4
Schnelle Rundreisen: Das Travelling-Salesman-Problem

Martin Grötschel

1 Städtereisen

Dieses Kapitel behandelt das bekannteste aller kombinatorischen Optimierungsprobleme, bei dessen Namensnennung man sich sofort vorstellen kann, worum es geht. Und jeder hat auch sogleich „gute“ Ideen, wie man dieses Problem „lösen“ kann. Dies ist nur einer von vielen Gründen für die besondere Eignung dieses Problems für den Mathematikunterricht. Wir sprechen hier vom Problem des Handlungsreisenden oder dem *Travelling-Salesman-Problem* (kurz: *TSP*).

Das Travelling-Salesman-Problem ist das am intensivsten untersuchte kombinatorische Optimierungsproblem. In diesem Kapitel wird eine Einführung in das TSP gegeben. Es werden Problemstellungen erläutert, Anwendungen skizziert und einige Schwierigkeiten bei der korrekten Modellierung der Zielfunktion dargelegt. Es ist gar nicht so klar, was in einem konkreten Problem die wirkliche Entfernung ist. Exakte und approximative Lösungsverfahren werden an Beispielen skizziert, und es wird angedeutet, dass man, obwohl TSPs zu den theoretisch schweren Problemen zählen, in der Praxis TSPs von atemberaubender Größe optimal lösen kann.

Die Aufgabe ist einfach formuliert. Wir betrachten eine Anzahl Städte, sagen wir n Stück. Wir wollen in einer Stadt starten, alle anderen genau einmal besuchen und am Ende zum Startort zurückkehren. Diese Rundreise (kurz: *Tour*) wollen wir so schnell wie möglich zurücklegen.

Das Travelling-Salesman-Problem
Finde eine kürzeste Rundreise durch n Städte. Jede Stadt soll dabei genau einmal besucht werden.

Vom Travelling-Salesman-Problem geht eine große Faszination aus, was neben dem Nutzen bei praktischen Anwendungen ein Grund für seine Berühmtheit ist. Worin der Reiz liegt, finden Sie am besten heraus, indem Sie selbst probieren, kür-

zeste Rundreisen zu finden.

- Wählen Sie sich einige Städte aus und versuchen Sie, eine kürzeste Tour durch alle diese Städte (ohne eine davon doppelt zu durchfahren) zu finden. Wie gehen Sie dabei vor? Versuchen Sie dies auf unterschiedliche Weisen.
- Wie viele verschiedene Rundreisen gibt es bei $3, 4, 5, \ldots, n$ Städten?
- In welchen anderen Zusammenhängen tauchen Rundreiseprobleme auf?

Legen Sie Ihren Überlegungen selbst gewählte Beispiele zugrunde oder verwenden Sie die folgende Entfernungstabelle für das Rheinlandproblem:

	Aachen	Bonn	Düsseldorf	Frankfurt	Köln	Wuppertal
Aachen	—	91	80	259	70	121
Bonn	91	—	77	175	27	84
Düsseldorf	80	77	—	232	47	29
Frankfurt	259	175	232	—	189	236
Köln	70	27	47	189	—	55
Wuppertal	121	84	29	236	55	—

Die Länge einer optimalen Tour für das Rheinlandproblem finden Sie am Ende dieses Kapitels. Die folgenden Ausführungen bearbeiten ein etwas größeres Beispiel. Für Ihre eigenen Überlegungen kann es aber immer wieder hilfreich sein, auch ein einigermaßen kleines Beispiel wie das Rheinlandproblem zur Verfügung zu haben, für das das Optimum bekannt ist.

Die Modellierung als Graph

Um eine kürzeste Tour durch eine Anzahl von Städten zu finden, muss man natürlich die Entfernungen zwischen je zwei Städten kennen. Man kann hier an Längen (in km, m oder mm) denken oder auch an Fahrzeiten (in Std, Min, Sekunden oder Millisekunden). Welches „Entfernungsmaß“ gewählt wird, hängt von den eigenen Wünschen oder den Vorgaben des „Auftraggebers“ ab. Wir werden ab jetzt die Städte immer mit $1, 2, \ldots, n$ und die Entfernung von Stadt i zur Stadt j mit $c(i, j)$ bezeichnen. Wir geben dazu keine Einheit an, denn mathematisch betrachtet ist es egal, ob Weglängen oder Fahrzeiten zu minimieren sind.

Das TSP hat verschiedene Varianten. Wir betrachten in diesem Artikel nur zwei. Sind Hin- und Rückweg zwischen zwei Städten stets gleich lang, gilt also $c(i, j) = c(j, i)$ für alle Städtepaare i und j, so nennen wir das TSP *symmetrisch*, kurz *STSP*, andernfalls heißt es *asymmetrisch*, kurz *ATSP*.

- In welchen Situationen sind die „Entfernung“ von Stadt A nach Stadt B und die „Entfernung“ von B nach A unterschiedlich? Suchen Sie Beispiele für solche asymmetrische TSPs! Denken Sie dabei auch an andere „Entfernungen“ als Weglängen.

