

# Traveling Salesman Problem

## Einführung zum Problem des Handlungsreisenden

### **Abstract**

Gegeben sei eine Menge von Städten. Ein Handlungsreisender soll eine Route planen, auf der er jede Stadt *genau einmal* besucht, wobei die geplante Route möglichst kurz sein soll. Diese Ausarbeitung soll eine kurze Einführung in die Problematik dieses Optimierungsproblems geben, sowohl aus algorithmischer als auch aus Komplexitätstheoretischer Sicht.

Florian Weingarten (270511)  
Florian.Weingarten@RWTH-Aachen.de

### **Betreuer:**

Dr. Walter Unger  
Lehrstuhl für Informatik 1  
Algorithmen und Komplexität  
RWTH Aachen

## Inhaltsverzeichnis

<b>1</b>	<b>Einleitung</b>	<b>2</b>
1.1	Mathematische Modellierung . . . . .	2
1.1.1	Optimierungsvariante . . . . .	2
1.1.2	Entscheidungsvariante . . . . .	3
1.1.3	Mögliche Lösungen . . . . .	3
1.2	Beispiel . . . . .	3
<b>2</b>	<b>Varianten und Spezialfälle von TSP</b>	<b>5</b>
2.1	Asymmetrisches vs. symmetrisches TSP . . . . .	5
2.2	Metrisches TSP ( $\Delta$ -TSP) . . . . .	5
2.2.1	Euklidisches TSP . . . . .	5
2.2.2	Andere Metriken . . . . .	6
2.3	Multiple-TSP (mTSP) . . . . .	6
<b>3</b>	<b>Komplexität von TSP</b>	<b>6</b>
3.1	Äquivalenz von Entscheidungs- und Optimierungsvarianten . . . . .	7
3.2	TSP ist in NP . . . . .	8
3.3	NP-Schwere von TSP . . . . .	8
<b>4</b>	<b>Approximierbarkeit</b>	<b>10</b>
<b>5</b>	<b>TSP exakt lösen</b>	<b>11</b>
5.1	Brute Force . . . . .	11
5.2	Dynamische Programmierung . . . . .	11
<b>6</b>	<b>TSP approximieren</b>	<b>12</b>
6.1	Nearest-Neighbour Heuristik . . . . .	12
6.1.1	Beispiel . . . . .	13
6.2	MST-Heuristik . . . . .	13
6.2.1	Beispiel . . . . .	15
6.3	Algorithmus von Cristofides . . . . .	15
6.3.1	Beispiel . . . . .	17
	<b>Literatur</b>	<b>18</b>

# 1 Einleitung

Das *Problem des Handlungsreisenden* (englisch *Traveling Salesman Problem*, abgekürzt TSP) ist ein kombinatorisches Optimierungsproblem aus dem Bereich der theoretischen Informatik. Anschaulich lässt sich das Problem sehr leicht formulieren: Ein Handlungsreisender soll eine Rundreise durch eine bestimmte Anzahl von Städten machen, jede Stadt dabei genau einmal besuchen und am Ende wieder zum Ausgangsort zurückkehren. Die Tour soll so geplant werden, dass die zurückgelegte Strecke möglichst kurz ist.

Das TSP tritt in vielen praktischen Anwendungen auf, z.B. in der Tourenplanung und der Logistik, aber auch in nicht so naheliegenden Bereichen wie z.B. dem Entwurf von Mikrochips. Die Begriffe „Ort“ und „Entfernung“ sind also nicht wörtlich gemeint. Je nach Kontext kann mit „Entfernung“ auch eine Zeitspanne gemeint sein, ein Preis oder anderes.

TSP gehört zu einer für die theoretische Informatik sehr wichtigen Klasse von Problemen, den sogenannten NP-vollständigen Problemen. Es wird sehr stark angenommen, dass für NP-vollständige Probleme keine effizienten Lösungsverfahren existieren, d.h. dass die Worst-Case Laufzeit jedes deterministischen Algorithmus mindestens exponentiell von der Anzahl der zu besuchenden Städte abhängt.

Ziel dieses Textes ist es, dem Leser einen groben Überblick über TSP und seine Varianten zu geben, die Zugehörigkeit zur Klasse der NP-vollständigen Probleme zu beweisen und einige heuristische Lösungsverfahren vorzustellen.

Im folgenden wird davon ausgegangen, dass der Leser über grundlegende Kenntnisse der theoretischen Informatik verfügt, insbesondere mit *Graphentheorie*, *Komplexitätstheorie* und *Datenstrukturen und Algorithmen* vertraut ist.

Die verwendete Notation ist angelehnt an das Vorlesungsskript zur Vorlesung *Berechenbarkeit und Komplexität* [14] an der RWTH-Aachen von Prof. Dr. Berthold Vöcking.

## 1.1 Mathematische Modellierung

### 1.1.1 Optimierungsvariante

Wir definieren das TSP, analog zu [14], mit Hilfe eines vollständigen ungerichteten gewichteten Graphen.

**Definition 1.1.** Sei  $G = (V, E)$ ,  $V = \{1, 2, \dots, n\}$  die Knotenmenge und  $E := V \times V$  die (vollständige) Kantenmenge. Zusätzlich sei  $G$  gewichtet, d.h. es existiert eine Kostenfunktion  $c : E \rightarrow \mathbb{N}$  die jeder Kante  $(i, j) \in E$  ein „Gewicht“ zuordnet. Eine TSP-Tour oder Rundreise ist ein Hamiltonkreis in  $G$ , d.h. ein Kreis in  $G$ , der jeden Knoten genau einmal enthält.

Die Knoten repräsentieren z.B. Städte, die Kanten die Verbindungen zwischen den Städten und ein Gewicht  $c(i, j)$  kann z.B. interpretiert werden als die Entfernung zwischen den Städten  $i$  und  $j$  oder die Reisezeit o.ä. Ausserdem gehen wir im weiteren (vorerst) davon aus, dass die Kostenfunktion symmetrisch ist, d.h.  $c(i, j) = c(j, i)$ ,  $\forall i, j$ .

Jede solche Rundreise lässt sich darstellen als Permutation  $\pi \in S_n$ .

**Definition 1.2.** Eine TSP-Tour  $\pi$  heißt optimal, wenn es keine Tour gibt, deren Kantensumme geringer ist.

Gesucht ist nun also die „günstigste“ Rundreise, d.h. wir suchen eine Permutation  $\pi$ , so dass

$$\sum_{i=1}^{n-1} c(\pi(i), \pi(i+1)) + c(\pi(n), \pi(1))$$

minimal ist unter allen möglichen Touren.

### 1.1.2 Entscheidungsvariante

Bei der Entscheidungsvariante des TSP gehen wir analog vor. Zusätzlich ist eine Zahl  $b \in \mathbb{N}$  gegeben. Die Aufgabe ist es nun festzustellen ob eine Rundreise existiert, deren Kosten nicht größer als  $b$  sind.

Wir werden sehen, dass die Entscheidungsvariante von TSP zur Klasse der NP-vollständigen Probleme gehört.

### 1.1.3 Mögliche Lösungen

Wie oben schon erwähnt, ist eine mögliche Lösung eine Permutation auf  $n$  Elementen. Ohne Einschränkung kann man  $\pi(1) = 1$  annehmen. Es gibt  $(n-1)!$  mögliche Permutationen auf  $n-1$  Elementen. Geht man zusätzlich von symmetrischer Kostenfunktion aus, so ergibt sich eine Gesamtzahl von  $\frac{1}{2} \cdot (n-1)!$  möglichen Lösungen (was natürlich der Anzahl der möglichen Hamiltonkreise auf einem vollständigen Graphen mit  $n$  Knoten entspricht).

