende Tour $T'$ kürzer ist. Die Knoten von $E(T) \triangle E(T')$ formen einen geschlossenen alternierenden Spaziergang P. oBdA kann man annehmen, dass P geeignet ist $\Rightarrow$ Wir finden P. $\square$

**Bemerkung 4.3.** Für $G = K_n$, $p_1 = 5$, $p_2 = 2$ liefert der Lin-Kernighan Algorithmus meist bessere Lösungen als 3-Opt und ist dabei auch noch schneller.

**Bemerkung 4.4.** Der Lin-Kernighan Algorithmus hat keine Chance, einen nichtsequenziellen Austausch zu finden, wie er bei der Transformation von Graph 1 in Graph 2 notwendig wäre, da er mit den alternierenden Spaziergängen arbeitet.

## Literatur


[1] B. Korte, J. Vygen: *Combinatorial Optimization:* Theory and Algorithms, No. 1 (2000)

[2] S. Lin, B. Kernighan: *An Effective Heuristic Algorithm for the Traveling Salesman Problem* (1971)

[3] Chandra, Karloff, Tovey: *New results on the old k-opt algorithm for the traveling salesman problem.* SIAM Journal on Computing 28, 1998–2029 (1999)

[4] Englert, Röglin, Vöcking: *Worst case and probabilistic analysis of the 2-opt algorithm for the TSP.* Proceedings of the 18th Annual ACM-SIAM Symposium on Discrete Algorithms, S. 1295–1304 (2007)