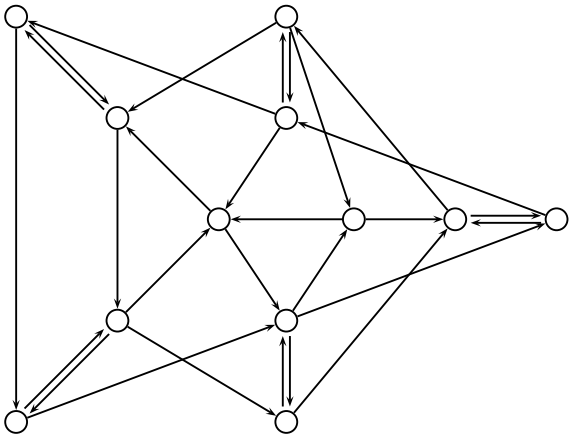


Abbildung 1. Ist dieser gerichtete Graph hamiltonsch?

Wir werden sogleich sehen, dass die Anwendungen des TSP weit über das Bereisen von Städten hinausgehen. Wir wollen daher beim TSP nicht mehr von Städten sprechen, sondern die graphentheoretische Sprache benutzen. Die Städte werden dann als *Knoten* eines gerichteten oder ungerichteten Graphen aufgefasst, und *Bögen* oder *Kanten* sind in dem Graphen dann vorhanden, wenn es eine Verbindung zwischen den beiden Endknoten gibt. Gesucht wird eine (Rund-)Tour, die durch alle Knoten genau einmal führt. Geht man von einem gerichteten Graphen aus, so wird ein gerichteter Kreis gesucht, also ein Kreis, der entlang den Richtungen der Bögen durchfahren werden kann.

Definitionen

Ein geschlossener Kantenzug, der durch jeden Knoten eines zusammenhängenden Graphen genau einmal führt, heißt *Hamiltonkreis*.

Ein Graph, der einen Hamiltonkreis enthält, heißt *hamiltonscher Graph*.

In gerichteten Graphen gelten diese Definitionen analog, dann natürlich für gerichtete Kreise. Der Name geht auf den irischen Mathematiker und Physiker Sir William Rowan Hamilton (1805–1865) zurück. Er erfand 1856 das „Icosian Game“, bei dem man auf einem Graphen, der die Ecken und Kanten eines Dodekaeders darstellt, einen Hamiltonkreis konstruieren soll.

Beim TSP geht man in der Regel davon aus, dass der Graph vollständig ist, man also von jedem Knoten auf direktem Weg zu jedem anderen Knoten gelangen kann. Falls das in einem Anwendungsfall nicht so ist, wird die entsprechende Kante mit einem sehr hohen Gewicht oder Wert belegt. Falls in einer Optimallösung eine solche Kante auftritt, bedeutet das, dass es im vorliegenden Problem gar keine Rundreise gibt. Gleiches gilt für ATSPs. Hier sogleich ein derartiges Beispiel.

Abbildung 1 zeigt einen gerichteten Graphen mit 12 Knoten und 28 Bögen.

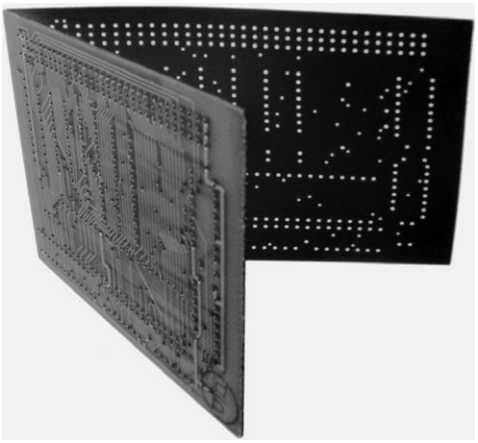


Abbildung 2. Eine noch unbestückte Leiterplatte

Wir weisen jedem Bogen dieses Graphen die Länge 1 zu. Alle nicht gezeichneten Bögen, die den Graphen zu einem vollständigen Graphen ergänzen, erhalten die Länge $M > 1$, also irgendeinen Wert, der größer als 1 ist. Auf diese Weise haben wir ein ATSP definiert. Mit Hilfe dieses ATSPs kann man herausfinden, ob der ursprüngliche Graph einen Hamiltonkreis enthält: In dem gerichteten Graphen aus Abbildung 1 gibt es genau dann einen gerichteten Kreis, der alle Knoten enthält, also einen hamiltonschen Kreis, wenn das so definierte ATSP eine Tour mit Länge 12 hat. Enthält der Graph keinen gerichteten hamiltonschen Kreis, so verwendet jede kürzeste Tour mindestens einen Bogen der Länge M und ist somit länger als 12. Die Länge einer Optimallösung des ATSP zeigt also, ob man in dem ursprünglichen Graphen einen Hamiltonkreis finden kann. Wenn ja, so ist diese optimale Tour der gesuchte Hamiltonkreis.

- Enthält der gerichtete Graph aus Abbildung 1 einen gerichteten hamiltonschen Kreis?

2 Das Bohren von Leiterplatten

Bevor wir uns dem Ursprung des Problems widmen, sehen wir uns eine aktuelle Anwendung des TSP etwas genauer an.