Da jede mögliche Lösung ein Hamiltonkreis ist, also alle  $n$  Knoten besucht, und damit genau  $n$  Kanten enthält, lassen sich direkt leicht untere und obere Schranken für die Kosten einer Lösung angeben (und damit auch einer optimalen Lösung). Da genau  $n$  Kanten benutzt werden müssen, lässt sich als untere Schranke direkt die Summe der  $n$  „günstigsten“ Kanten angeben, analog ist natürliche jede Lösung nicht teurer als die Summe der teuersten  $n$  Kanten. Ausserdem ist jede Tour mindestens so lang wie ein minimal aufspannender Baum in  $G$ . Diese Schranken lassen sich in polynomieller Zeit bestimmen (z.B. mit dem Algorithmus von Prim in  $O(|V|^2)$ , siehe [7]).

## 1.2 Beispiel

Abbildung 1 zeigt alle ungerichteten Hamiltonkreise auf  $K_5$ , dem vollständigen Graphen mit fünf Knoten, d.h. alle möglichen Rundreisen von (symmetrischem) TSP auf 5 Städten.

Angenommen wir haben folgende TSP-Instanz auf  $K_5$  mit symmetrischer Distanzmatrix

$$c := \begin{pmatrix} - & 3 & 4 & 5 & 6 \\ 3 & - & 5 & 1 & 4 \\ 4 & 5 & - & 2 & 3 \\ 5 & 1 & 2 & - & 5 \\ 6 & 4 & 3 & 5 & - \end{pmatrix}$$

Dann ergeben sich für die möglichen Rundreisen folgende Kosten:

(a)  $cost(ABCDEA) = 3 + 5 + 2 + 5 + 6 = 21$

(b)  $cost(ABCEDA) = 3 + 5 + 3 + 5 + 5 = 21$

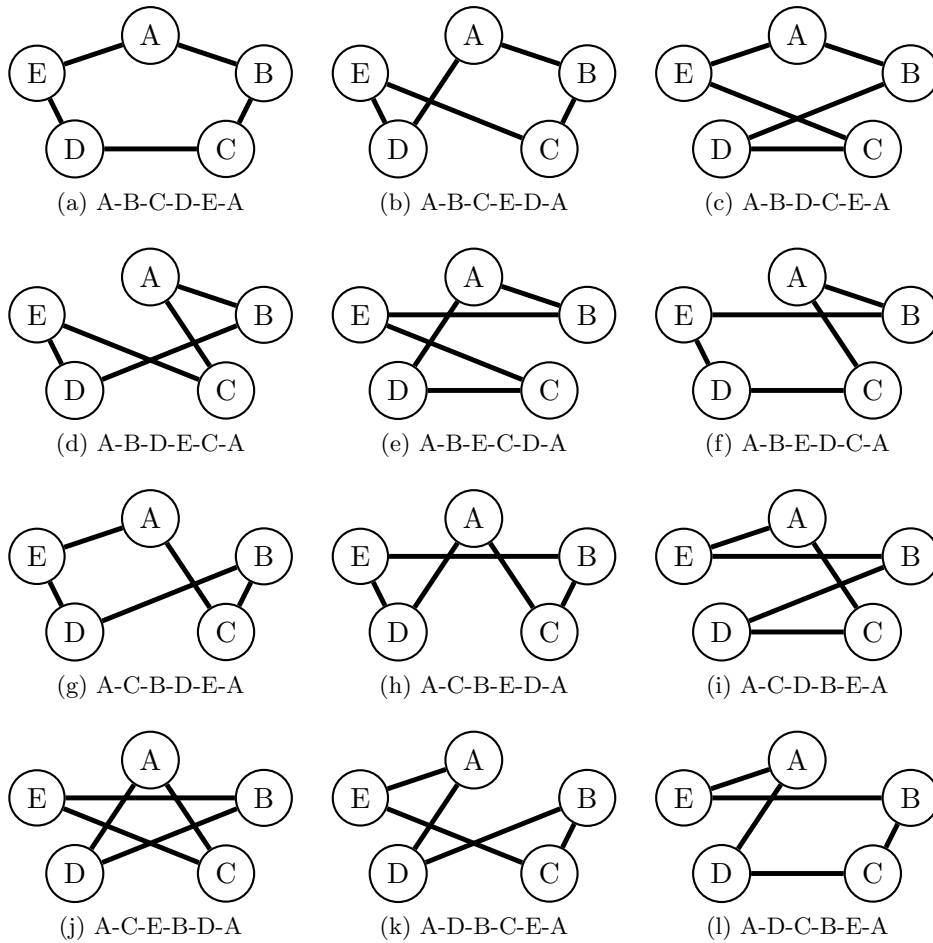


Abbildung 1: Alle  $\frac{1}{2} \cdot (5 - 1)! = 12$  Hamiltonkreise in  $K_5$  mit fixem Startpunkt A

(c)  $cost(ABDC EA) = 3 + 1 + 2 + 3 + 6 = \mathbf{15} \leftarrow$  optimale Lösung!

(d)  $cost(ABDECA) = 3 + 1 + 5 + 3 + 4 = 16$

(e)  $cost(ABECDA) = 3 + 4 + 3 + 2 + 5 = 17$

(f)  $cost(ABEDCA) = 3 + 4 + 5 + 2 + 4 = 18$

(g)  $cost(ACBDEA) = 4 + 5 + 1 + 5 + 6 = 21$

(h)  $cost(ACBEDA) = 4 + 5 + 4 + 5 + 5 = 23$

(i)  $cost(ACDBEA) = 4 + 2 + 1 + 4 + 6 = 17$

(j)  $cost(ACEBDA) = 4 + 3 + 4 + 1 + 5 = 17$

(k)  $cost(ADBCEA) = 5 + 1 + 5 + 3 + 6 = 20$

(l)  $cost(ADCBEA) = 5 + 2 + 5 + 4 + 6 = 22$

Die Tour  $ABDC EA$  ist also bei gegebener Kostenmatrix  $c$  die (einzige) optimale Lösung.

## 2 Varianten und Spezialfälle von TSP

### 2.1 Asymmetrisches vs. symmetrisches TSP

Das asymmetrische TSP ist eine Verallgemeinerung des uns bisher bekannten symmetrischen TSP. Wir sind in unserer Modellierung von einer symmetrischen Kostenfunktion ausgegangen, d.h.  $c(i, j) = c(j, i)$  für alle  $i, j$ . Lässt man diese Bedingung weg, ist es möglich auch Situationen zu modellieren, in denen z.B. die Entfernung von Stadt  $i$  zu Stadt  $j$  nicht die selbe ist wie die in umgekehrte Richtung. Mögliche Gründe hierfür sind z.B. gesperrte Straßen in eine Richtung (Einbahnstraßen, Baustellen, etc.).

Durch die fehlende Symmetrie verdoppelt sich die Anzahl der möglichen Lösungen, d.h. wir haben es nun mit  $(n - 1)!$  statt mit  $\frac{1}{2} \cdot (n - 1)!$  möglichen TSP-Touren zu tun.

### 2.2 Metrisches TSP ( $\Delta$ -TSP)

Ein wichtiger Spezialfall von TSP, vor allem für die Approximierbarkeit, ist das sogenannte Metrische TSP, auch  $\Delta$ -TSP genannt. In diesem Spezialfall ist zusätzlich die Bedingung gestellt, dass es sich bei der Kostenfunktion  $c$  um eine Metrik handelt, dass also durch  $(V, c)$  ein metrischer Raum definiert ist.

Konkret bedeutet das, dass  $c$  die folgenden Axiome erfüllen muss:

- (1)  $c(i, j) \geq 0$  für alle  $i, j \in V$  und  $c(i, j) = 0 \Leftrightarrow i = j$  (positive Definitheit)
- (2)  $c(i, j) = c(j, i)$  (Symmetrie)
- (3)  $c(i, j) + c(j, k) \geq c(i, k)$  (Dreiecksungleichung)

Die erste Eigenschaft lässt sich als die Forderung interpretieren, dass jeder Knoten einen Abstand von 0 zu sich selbst haben muss und dass unterschiedliche Punkte einen echt positiven Abstand haben. Die zweite Bedingung ist die schon erwähnte Symmetrie. Die wichtige neue Eigenschaft ist hier die dritte. Die Dreiecksungleichung besagt in diesem Fall, dass ein Umweg von einer Stadt  $i$  zu einer Stadt  $k$  über eine dritte Stadt  $j$  niemals kürzer ist als der direkte Weg von  $i$  nach  $k$ , d.h. „Umwege lohnen sich nicht“.

#### 2.2.1 Euklidisches TSP

Euklidisches TSP (oder auch planares TSP genannt) ist ein Spezialfall von  $\Delta$ -TSP. Wir gehen davon aus, dass es sich um einen ebenen Graphen handelt, d.h. um einen Graphen der in die euklidische Ebene  $\mathbb{R}^2$  eingebettet ist. Jeder Knoten  $v$  lässt sich also darstellen als  $v = (v_1, v_2) \in \mathbb{R}^2$ . Als Metrik  $c : \mathbb{R}^2 \times \mathbb{R}^2 \rightarrow \mathbb{R}$ , wird hier einfach die von der euklidischen Vektornorm  $\|\cdot\|_2$  induzierte Metrik gewählt, d.h. für  $i = (i_1, i_2)$  und  $j = (j_1, j_2)$  gilt

$$c(i, j) := \|i - j\|_2 = \left\| \begin{pmatrix} i_1 - j_1 \\ i_2 - j_2 \end{pmatrix} \right\|_2 = \sqrt{(i_1 - j_1)^2 + (i_2 - j_2)^2}$$

### 2.2.2 Andere Metriken

Nicht nur die von der euklidischen Norm  $\|\cdot\|_2$  induzierte Metrik spielt in der Praxis eine Rolle. Wichtig sind z.B. auch die von der Maximumsnorm  $\|\cdot\|_\infty$  und von der Betragssummennorm  $\|\cdot\|_1$  induzierten Metriken. Es gilt:

$$\|x\|_\infty = \max_i \{|x_i|\}$$

$$\|x\|_1 = \sum_{i=1}^n |x_i|$$

Die erste wird *Maximums-Metrik* genannt. Hier ist die Distanz zwischen zwei Knoten das Maximum der Distanzen der einzelnen Komponenten. Als Anwendung denke man an das Beispiel eines Bohrers der Löcher in einer Leiterplatte bohren soll und der sich zwar gleichzeitig in x und y Richtung bewegen kann, aber nur in beide Richtungen mit der gleichen Geschwindigkeit, d.h. die Gesamtzeit hängt von der Länge der längsten Richtungen ab.

Die zweite Metrik nennt sich *City-Block-Metrik* oder *Manhattan-Metrik*. Der Name ist an die schachbrettartige Anordnung von Häuserblocks in amerikanischen Großstädten (z.B. Manhattan) angelehnt. Anwendung findet diese Metrik immer dann wenn eine Bewegung nur in eine Richtung gleichzeitig möglich ist (z.B. nur in x Richtung oder nur in y Richtung, aber niemals in beide gleichzeitig), wie es z.B. für einen Lieferanten in Manhattan der Fall ist.

### 2.3 Multiple-TSP (MTSP)

Das MULTIPLE-TSP ist eine, für die Praxis wichtige, Verallgemeinerung des TSP. Wie der Name bereits suggeriert, geht man hier von mehreren Handlungsreisenden aus, die alle gleichzeitig reisen. Gesucht sind nun mehrere Rundreisen, so dass die Summe der Kosten der Rundreisen minimal ist und gleichzeitig jede Stadt nur von genau einem Handlungsreisenden besucht wird (und möglichst jeder Handlungsreisende mindestens eine Stadt besucht).

## 3 Komplexität von TSP

In diesem Abschnitt wollen wir die Komplexität von TSP untersuchen.

**Definition 3.1.** Ein Problem  $L$  heißt vollständig bzgl. einer Komplexitätsklasse  $K$  wenn sich jedes Problem  $L' \in K$  auf  $L$  in Polynomialzeit reduzieren lässt und  $L \in K$  gilt.

**Definition 3.2.** NPC ist die Klasse der sogenannten NP-vollständigen Probleme (engl. NP-complete). D.h.  $L \in \text{NPC}$  genau dann, wenn:

- (1)  $L \in \text{NP}$
- (2)  $L' \leq_p L$  für alle  $L' \in \text{NP}$  (d.h.  $L$  ist NP-hart, siehe unten)

**Definition 3.3.** Sei  $\Sigma^*$  die Menge der möglichen Eingaben. Eine Sprache  $L'$  ist auf eine Sprache  $L$  reduzierbar, wenn eine totale, berechenbare Funktion  $f : \Sigma^* \rightarrow \Sigma^*$  existiert, so dass gilt:  $w \in L' \Leftrightarrow f(w) \in L$ . Man schreibt  $L' \leq L$ . Ist  $f$  in Polynomialzeit berechenbar, so heißt  $f$  auch Polynomialzeitreduktion und man schreibt  $L' \leq_p L$ .

In diesem Abschnitt soll gezeigt werden, dass

$$\text{TSP} \in \text{NPC}$$

D.h. wir wollen zeigen, dass TSP in NP ist und dass TSP ein NP-hartes Problem ist.

In 1.1 wurden bereits zwei Modellierungen vorgestellt. In diesem Abschnitt soll nur auf die *Entscheidungsvariante* eingegangen werden, was allerdings keine Einschränkung darstellt, wie wir gleich sehen werden.

### 3.1 Äquivalenz von Entscheidungs- und Optimierungsvarianten

Es seien die folgenden drei Varianten von TSP gegeben:

- (1) Existiert eine optimale TSP-Tour mit Kosten  $\leq b$ ? (Entscheidungsvariante)
- (2) Wie hoch sind die Kosten einer optimalen TSP-Tour? (Optimierungsvariante, Typ 1)
- (3) Bestimme eine optimale TSP-Tour (Optimierungsvariante, Typ 2)

Intuitiv würde man vermuten, dass Variante 3 am schwierigsten zu lösen ist, tatsächlich gilt aber der folgende Satz.

**Satz 3.1.** *Alle drei Varianten sind aus komplexitätstheoretischer Sicht „gleich schwierig“, d.h. es existieren in Polynomialzeit berechenbare Reduktionen der Varianten aufeinander.*

Wenn also eine der drei Varianten in Polynomialzeit lösbar ist, dann auch die beiden übrigen:  
 (1)  $\Leftrightarrow$  (2)  $\Leftrightarrow$  (3)

**Beweis** Offensichtlich gilt (3)  $\Rightarrow$  (2)  $\Rightarrow$  (1), da ein Algorithmus, der in Polynomialzeit eine optimale Tour bestimmt auch direkt deren Wert bestimmen kann und damit (1) und (2) löst. Ein Algorithmus der (2) löst kann seine Lösung mit  $b$  aus (1) vergleichen und damit auch (1) lösen.