Abbildung 2 zeigt eine Leiterplatte (und ihren Schatten auf einem Blatt Papier). Sie ist noch „unbestückt“. Auf ihr sind Leiterbahnen und Löcher zu sehen. Die Leiterbahnen sind „gedruckte Kabel“, durch die nach der Fertigstellung Strom fließen wird. Man nennt Leiterplatten auch Platinen oder gedruckte Schaltungen (engl. printed circuit board, kurz PCB). Die Löcher werden später einmal die „Füße“ von elektronischen Bauteilen aufnehmen oder Leiterbahnen auf der Oberseite

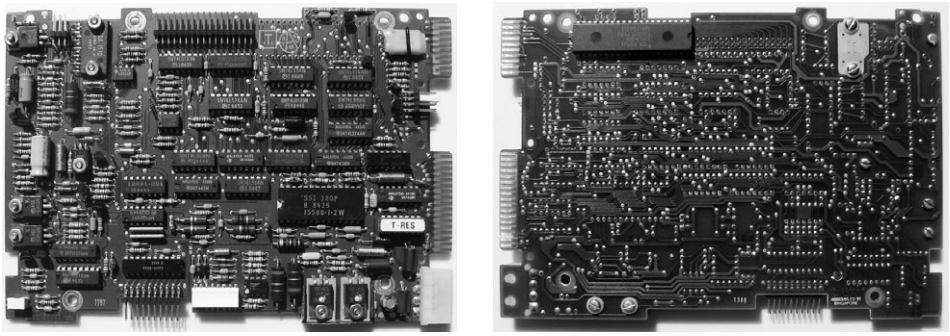


Abbildung 3. Vorder- und Rückseite einer fertig bestückten Leiterplatte

mit Leiterbahnen auf der Unterseite verbinden. Für Elektroniker ist eine Leiterplatte ein Schaltungsträger für elektronische Baugruppen. In Abbildung 3 sind die zwei Seiten einer fertig bestückten und gelöteten Leiterplatte zu sehen. Solche Objekte werden auch *Flachbaugruppen* genannt. Die Löcher sind jetzt mit Lötzinn gefüllt und nur noch als helle Punkte erkennbar. Leiterplatten sind aus unserem täglichen Leben nicht wegzudenken. Fast jedes elektronische Gerät enthält mindestens eine Leiterplatte.

Wir wollen uns hier mit einem kleinen Schritt im Produktionsprozess von Leiterplatten beschäftigen: dem Bohren der Löcher. In der Elektronik wird ein Loch in einer Leiterplatte auch Durchkontaktierung oder (ganz kurz) *Via* genannt. Große Leiterplatten können mehrere Tausend und sogar noch mehr Löcher enthalten. Diese werden in der Regel mit mechanischen Bohrmaschinen, häufig auch mit Laserbohrern hergestellt. Das geht meistens wie folgt. Das Bohrgerät befindet sich in einer „Nullposition“, z. B. dort, wo es den passenden Bohrer aus einem Magazin aufnimmt. Die Leiterplatte ist auf einem „Tisch“ befestigt. Nun fährt der Bohrer zu einer Stelle, an der ein Loch gebohrt werden muss, bohrt und fährt dann zu einer weiteren Bohrposition. Er macht das so lange, bis alle Löcher, die mit dem ausgewählten Bohrer hergestellt werden müssen, gebohrt sind. Dann fährt der Bohrkopf zur Nullposition zurück, tauscht den Bohrer aus und macht so lange weiter, bis alle Löcher gebohrt sind. Danach wird die Leiterplatte manuell oder durch eine Hebevorrichtung vom Tisch genommen, und es wird eine neue fixiert.

Es gibt auch Bohrgeräte, bei denen der Bohrkopf fest steht und der Tisch mit der Leiterplatte bewegt wird. Manche Bohrmaschinen können mehrere Leiterplatten, die übereinander gestapelt sind, in einem Schritt (mit langen Bohrern) verarbeiten, und es gibt sogar Geräte, bei denen identische Leiterplatten gleichzeitig in parallelen Stapeln durchbohrt werden. Wir wollen der Einfachheit halber jetzt annehmen, dass wir eine einzige Leiterplatte bearbeiten und der Bohrkopf bewegt wird.

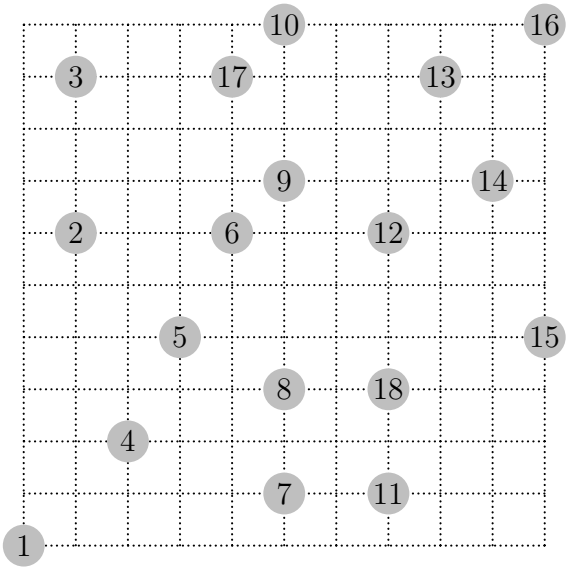


Abbildung 4. Ein kleines Bohrproblem mit 18 Löchern

- Was kann beim Bohren der Löcher in eine Leiterplatte optimiert werden?

Bei der Bohrzeit ist (außer man wählt eine andere Maschine) nichts einzusparen, alle Löcher müssen ja gebohrt werden. Eine Beschleunigung der Herstellung kann nur durch Verringerung der *Leerfahrtzeit* erreicht werden, das ist die Zeit, die das Gerät durch Bewegung des Bohrkopfes „vergeudet“. Das Ziel muss also sein, die Summe aller Fahrzeiten, die der Bohrkopf beim „Wandern von Loch zu Loch“ verbringt, so gering wie möglich zu machen. Bezeichnen wir die Nullstellung und jede Bohrposition als „Stadt“ und definieren wir als Entfernung zwischen Stadt i und Stadt j die Zeit, die die Bohrmaschine braucht, um sich von i nach j zu bewegen, so haben wir ein TSP vor uns. Genauer, gibt es k verschiedene Lochgrößen, so sind zur Minimierung der Bearbeitungszeit k TSPs zu lösen, für jeden Bohrertyp eines. Das Problem ist symmetrisch, da der Weg von i nach j genauso viel Zeit benötigt wie umgekehrt.