Listing 1: Algorithmus (1)  $\Rightarrow$  (2)

```

1 opt := Summe der n teuersten Kanten
2
3 for(b = opt-1, ..., 0) {
4   if(es existiert optimale Tour mit Kosten  $\leq$  b) {
5     opt := b
6   } else {
7     Schleife abbrechen
8   }
9 }
10
11 return opt

```

Listing 2: Algorithmus (2)  $\Rightarrow$  (3)

```

1  opt := Wert einer optimalen Tour
2
3  // Schleife ist polynomiell beschränkt in der Anzahl der Knoten,
4  // da maximal  $|V|^2$  viele Knoten durchlaufen werden
5  for all Edges  $e \in E$  {
6
7    Sei  $G'$  der Graph  $G$  nachdem die Kante  $e$  entfernt wurde
8    opt' := Wert einer optimalen Tour in  $G'$ 
9
10   if(opt == opt') {
11     Kante war nicht wichtig für die Tour, also entferne sie aus  $G$ 
12   }
13
14 }
15
16 return G

```

Der erste Algorithmus löst Variante (2) in Polynomialzeit, unter der Voraussetzung, dass ein Algorithmus bekannt ist, der Variante (1) in Polynomialzeit löst. Der zweite Algorithmus benutzt einen bekannten Polynomialzeitalgorithmus für (2) (z.B. den ersten) um (3) zu lösen. Damit folgt also: (1)  $\Rightarrow$  (2)  $\Rightarrow$  (3) und damit die Behauptung.  $\square$

### 3.2 TSP ist in NP

NP ist die Klasse der *Entscheidungsprobleme*, die in Polynomialzeit von einer nichtdeterministischen Turingmaschine entschieden werden können. Nach [6] können wir alternativ NP als die Klasse der Entscheidungsprobleme beschreiben, für die sich mögliche Lösungen als Zertifikat polynomieller Länge (auch „polynomieller Beweis“ oder „polynomieller Zeuge“ genannt) kodieren lassen und für die ein Polynomialzeitalgorithmus (Verifizierer) existiert, der die mögliche Lösung verifizieren kann. Grob gesagt kann man für Probleme in NP also effizient testen ob eine bereits gefundene (polynomiell kodierte) Lösung tatsächlich korrekt ist oder nicht.

**Satz 3.2.** TSP  $\in$  NP

**Beweis** Wie in 1.1.3 bereits erwähnt lässt sich jede mögliche Lösung als Permutation auffassen. Eine Permutation ist eindeutig bestimmt durch die Angabe der Bilder  $\pi(n)$  für alle  $n$ . Die Kodierung ist also in  $O(n)$  möglich.

Zu gegebener Schranke  $b$  für das Entscheidungsproblem lässt sich nun durch Summation der Kantengewichte der vorgeschlagenen Lösung testen, ob diese die Schranke tatsächlich nicht überschreitet. Dieser Test kann ebenfalls effizient durchgeführt werden.

Wir haben also einen Polynomialzeitverifizierer und ein Zertifikat polynomieller Länge.  $\square$

### 3.3 NP-Schwere von TSP

**Definition 3.4.** Ein Problem  $L$  heißt NP-schwer (NP-hart) wenn sich jedes Problem  $L' \in$  NP auf  $L$  polynomiell reduzieren lässt, d.h.  $L' \leq_p L$ .

Offensichtlich ist polynomielle Reduktion transitiv, d.h. aus  $L \leq_p L'$  und  $L' \leq_p L''$  folgt stets  $L \leq_p L''$ . Damit gilt, dass ein Problem  $K$  NP-hart ist wenn ein anderes NP-hartes Problem  $L$  existiert mit  $L \leq_p K$ .

Nach dem Satz von Cook ist  $SAT \in NPC$ . Richard P. Karp hat 1972 in [9] folgende Reduktionskette gezeigt:

$$SAT \leq_p CLIQUE \leq_p VERTEXCOVER \leq_p DHC \leq_p HC$$

Das (ungerichtete) Hamiltonkreisproblem (HC) ist also NP-schwer. Wir werden diese Tatsache nutzen um zu zeigen, dass auch TSP ein NP-schweres Problem ist, indem wir das Hamiltonkreisproblem in Polynomialzeit auf einen Spezialfall von TSP reduzieren, d.h. wir werden ein, in Polynomialzeit berechenbares, Verfahren angeben, um Eingaben für das Hamiltonkreisproblem so zu transformieren, dass eine HC-Instanz genau dann eine Ja-Instanz ist, wenn die transformierte Instanz eine Ja-Instanz von TSP ist.

**Definition 3.5.** Sei  $\{1, 2\}$ -TSP eine TSP Variante in der  $c(i, j) \in \{1, 2\}$  für alle  $(i, j) \in E$ , d.h. es sind nur Kantengewichte von 1 und 2 zulässig.

**Satz 3.3.**  $\{1, 2\}$ -TSP ist NP-hart.

**Beweis** Sei  $G = (V, E)$  ein ungerichteter Graph, also eine mögliche Instanz des Hamiltonkreisproblems. Wir transformieren diese Eingabe in eine gültige Eingabe für die Entscheidungsvariante von  $\{1, 2\}$ -TSP. Sei dazu  $b := |V|$ ,  $c(i, j) := \begin{cases} 1 & (v_i, v_j) \in E \\ 2 & \text{sonst} \end{cases}$  und betrachte  $K_b$ , d.h. den vollständigen Graphen auf  $b$  Knoten.

Zu zeigen:  $G$  enthält einen Hamiltonkreis  $\Leftrightarrow K_b$  enthält eine TSP-Tour mit Kosten  $\leq b$

„ $\Rightarrow$ “: Jeder Hamiltonkreis enthält genau  $b = |V|$  Kanten und  $b$  verschiedene Knoten. Sei  $(\pi(1), \pi(2), \dots, \pi(b), \pi(1))$  ein Hamiltonkreis in  $G$ , dann gilt  $(\pi(i), \pi(i+1)) \in E$  für  $1 \leq i \leq b$  und  $(\pi(b), \pi(1)) \in E$  (denn jede Kante des Hamiltonkreises ist natürlich in  $G$  enthalten). Dann folgt

$$\sum_{i=1}^{b-1} c(\pi(i), \pi(i+1)) + c(\pi(b), \pi(1)) = \sum_{i=1}^{b-1} 1 + 1 = (b-1) + 1 = b \leq b$$

also existiert in  $G$  ein Hamiltonkreis mit Kosten  $\leq b$  und damit in  $K_b$  eine TSP-Tour mit Kosten  $\leq b$ .

„ $\Leftarrow$ “: Es existiert eine optimale TSP-Tour in  $K_b$ , d.h. es gibt  $v_1, \dots, v_b \in V$  mit

$$\sum_{i=1}^{b-1} c(v_i, v_{i+1}) + c(v_b, v_1) \leq b$$

Da jede der Kanten in  $K_b$  mindestens Gewicht 1 hat ( $c(i, j) \in \{1, 2\}$  für alle  $i, j$ ) muss die Summe der Kosten der  $b$  Kanten der TSP-Tour mindestens  $b$  sein. Da die Tour nach Voraussetzung optimal ist, ist ihr Gesamtgewicht  $\leq b$ , d.h. also jede Kante der optimalen Tour in  $K_b$  muss tatsächlich in dem Graphen  $G$  vorkommen, denn würde sie nicht vorkommen, wäre ihr Gewicht nach Definition von  $c$  bereits 2 und damit das Gesamtgewicht bereits größer als  $b$ . Jede Tour ist ein Hamiltonkreis. Also ist  $G$  hamiltonsch.  $\square$

Da  $\{1, 2\}$ -TSP ein Spezialfall von TSP ist folgt direkt die NP-Härte vom allgemeinen TSP.

## 4 Approximierbarkeit

In diesem Abschnitt wollen wir untersuchen, inwieweit es möglich ist TSP mit dem Rechner zu lösen und anschliessend effiziente Algorithmen untersuchen, die TSP approximieren.

**Definition 4.1.** (analog zu [14]) Sei  $\Pi$  ein Optimierungsproblem (hier: ein Minimierungsproblem) und  $I$  eine Instanz von  $\Pi$ , dann bezeichnen wir mit  $\text{opt}(I)$  den optimalen (minimalen) Zielfunktionswert. Ein polynomieller  $\alpha$ -Approximationsalgorithmus ( $\alpha \in \mathbb{R}$ ,  $\alpha > 1$ ) für  $\Pi$  berechnet (in Polynomialzeit) für jede Instanz  $I$  eine zulässige Lösung, deren Wert höchstens  $\alpha \cdot \text{opt}(I)$  ist.  $\alpha$  heißt Approximationsgüte.

Analog zu [12] soll folgender Satz gezeigt werden:

**Satz 4.1.** Für das TSP in seiner allgemeinen Form existiert, unter der Annahme  $P \neq NP$ , für kein  $\alpha \in \mathbb{R}$ ,  $\alpha > 1$ , ein polynomieller  $\alpha$ -Approximationsalgorithmus.

**Beweis** Angenommen es existiere ein solcher  $\alpha$ -Approximationsalgorithmus  $A_\alpha$ . Sei  $G = (V, E)$  eine Instanz des Hamiltonkreisproblems (HC). Wir überführen  $G$  zu einer Instanz  $(K_n, c)$  von TSP wobei  $K_n$  der vollständige Graph auf  $n := |V|$  Knoten ist und

$$c(i, j) := \begin{cases} 1 & \text{falls } (i, j) \in E \\ \alpha \cdot n & \text{sonst} \end{cases}$$

Wir wenden nun unseren Algorithmus  $A_\alpha$  auf  $(K_n, c)$  an. Falls  $G$  eine Ja-Instanz ist, d.h. falls der Graph tatsächlich einen Hamiltonkreis enthält, so sind die Kosten der optimalen TSP-Tour genau  $n$  und damit die Kosten der Tour, die unser Algorithmus findet kleiner oder gleich  $\alpha \cdot n$ .

Ist  $G$  allerdings nicht hamiltonsch, so existiert in jeder TSP-Tour mindestens eine Kante, die in  $G$  **nicht** enthalten war, d.h. eine Kante mit Kosten  $\alpha \cdot n$ . Die Kosten der optimalen TSP-Tour betragen nun also mindestens  $(n-1) + \alpha \cdot n$ , damit sind auch die Kosten der Tour, die  $A_\alpha$  findet mindestens  $(n-1) + \alpha \cdot n > \alpha \cdot n$  ( $n \geq 2$ ).

D.h.  $G$  ist genau dann hamiltonsch, wenn die Kosten der Tour, die  $A_\alpha$  berechnet kleiner oder gleich  $\alpha \cdot n$  sind. Mit  $A_\alpha$  lässt sich also das Hamiltonkreisproblem entscheiden. Da  $A_\alpha$  in polynomieller Zeit berechenbar ist folgt  $HC \in P$ . Da HC ein NP-schweres Problem ist gilt nach Definition, dass sich jedes Problem in NP in Polynomialzeit auf HC reduzieren lässt, also  $P=NP$ , im Widerspruch zur Annahme, dass  $P \neq NP$ .

Ein solcher Algorithmus  $A_\alpha$  kann also (unter der Annahme  $P \neq NP$ ) nicht existieren. □

Analog kann man zeigen, dass für TSP in der allgemeinen Form nicht nur für konstantes  $\alpha$  kein Approximationsalgorithmus existieren kann sondern, dass auch für  $\alpha(n) = 2^n$  keiner existiert, siehe dazu [14]. Wir werden sehen, dass  $\Delta$ -TSP jedoch  $\frac{3}{2}$ -approximierbar ist. Für euklidisches  $\Delta$ -TSP existiert nach [2] sogar ein PTAS (*polynomial time approximation scheme*), d.h. für jedes  $\epsilon > 0$  existiert ein polynomieller  $(1 + \epsilon)$ -Approximationsalgorithmus, d.h. euklidisches  $\Delta$ -TSP lässt sich beliebig gut effizient approximieren (eine bessere Approximationsgüte hat hier allerdings eine praktisch höhere Laufzeit zur Folge, da der konstruierte Algorithmus eine Laufzeit von  $O(n^{\frac{1}{\epsilon}} \cdot \log n)$  hat).

Im folgenden soll zwischen zwei Typen von Algorithmen unterschieden werden. Die exakten Algorithmen, die exakte und wünschenswerterweise beweisbar optimale Lösungen liefern so wie

die Approximationsalgorithmen (sogenannte Heuristiken), die zwar im Normalfall keine optimale Lösung liefern, aber eine „gute“ Näherung, d.h. die gefundenen Lösungen weichen nur bis zu einem (vorher bereits bekannten) Grad von dem Optimum ab.

## 5 TSP exakt lösen

Alle bisher bekannten deterministischen exakten Algorithmen für das allgemeine TSP haben exponentielle Laufzeit (und unter der weitverbreiteten, aber unbewiesenen, Annahme  $P \neq NP$  existiert auch kein Polynomialzeitalgorithmus).

### 5.1 Brute Force

Die wohl naheliegendste Möglichkeit eine Instanz des allgemeinen TSP zu lösen ist wohl die *Brute Force* Methode. D.h. wir generieren einfach alle möglichen Lösungen und bestimmen dann die minimale. Das bestimmen eines Minimums geht offensichtlich in Polynomialzeit. Unglücklicherweise ist diese Methode völlig ineffizient, da, wie bereits in 1.1.3 erwähnt, die Anzahl der möglichen Lösungen proportional zu  $n!$  ist, genauer gesagt existieren für symmetrisches TSP  $\frac{1}{2}(n-1)!$  mögliche Lösungen.

Um sich dieses enorme Wachstum vorstellen zu können wollen wir ein Beispiel betrachten. Angenommen wir haben einen Computer, der pro Sekunde eine Quadrillionen (d.h.  $10^{24}$ ) mögliche Lösungen generieren kann. Wir wollen nun eine optimale Tour durch die 20 größten Städte Deutschlands finden. Der Computer würde dafür

$$\frac{\frac{1}{2}(20-1)!}{10^{24}} \approx 60.823 \text{ Nanosekunden (} 10^{-9} \text{ Sekunden)}$$

benötigen um alle möglichen Lösungen zu generieren. Der gleiche Computer würde für nur 10 Städte mehr, also für 30 Städte, bereits über 50 Tage benötigen und für insgesamt 50 Städte länger als  $9 \cdot 10^{30}$  **Jahre** (als Vergleich: unser Universum ist schätzungsweise  $1.37 \cdot 10^{10}$  Jahre alt). Für 60 Städte übertrifft die Anzahl der möglichen Lösungen bereits die geschätzte Anzahl der Protonen im Universum.

Wegen  $n! \leq n^n = 2^{n \cdot \lg(n)} \in O(2^{p(n)})$  ( $p$  Polynom) hat dieser naive Algorithmus also exponentiellen Zeitaufwand. (Es gilt  $TSP \in EXPTIME$ )

### 5.2 Dynamische Programmierung

Eine schnellere Möglichkeit eine TSP-Instanz exakt zu lösen ist mittels Rekursion, z.B. mit dem Prinzip der dynamischen Programmierung. Hier definiert man kleinere Teilprobleme, die man rekursiv löst und deren Zwischenwerte man in einer Tabelle speichert. Der Algorithmus betrachtet zunächst nur kleine Probleme und weitet seine Lösung dann immer weiter auf größere Teilmengen der Städte aus.

Die erzeugte Tabelle enthält  $n$  Zeilen und für jede Teilmenge eine Spalte, also maximal  $2^n$  Spalten, damit enthält die Tabelle maximal  $n \cdot 2^n$  Einträge. Mit Hilfe dieser Idee ist es möglich, TSP-Instanzen mit ungefähr  $n^2 \cdot 2^n$  Operationen zu lösen, was zwar immernoch exponentiell, aber schon deutlich besser als  $n!$  ist.

Die Zeitkomplexität ist zwar schon deutlich besser als bei der Brute Force Variante, es ist aber hier unbedingt zu beachten, dass für die Tabelle unter Umständen erhebliche Mengen von Speicher verfügbar sein müssen, da sie exponentiell viel Platz benötigt!