Abbildung 4 zeigt ein kleines Bohrproblem auf einem 11×11-Gitter, das übersichtlich genug für unsere nächsten Überlegungen ist. Der Knoten 1 repräsentiert den Punkt $(x_1, y_1) = (0, 0)$ und ist die Startposition. Hier fährt der Bohrer los. Zwei benachbarte Gitterlinien haben den Abstand 1. Alle 18 eingezeichneten Löcher haben die gleiche Größe und müssen in einer Tour besucht werden, die zum Startpunkt (0,0) zurückführt.

- Bestimmen Sie die kürzeste Tour durch alle 18 Knoten aus Abbildung 4 durch Ausprobieren oder mit Hilfe Ihrer eingangs selbst entwickelten Verfahren.

Viele technische „Verfahrprobleme“ lassen sich auf TSPs zurückführen. Bei der

Herstellung von Flachbaugruppen etwa werden in einem weiteren Schritt des Produktionsprozesses Bauteile mit Hilfe eines Bestückungsautomaten auf die Leiterplatte aufgesetzt. Wenn zum Beispiel bei einem solchen Automaten alle Bauteile eines Typs hintereinander dem Bestückungskopf zugeführt werden, so ist für jeden Bauteiltyp ein TSP zu lösen.

Schweißroboter werden u. a. in der Automobilindustrie in großer Zahl eingesetzt. Sie müssen z. B. verschiedene Karosserieteile durch Punktschweißen aneinanderfügen. Die Planung der Bearbeitungsreihenfolge führt dabei auf ein TSP. Hierbei ist die Berechnung der Entfernung von zwei Schweißpunkten aufwändig, weil der Roboter unter Umständen komplizierte Manöver ausführen muss, um von einem Punkt zum anderen zu gelangen: Seine Bewegungsmöglichkeiten sind beschränkt und verschiedene Karossierieteile können die Fahrt auf dem zeitlich kürzesten Weg verhindern. Hier können sowohl symmetrische als auch asymmetrische TSPs auftreten.

Asymmetrische TSPs kommen u. a. bei Säuberungs- oder Umrüstaktivitäten vor. In Walzwerken beispielsweise werden die verschiedenen Stahlprofile des Produktionsprogramms in der Regel zyklisch gefertigt. Bei jedem Profilwechsel ist eine zeitaufwändige Umrüstung erforderlich, hier jedoch ist die Umrüstzeit aufgrund technischer Bedingungen häufig asymmetrisch. Das Umrüsten von Profil A zu Profil B kann erheblich länger dauern als umgekehrt. Ähnlich ist es bei der Reihenfolgeplanung von Mehrproduktenpipelines oder Lackierkammern. In der Fahrzeugindustrie wird häufig in zyklischer Reihenfolge lackiert. Wenn weiß auf schwarz folgt, muss sorgfältiger gereinigt werden als umgekehrt.

Die Knoten eines TSP können also ganz unterschiedliche reale Bedeutung haben: Städte, Bohrlöcher, Stahlprofile, Flüssigkeiten in einer Pipeline oder Farben in einer
Lackiererei. Die Entfernungen zwischen Knoten entsprechen meistens dem Zeitaufwand, den man in der realen Anwendung benötigt, um von einem Knoten zum anderen zu gelangen.

Eine Vielzahl weiterer TSP-Anwendungen findet man auf der Webseite http://www.tsp.gatech.edu/apps/index.html und in Kapitel 2 von [15] und in Kapitel 1 von [12].

3 Löcher bohren: Die Zielfunktion

Sie haben nun schon viel über optimale Rundreisen nachgedacht. Dass je nach Anwendung unterschiedliche Größen optimiert werden, liegt auf der Hand. Dass aber bei einem konkreten Optimierungsproblem wie dem Bohren von Löchern noch ein Entscheidungsspielraum besteht, wie der Abstand zwischen zwei Bohlöchern definiert werden soll, ist vielleicht nicht auf den ersten Blick klar.

In der Optimierung bezeichnet man die Funktion, deren Wert (unter gewissen Nebenbedingungen) so klein oder so groß wie möglich gemacht werden soll, als *Zielfunktion*. Beim TSP ist die Zielfunktion die Summe der Entfernungen der Kanten oder Bögen einer Tour. In Abbildung 4 ist ein Bohrproblem graphisch dargestellt.

- Wie lässt sich der zeitliche Abstand zwischen zwei Bohrlöchern bestimmen? Überlegen Sie sich dafür, wie der Bohrkopf angetrieben wird.

Falls Sie sich an die Lösung des durch Abbildung 4 skizzierten TSP gemacht haben, haben Sie vermutlich als Entfernung an den „gewöhnlichen", so genannten *euklidischen Abstand* oder ℓ_2*-Abstand* zwischen zwei Punkten gedacht. Dieser mit $c_2(i,j)$ bezeichnete Abstand zwischen zwei Punkten (x_i, y_i), (x_j, y_j) ist folgendermaßen definiert:

Definition
Der *euklidische Abstand* zweier Punkte (x_i, y_i), (x_j, y_j):

$$c_2(i,j) := \sqrt{|x_i - x_j|^2 + |y_i - y_j|^2}$$

Wenn man diesen Abstand mit einem die Geschwindigkeit des Bohrkopfs widerspiegelnden Faktor multipliziert, dann ist dies wahrscheinlich eine gute Schätzung der Fahrzeit, so die Vermutung.