Für eine leichtverständliche Beschreibung des Algorithmus, siehe z.B. [10].

Wir haben also gesehen, dass für das allgemeine TSP (unter der Annahme  $P \neq NP$ ) kein effizienter Algorithmus existiert, aber auch kein effizienter Approximationsalgorithmus. Um polynomielle Approximationsalgorithmen zu bekommen müssen wir daher die Allgemeinheit des Problems aufgeben.

## 6 TSP approximieren

### 6.1 Nearest-Neighbour Heuristik

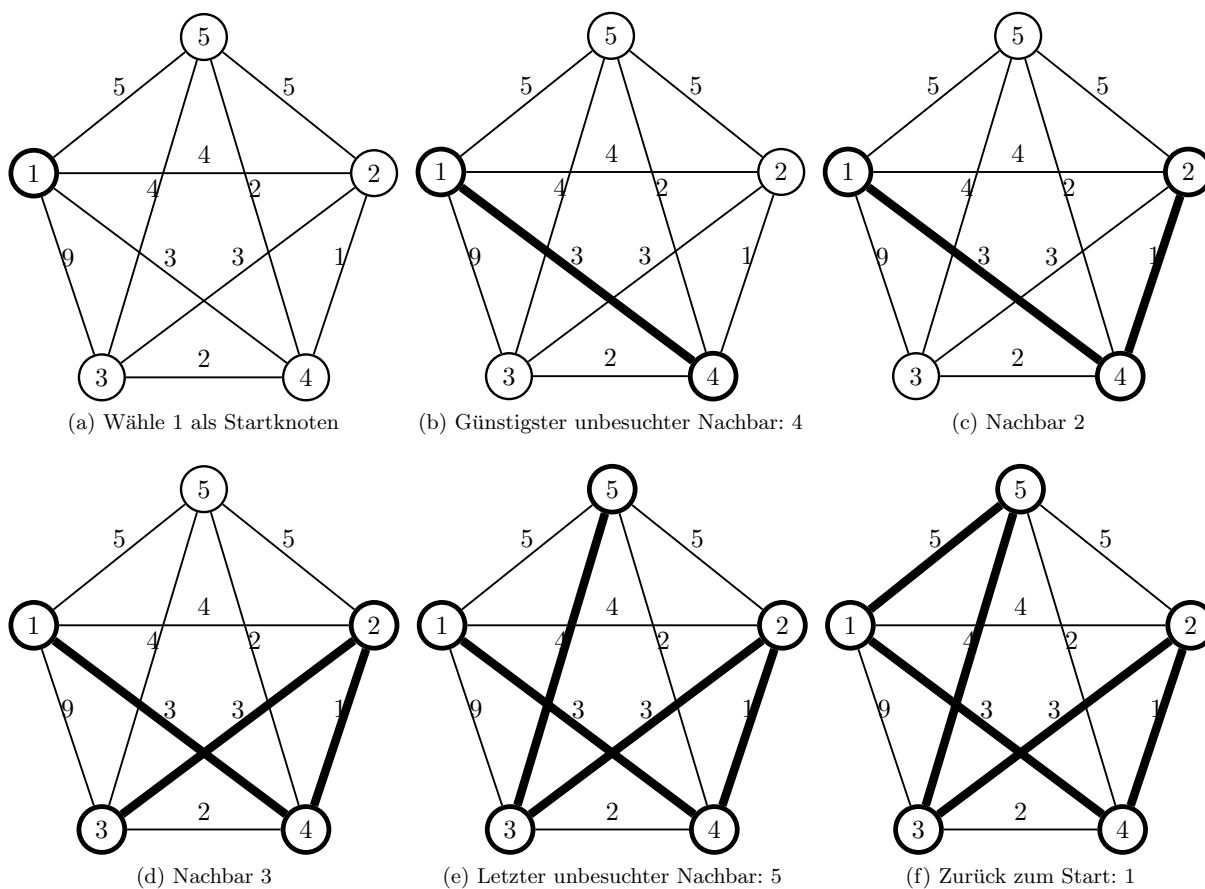
Eine sehr naive, aber auch sehr schnelle Heuristik ist die *Nearest-Neighbour* Heuristik. Es handelt sich dabei um einen Greedy-Algorithmus, d.h. einen Algorithmus, der mit dem Gedanken entworfen wurde, dass die konsequente Wahl von *lokal optimalen* Lösungen zu einem *globalen* Optimum führt.

#### Nearest-Neighbour Greedy Algorithmus

- (1) Markiere alle Knoten als unbesucht
- (2) Wähle beliebigen Startknoten als aktuellen Knoten
- (3) Wähle einen Knoten  $v$  aus, der mit minimalen Kosten vom aktuellen Knoten aus erreicht werden kann
- (4) Markiere  $v$  als besucht
- (5) Falls alle Knoten markiert sind, terminiere, falls nicht, gehe zu 3

Da der Graph endlich ist terminiert dieser Algorithmus offensichtlich auf jeden Fall. Die Reihenfolge, in der die Knoten markiert wurden ist nun eine gültige Lösung. Nach diesem Prinzip können oft sehr schnell gute TSP-Touren gefunden werden. Der Nachteil ist allerdings, dass diese Touren auch beliebig schlecht sein können. Nach [5] existiert sogar zu jeder Anzahl von Städten eine TSP Instanz (d.h. eine zugehörige Distanzfunktion), für die der Nearest-Neighbour Algorithmus die **schlechtest mögliche** TSP-Tour generiert.

### 6.1.1 Beispiel



Die Nearest-Neighbour Heuristik wählt hier also die Tour  $\pi := (142351)$  mit

$$\text{cost}(\pi) = 3 + 1 + 3 + 4 + 5 = 16$$

Vergleicht man das mit allen möglichen 12 Touren

$$(123451), (123541), (124351), (124531), (125341), (125431)$$

$$(132451), (132541), (134251), (135241), (142351), (143251)$$

so sieht man, dass hier tatsächlich

$$\text{cost}(\pi) = \min\{16, 16, 16, 20, 18, 22, 20, 22, 22, 22, 16, 18\}$$

gilt

## 6.2 MST-Heuristik

Es soll nun gezeigt werden, dass sich der Spezialfall des metrischen TSP ( $\Delta$ -TSP) mit einer Approximationsgüte von 2 approximieren lässt, d.h. es existiert ein effizienter Algorithmus, dessen Ergebnis höchstens doppelt so teuer ist wie die optimale Lösung. Der Algorithmus wird als *Metric-TSP-via-MST* oder kurz als MST-Heuristik bezeichnet.

**Definition 6.1.** (analog zu [13]) Sei  $G$  ein zusammenhängender und nicht trivialer Graph. Existiert in  $G$  ein geschlossener Kantenzug  $Z$ , der alle Kanten des Graphen enthält, so heißt  $Z$  Eulertour und  $G$  heißt eulerscher Graph.

**Satz 6.1.** Ein Graph ist genau dann eulersch, wenn der Grad jedes Knoten gerade ist (Beweis siehe [13]).

Mit Hilfe dieser Eigenschaft lässt sich mittels Tiefensuche effizient testen ob ein Graph eulersch ist. Diese Frage nennt man das *Eulerkreisproblem* (dieses Problem ist also in der Komplexitätsklasse P). In gegebenen eulerschen Graphen einen Eulerkreis zu bestimmen ist ebenfalls effizient möglich (z.B. mit dem *Algorithmus von Hierholzer*).