Das ist bei Bohrproblemen aber nicht unbedingt so. Der Bohrkopf wird von zwei Motoren angetrieben, einer bewegt sich in x-Achse, der andere in y-Achse, nennen wir sie X- und Y-Motor. Soll die Maschine von (x_i, y_i) nach (x_j, y_j) fahren, so bewegt sich der X-Motor von x_i nach x_j und der Y-Motor von y_i nach y_j. Derjenige Motor, der am längsten braucht, bestimmt die zeitliche Entfernung zwischen den zwei Punkten. Die Bewegung ist jedoch nicht gleichförmig. Sie beginnt mit einer Beschleunigungsphase, dann fährt der Motor mit Maximalgeschwindigkeit, danach kommt die Bremsphase, zum Schluss muss der Bohrkopf noch ausgerichtet werden. Bei kurzen Strecken kommt es gar nicht zur Schnellfahrt, weil während der Beschleunigungsphase schon die Bremsphase eingeleitet werden muss. Ferner sind der X-Motor und der Y-Motor häufig nicht gleich schnell. Das kann an technischen Gegebenheiten oder unterschiedlicher Abnutzung liegen. Die „wahren" zeitlichen Entfernungen sind also durchaus schwierig zu ermitteln, selbst dann, wenn man alle technischen Parameter kennt.

In der Praxis behilft man sich häufig damit, dass man die Enfernung durch eine Näherungsformel schätzt. Ein üblicher Ansatz zur Abschätzung der Fahrzeit des Bohrkopfes besteht darin, das Maximum der Werte $|x_i - x_j|$ und $|y_i - y_j|$ zur Entfernung zwischen den Punkten (x_i, y_i) und (x_j, y_j) zu erklären. In der Mathematik wird dieser Abstand ℓ_∞*-Abstand* oder $\max$*-Abstand* genannt.

Definition
Der max-*Abstand* zweier Punkte (x_i, y_i), (x_j, y_j):

$$c_{\max}(i,j) := \max\{|x_i - x_j|, |y_i - y_j|\}$$

Manchmal ist einer der beiden Motoren langsamer als der andere, sagen wir, der X-Motor braucht doppelt so viel Zeit pro Längeneinheit wie der Y-Motor, dann definieren wir die Zielfunktion

$$c_{\max 2x}(i,j) := \max\{2\,|x_i - x_j|, |y_i - y_j|\}.$$

Ein Betriebsingenieur ist möglicherweise gar nicht an der schnellsten Lösung interessiert, sondern an einer Tour mit insgesamt geringster Maschinenbeanspruchung. Die Maschinenbelastung entspricht der gesamten Laufzeit beider Motoren, es müssen also die Laufzeiten des X-Motors und des Y-Motors summiert werden. Dieser Abstand wird ℓ_1-*Abstand* oder *Manhattan-Abstand* genannt. Stellen Sie sich die Straßen von Manhattan vor und wie Sie von einer Kreuzung zu einer anderen gelangen. Wie berechnen Sie den (gefahrenen) Abstand zwischen diesen beiden Punkten? Diese Beobachtung gab dem Manhattan-Abstand seinen Namen.

Definition
Der *Manhattan-Abstand* zweier Punkte (x_i, y_i), (x_j, y_j):

$$c_1(i,j) := |x_i - x_j| + |y_i - y_j|$$

Schließlich kann man noch einen Kompromiss zwischen den Abständen $c_{\max}$ und c_1 suchen. Man möchte eine bezüglich der $c_{\max}$-Entfernung kürzeste Tour finden, und unter allen diesbezüglich minimalen Touren sucht man sich eine, die verschleißminimierend ist.

Hier ist die Grundidee: Man hat bereits eine ganzzahlige nichtnegative Zielfunktion wie z. B. $c_{\max}$. Dann möchte man unter allen optimalen Lösungen hinsichtlich dieser Zielfunktion eine Lösung haben, die bezüglich einer zweiten nichtnegativen ganzzahligen Zielfunktion so klein wie möglich ist. Zwei Zielfunktionen werden also geschickt miteinander kombiniert, so dass unter den verschiedenen Optimallösungen der ersten Zielfunktion diejenige herausgefischt wird, die für die zweite Zielfunktion den kleinsten Wert hat.

Um das zu erreichen, wendet man einen Trick an, den wir uns an unserem Beispiel klar machen. Eine optimale Rundreise für den $c_{\max}$-Abstand hat die Länge 43. Eine nächstschlechtere Lösung hätte die Länge 44. In dieser „Lücke“ zwischen 43 und 44 bringen wir nun die Werte der zweiten Zielfunktion unter. Wie macht man das?

Wählen wir als zweite Zielfunktion $c_{min}(i,j) := min\{|x_i - x_j|, |y_i - y_j|\}$. Wir bestimmen zuerst eine echte obere Schranke M für den Maximalwert der zweiten Zielfunktion. 180 ist eine leicht zu ermittelnde obere Schranke, denn es müssen 18 Kanten ausgewählt werden, von denen keine länger als 10 Längeneinheiten sein kann. Damit es eine echte obere Schranke wird, legen wir noch etwas drauf und machen 200 daraus.

Teilt man den Wert einer Lösung für c_{min} durch diese obere Schranke, so ergibt das eine Zahl, die größer oder gleich Null und kleiner als Eins ist. Addieren wir nun für eine Tour T zu $c_{max}(T)$ noch $\frac{c_{min}(T)}{200}$ hinzu, so passiert genau das Gewünschte: Die Länge von T bezüglich c_{max} vergrößert sich um einen Wert größer oder gleich Null und kleiner als Eins. Die Summe nimmt für die Touren den kleinsten Wert an, die für c_{max} optimal sind und unter diesen für c_{min} den kleinsten Wert haben. Diese Tour ist für c_{min} alleine betrachtet nicht unbedingt optimal, sondern nur unter den für c_{max} optimalen die beste. Der neue Abstand $c_{max\,min}$ ist also so definiert:

$$c_{max\,min}(i,j) := \max\{|x_i - x_j|, |y_i - y_j|\} + \frac{1}{200}\min\{|x_i - x_j|, |y_i - y_j|\}.$$

Jetzt kommt noch ein weiterer geschickter Schachzug: Wir haben ja die beiden Zielfunktionen c_{min} und c_{max} gewählt. Die Manhattan-Entfernung c_1 lässt sich damit auch so ausdrücken: $c_1 = c_{max} + c_{min}$.

Das kann nun in die Definition von $c_{max\,min}$ eingeflochten werden. Schreiben wir sie erst einmal so:

$$\begin{aligned} c_{max\,min}(i,j) := {} & \frac{199}{200}\max\{|x_i - x_j|, |y_i - y_j|\} \\ & + \frac{1}{200}\max\{|x_i - x_j|, |y_i - y_j|\} + \frac{1}{200}\min\{|x_i - x_j|, |y_i - y_j|\}. \end{aligned}$$

Daraus folgt dann die Gleichung

$$c_{max\,min}(i,j) := \frac{199}{200}\,c_{max}(i,j) + \frac{1}{200}\,c_1(i,j).$$

Die Parameter $\frac{199}{200}$ und $\frac{1}{200}$ zur Gewichtung der beiden Abstände sind also so gewählt, dass unter allen c_{max}-minimalen Touren eine c_1-minimale Tour ausgewählt wird.

- Experimentieren Sie anhand eigener Beispiele mit den verschiedenen Abständen. Wie unterscheiden sich die Resultate? Gibt es Kanten, die in jeder Version vorkommen?
- Wählen Sie sich einen Beispielgraphen mit verschiedenen Hamiltonkreisen. Stellen Sie eine Tabelle auf, in der angegeben ist, wie lang jeder Hamiltonkreis bezüglich der verschiedenen Abstände ist.

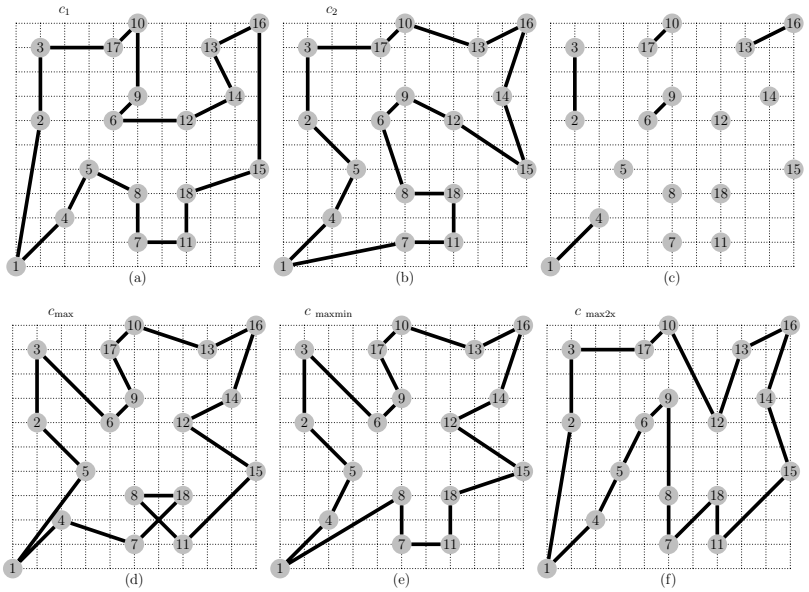


Abbildung 5. Optimale Touren für verschiedene Zielfunktionen. In (c) sieht man die Kanten, die in allen gezeigten Touren vorkommen.

- Konstruieren Sie ein Beispiel, das bei verschiedenen Zielfunktionen unterschiedliche Touren liefert.

Die dritte Aufgabe ist in doppelter Hinsicht schwer, zeigt aber auch deutlich, wie komplex das so einfach erscheinende TSP ist. Erstens müssen Sie für Ihr selbst konstruiertes Beispiel für die verschiedenen Zielfunktionen optimale Lösungen finden (mehr dazu im Abschnitt 5). Zweitens ist es nicht einfach, Beispiele zu finden, für die tatsächlich je nach gewähltem Abstand unterschiedliche optimale Touren existieren. Beginnen Sie mit sehr wenigen Knoten und fügen Sie nach und nach mehr Knoten hinzu. Testen Sie dabei immer, wie der neue Knoten sich in die bestehende Tour einfügt.