### Algorithmus: MST-Heuristik

- (1) Erzeuge einen minimalen aufspannenden Baum (MST)  $T$  von  $G$
- (2) Verdopple jede Kante in  $T$
- (3) Wähle einen beliebigen Startknoten  $v$  in  $T$
- (4) Bestimme einen Eulerkreis  $K$  in  $T$ , beginnend bei  $v$
- (5) Entferne in dem Eulerkreis alle Kanten, die zu bereits besuchten Knoten führen und ersetze sie durch eine Kante zum Knoten der im Eulerkreis an nächster Stelle kommt (es sei denn es handelt sich um den Startknoten). Der neue Graph heiße  $K'$

(1) geht z.B. mit dem Algorithmus von Prim oder dem Algorithmus von Kruskall in Polynomialzeit. Durch Verdopplung aller Kanten wird offensichtlich der Grad jedes Knoten verdoppelt, d.h. der Grad jedes Knotens ist nun gerade und der Graph somit eulersch, also existiert ein Eulerkreis. (4) ist, wie oben bereits erwähnt, ebenfalls in Polynomialzeit durchführbar.

**Satz 6.2.** Die MST-Heuristik ist ein polynomieller 2-Approximationsalgorithmus für  $\Delta$ -TSP.

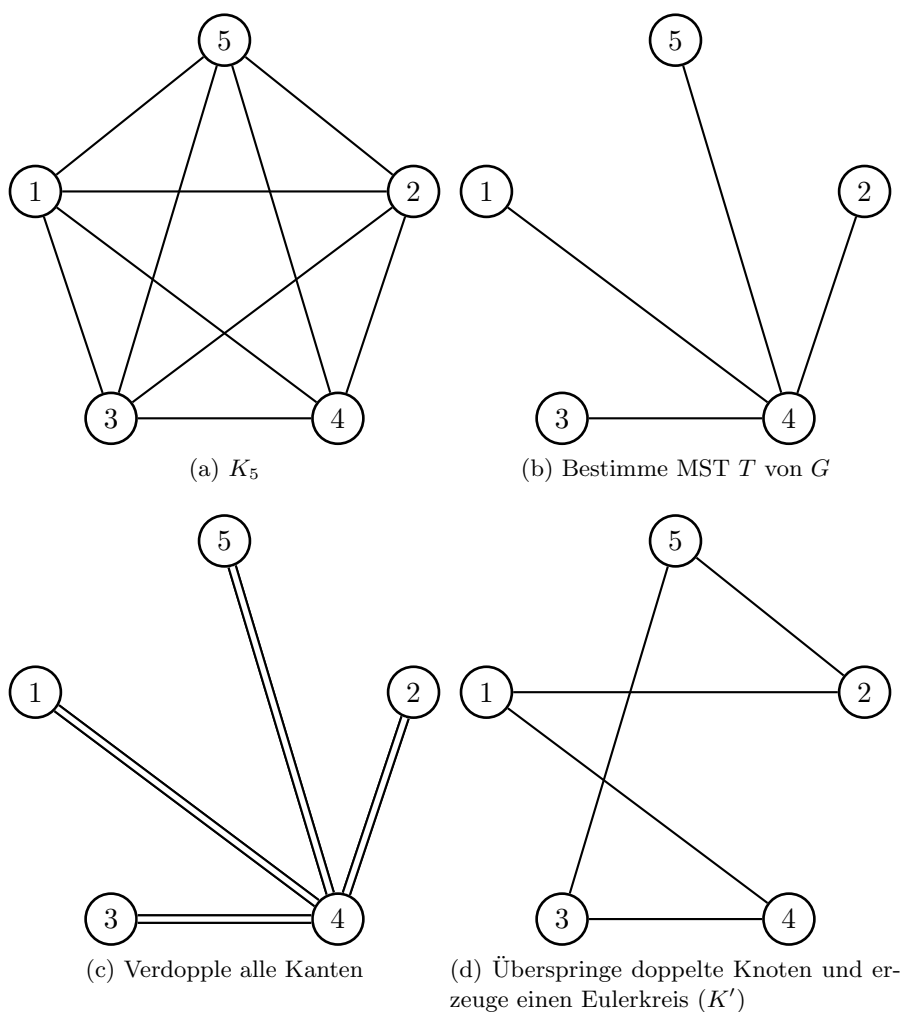
**Beweis**  $K'$  ist nun eine mögliche TSP-Tour, da jeder Knoten besucht wird und  $K'$  somit ein Hamiltonkreis ist. Die Kosten jeder optimalen TSP-Tour sind niemals geringer als die Kosten eines beliebigen MST, d.h. es gilt  $cost(T) \leq opt(G)$ .

Der Eulerkreis  $K$  enthält (vor dem entfernen) jede Kante doppelt, d.h.  $cost(K) = 2 \cdot cost(T)$ . Da bei metrischem TSP die Distanzfunktion immer die Dreiecksungleichung erfüllt, kann das überspringen von Knoten die Kosten nicht erhöhen, d.h.

$$cost(K') \leq cost(K) = 2 \cdot cost(T) \leq 2 \cdot opt(G)$$

□

### 6.2.1 Beispiel



### 6.3 Algorithmus von Cristofides

Als nächstes wollen wir einen Algorithmus betrachten, der 1971 von Nico Christofides vorgestellt wurde, und der eine ähnliche Herangehensweise wie die MST-Heuristik verfolgt.

**Definition 6.2.** (analog zu [11]) Eine Kantenmenge  $M \subseteq E$  heißt Matching in einem Graphen  $G = (V, E)$ , falls kein Knoten des Graphen zu mehr als einer Kante aus  $M$  inzident ist, oder formal ausgedrückt, wenn  $e \cap f = \emptyset$  für alle  $e, f \in M$  mit  $e \neq f$ . Man sagt ein Knoten  $v$  wird von  $M$  überdeckt, falls es keine Kante  $e \in M$  gibt, die  $v$  enthält. Ein Matching heißt perfekt, wenn jeder Knoten durch genau eine Kante aus  $M$  überdeckt wird, oder anders ausgedrückt, wenn  $|M| = \frac{1}{2} \cdot |V|$ .

**Algorithmus von Christofides** (siehe [8])

- (1) Bestimme einen minimalen aufspannenden Baum  $T$  von  $G$
- (2) Bestimme die Menge  $V' = \{v \in V | v \text{ hat ungeraden Grad in } T\} \subseteq V$

- (3) Finde ein perfektes Matching mit minimalen Kosten  $M$  auf  $V'$
- (4) Sei  $H$  die Vereinigung von  $T$  und  $M$
- (5) Finde eine Euler-Tour auf den Kanten von  $H$
- (6) Bereinige die Euler-Tour um wiederholt vorkommende Knoten

(1) und (2) geht effizient, ebenso (5), ausserdem erhöht (6) die Kosten nicht (siehe MST-Heuristik). Der „teuerste“ Schritt dieses Algorithmus ist das finden eines günstigsten perfekten Matchings, was in  $O(n^3)$  möglich ist (z.B. mit dem *Hungarian Algorithm*).

Die Idee ist im Grunde, alle Knoten mit ungeradem Grad auszuwählen und für diese ein perfektes Matching zu finden. Zusammen mit dem minimalen Spannbaum (der alle Knoten enthält, also auch die, die bereits geraden Grad hatten) enthält der neue Graph  $H$  nun alle Knoten und jeder Knoten hat geraden Grad, somit ist  $H$  eulersch und (5) ist durchführbar.

Ein perfektes Matching in dem von  $V'$  induzierten Teilgraphen existiert auf jedenfall, weil er ebenfalls vollständig ist und weil  $|V'|$  gerade ist (Beweis siehe [8]).

**Satz 6.3.** *Der Christofides Algorithmus ist ein polynomieller  $\frac{3}{2}$ -Approximationsalgorithmus für metrisches TSP.*

**Beweis** Die gefundene Euler Tour in  $H$  benutzt alle Kanten und besucht damit auch alle Knoten (weil  $T$  aufspannender Baum ist), ist also eine mögliche TSP-Tour. Wie bei der MST-Heuristik schon erwähnt, gilt  $cost(T) \leq opt$ .