In Abbildung 5 sehen Sie verschiedene optimale Touren für das 18-Löcher-Bohrproblem: In 5(a) ist eine kürzeste Tour im Manhattan-Abstand c_1 zu sehen, Bild 5(b) zeigt eine kürzeste Tour bezüglich des euklidischen Abstands c_2. Die beiden Touren in 5(d), (e) sind bezüglich des max-Abstandes $c_{\max}$ minimal, wobei (e) auch bezüglich $c_{\max\min}$ minimal ist. Die Tour in (f) ist bezüglich der Entfernung $c_{\max 2x}$ minimal (mit langsamem X-Motor). Bild 5(c) zeigt diejenigen Kanten, die in allen vier optimalen Touren (a), (b), (d), (e) vorkommen.

Die folgende Tabelle gibt die Zielfunktionswerte der Touren (a), (b), (d), (e), (f) bezüglich der fünf Zielfunktionen c_1, c_2, c_{max}, $c_{max\,min}$, $c_{max\,2x}$ wieder.

	c_1-Wert	c_2-Wert	c_{max}-Wert	$c_{max\,min}$-Wert	$c_{max\,2x}$-Wert
Tour (a)	58	50,0822	47	47,0550	68
Tour (b)	60	48,5472	44	44,0800	72
Tour (d)	70	52,4280	43	43,1350	76
Tour (e)	64	49,5955	43	43,1050	72
Tour (f)	66	53,4779	48	48,0900	62

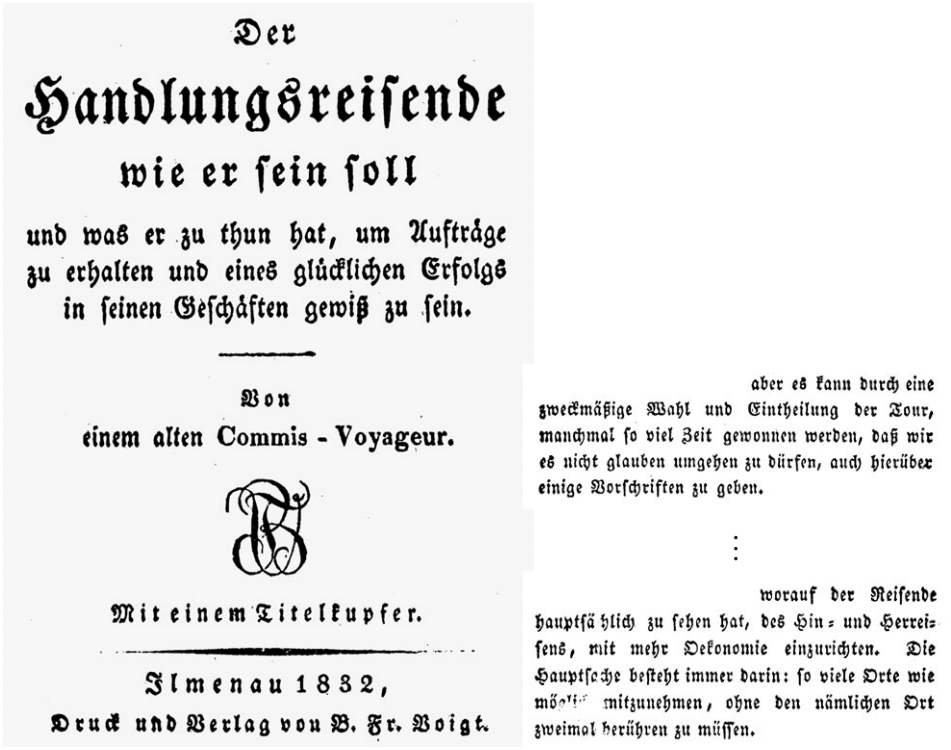
An den Touren aus unserem Beispiel sieht man bereits, dass man sich sehr genau überlegen muss, wie man ein Bohrproblem mathematisch modelliert. Der „wahre Abstand“ zwischen zwei Bohrlöchern ist nicht die offensichtliche euklidische Entfernung. In Tour (d) überschneiden sich sogar zwei Kanten. Intuitiv ist das unsinnig, aber wir optimieren hier bezüglich des max-Abstandes, und in diesem Fall ist das Wegstück (7, 18, 8, 11) genauso lang wie das Wegstück (7, 8, 18, 11), und das Programm entscheidet zufällig, welches Wegstück in die optimale Tour aufgenommen wird. Fazit: Praxisgerechte Problemlösungen erfordern Nachdenken.

Löcherbohren ist aufwändig. Leiterplatten gehen dabei gelegentlich zu Bruch insbesondere, wenn Löcher sehr eng beeinander liegen. Man kann sich daher fragen, ob man wirklich so viele Löcher bohren muss, um eine Leiterplatte funktionsfähig zu machen. Gibt es konstruktive Maßnahmen, um die Löcherzahl zu verringern?

Man kann das tatsächlich dadurch bewerkstelligen, daß man die Leiterbahnen anders auslegt. Besonders hilfreich ist dabei, eine „günstige“ Verteilung der Bahnstücke auf die Ober- und Unterseite. Hier stellt sich ein neues kombinatorisches Optimierungsproblem: Man weise die Leiterbahnen den zwei Seiten der Leiterplatte so zu, daß die Anzahl der zu bohrenden Löcher möglichst gering ist. Durch den Einsatz von mathematischer Optimierung kann man die Löcherzahl (gegenüber der gängigen Praxis) deutlich reduzieren, siehe hierzu [8], und somit auch die Arbeit für den Bohrer.