Wieder analog zu [8]: Sei  $\tau$  eine optimale TSP-Tour. Aus  $\tau$  erhalten wir einen Kreis  $\tau'$ , der die Knoten aus  $V'$  verbindet, indem wir alle Knoten in  $\tau$ , die nicht in  $V'$  enthalten sind, überspringen. Aufgrund der Dreiecksungleichung ist das überspringen günstiger als der „Umweg“ (oder genauso teuer), d.h.  $cost(\tau') \leq cost(\tau)$ . Wir können  $\tau'$  zerlegen in zwei perfekte Matchings, die jeweils aus jeder zweiten Kante auf dem Kreis  $\tau'$  gebildet werden. Das günstigere der beiden Matchings hat höchstens die Kosten  $\frac{1}{2} \cdot cost(\tau') \leq \frac{1}{2} \cdot cost(\tau) = \frac{1}{2} \cdot opt$ . Also hat das günstigste perfekte Matching auf  $V'$  höchstens die Kosten  $\frac{1}{2} \cdot opt$ .

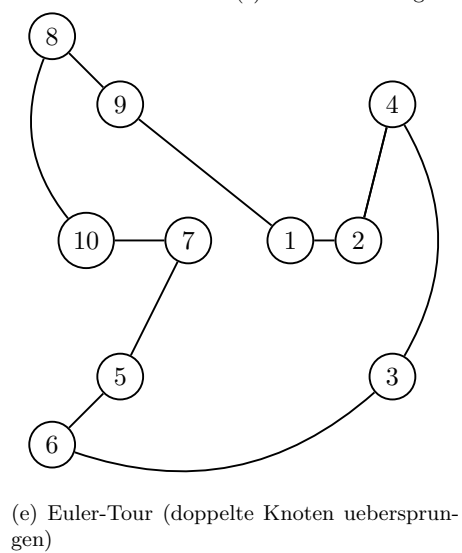
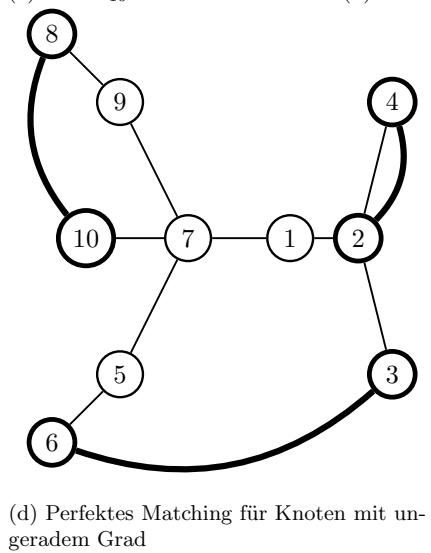
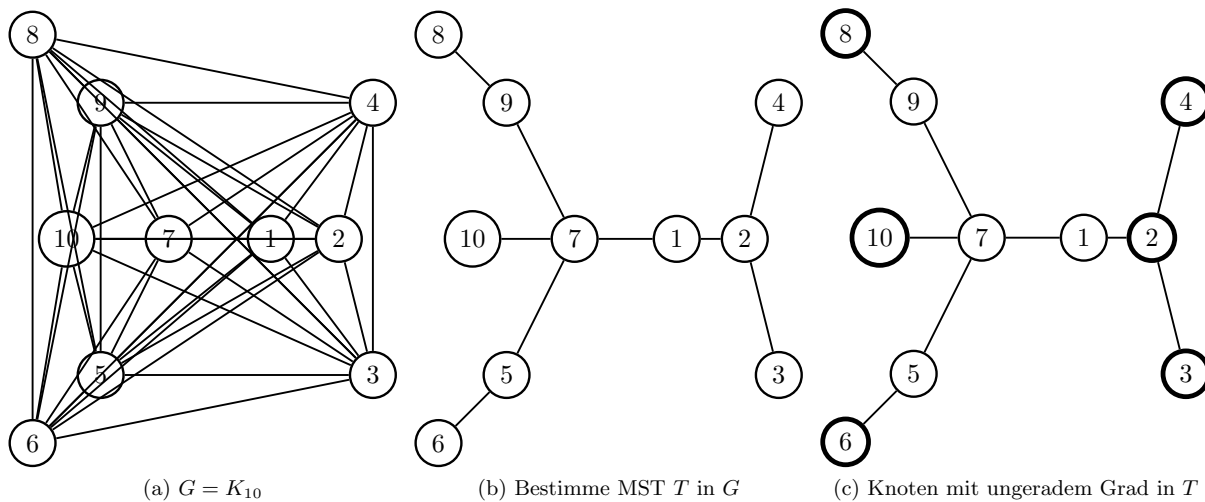
Die Euler-Tour hat die Kosten:

$$cost(H) \leq cost(M) + cost(T) \leq opt + \frac{1}{2} \cdot opt = \frac{3}{2} \cdot opt$$

□

Bisher hat niemand einen polynomiellen Approximationsalgorithmus für metrisches TSP mit einer besseren Approximationsgüte gefunden oder bewiesen, dass es keinen solchen geben kann. Man weiss lediglich, dass kein  $\alpha$  Approximationsalgorithmus für  $\Delta$ -TSP mit  $\alpha < \frac{220}{219}$  existieren kann (es sei denn  $P=NP$ ).

## 6.3.1 Beispiel



## Literatur

- [1] Christel Baier Alexander Asteroth. *Theoretische Informatik*. Pearson Studium, 2003.
- [2] Sanjeev Arora. Polynomial time approximation schemes for Euclidean traveling salesman and other geometric problems. *Journal of the ACM*, 45(5):753–782, 1998.
- [3] Daniel J. Rosenkrantz, Richard E. Stearns, Philip M. Lewis II. An Analysis of Several Heuristics for the Traveling Salesman Problem. *SIAM Journal on Computing*, pages 563–581, 1977.
- [4] David S. Johnson, Lyle A. McGeoch. The Traveling Salesman Problem: A Case Study in Local Optimization. *Local Search in Combinatorial Optimization*, pages 215–310, 1997.
- [5] G. Gutin, A. Yeo, A. Zverovich. Traveling salesman should not be greedy: domination analysis of greedy-type heuristics for the TSP. *Discrete Applied Mathematics 117*, pages 81–86, 2002.
- [6] Prof. Dr. Erich Grädel. *Komplexitätstheorie*. RWTH Aachen University, Lehr und Forschungsgebiet Mathematische Grundlagen der Informatik, 2006.
- [7] Prof. Dr.-Ing. H. Ney. *Algorithmen und Datenstrukturen*. RWTH Aachen University, Lehrstuhl für Informatik 6, 2006.
- [8] Prof. Dr. Berthold Vöcking. Einführung in Approximationsalgorithmen: Optimierungsprobleme auf Graphen und Metriken. *Effiziente Algorithmen*, pages 7–11, 2007.
- [9] Richard M. Karp. Reducibility Among Combinatorial Problems. In *Complexity of Computer Computations*, pages 85–103. 1972.
- [10] Stefan Näher, Universität Trier. Das Traveling Salesman Problem (oder die optimale Tour für den Nikolaus). *40. Algorithmus der Woche*, 2006.
- [11] Prof. Dr. Angelika Steger. *Diskrete Strukturen 1: Kombinatorik, Graphentheorie, Algebra*. Springer, 2002.
- [12] Thomas Emden-Weinert, Stefan Hougardy, Bernd Kreuter, Hans Jürgen Prömel, Angelika Steger. Approximationsalgorithmen. *Einführung in Graphen und Algorithmen*, pages 287–289, 1996.
- [13] Prof. Dr. Lutz Volkmann. *Graphen an allen Ecken und Kanten*. RWTH Aachen University, Lehrstuhl II für Mathematik, 2006.
- [14] Prof. Dr. Berthold Vöcking. *Berechenbarkeit und Komplexität*. RWTH Aachen University, Lehrstuhl für Informatik 1, 2006.