4 Der Ursprung des Travelling-Salesman-Problems

Ein Problem wie das TSP ist natürlich uralt. Mit Fragestellungen dieser Art beschäftigt sich bestimmt jeder, der Reisen plant. Eine der ersten Quellen, die das Problem explizit nennen, ist ein Reisehandbuch aus dem Jahre 1832. Abbildung 6 zeigt eine Kopie der Titelseite des Buches. Die zwei Textausschnitte rechts (Seiten 188 und 189 des Buches) beschreiben das TSP. Der Autor gibt dann auf den nächsten Seiten konkrete Touren, z. B. die „Tour von Frankfurt nach Sachsen“ auf



Der
Handlungsreisende
wie er sein soll
und was er zu thun hat, um Aufträge zu erhalten und eines glücklichen Erfolgs in seinen Geschäften gewiß zu sein.

Von
einem alten Commis - Voyageur.

Mit einem Titelkupfer.

Ilmenau 1832,
Druck und Verlag von B. Fr. Voigt.

aber es kann durch eine zweckmäßige Wahl und Eintheilung der Tour, manchmal so viel Zeit gewonnen werden, daß wir es nicht glauben umgehen zu dürfen, auch hierüber einige Vorschriften zu geben.

⋮

worauf der Reisende hauptsächlich zu sehen hat, des Hin- und Herreisens, mit mehr Oekonomie einzurichten. Die Hauptsache besteht immer darin: so viele Orte wie möglich mitzunehmen, ohne den nämlichen Ort zweimal berühren zu müssen.

Abbildung 6. Reisehandbuch von 1832 mit einer frühen Beschreibung des TSP

den Seiten 192–194, an, die er für günstig hält. Das kann man heute kaum noch überprüfen, da wir die Straßen- und sonstigen Reisebedingungen zur damaligen Zeit nicht mehr rekonstruieren können. Die aus heutiger Sicht kürzeste Rundreise durch die Städte in Hessen und Sachsen ist als grüne Tour in Abbildung 9 zu sehen.

Karl Menger, ein österreichischer Mathematiker, hat sich in den 20er Jahren des 20. Jahrhunderts offenbar erstmals aus mathematischer Sicht mit dem TSP beschäftigt. Die ersten wissenschaftlichen Arbeiten stammen aus den 50er Jahren, darunter der Artikel [4], in dem das erste „große“ TSP gelöst wurde. Die Autoren fanden eine kurze Rundreise durch 49 Städte in den USA und lieferten einen Beweis dafür, dass ihre Tour die kürzeste ist. Seither haben sich unzählige Mathematiker, Informatiker und Laien mit dem TSP beschäftigt. Übersichten hierzu findet man u. a. in den Büchern [15], [12].

Eine Eigenschaft, die das TSP so faszinierend macht, liegt darin, dass jeder, der sich mit dem Problem beschäftigt, sehr bald gute Einfälle zur Lösung des TSP hat.

- Liebe Leserin, lieber Leser, nehmen Sie sich jetzt nochmals etwas Zeit und denken Sie über Lösungsmethoden nach. Entwerfen Sie also Algorithmen zur Lösung für das TSP. Experimentieren Sie mit möglichst vielen verschiedenen Verfahren und mit unterschiedlichen Graphen.

Als Beispiele können Sie das 18-Löcher-Bohrproblem oder auch selbst gewählte Rundreiseprobleme verwenden.

Ich bin sicher, dass Ihnen mehr Methoden einfallen, als ich nachfolgend skizzieren werde. Wenn Sie ein paar Ideen entwickelt und ausprobiert haben, dann denken Sie bei jedem der Verfahren anschließend darüber nach, ob es tatsächlich optimale Lösungen liefert. Probieren Sie aus, was passiert, wenn Sie das gleiche Verfahren mit verschiedenen Startknoten durchführen. Bauen Sie sich auch verschiedene Graphen, mit denen Ihre Verfahren optimale Lösungen konstruieren. Und wundern Sie sich nicht, wenn Ihnen diese Aufgabe schwierig vorkommt, denn das hat grundsätzliche Ursachen.

5 Lösungsmethoden

Mathematiker unterscheiden bei kombinatorischen Optimierungsproblemen (das gilt analog auch für andere Optimierungsprobleme) grundsätzlich zwei Lösungsansätze: exakte Algorithmen und Heuristiken.

Exakte Algorithmen zeichnen sich dadurch aus, dass sie für jedes Problembeispiel (also in unserem Fall für jedes konkrete TSP mit jeder beliebigen „Entfernung“ zwischen zwei Knoten) eine Lösung konstruieren, die optimal ist. Und sie müssen einen Beweis für die Optimalität liefern! Bei *Heuristiken* ist der Anspruch geringer. Hier möchte man (möglichst in kurzer Zeit) eine zulässige Lösung finden (in unserem Falle eine Tour), und man hofft, dass der Wert der von der Heuristik gelieferten Lösung nicht zu weit vom Wert einer Optimallösung entfernt ist.

Laien ist der Unterschied häufig nicht so richtig klar. Ich habe mehrfach Briefe von Personen erhalten, die mir von ihren „revolutionären Methoden“ zur Lösung von TSPs berichteten und mich um Hilfe bei der Vermarktung ihrer Software baten. In der Regel hatten sie (falls sie nicht aufgrund von angeblich anstehenden Patentierungsverfahren „Nebel warfen“ und deswegen die Details nicht erklären wollten) eine einfache Heuristik entworfen, die bei ein paar selbst konstruierten kleinen Beispielen anscheinend optimale Lösungen geliefert hatte. Sie dachten, das wird dann auch bei anderen TSPs so sein, und hatten bereits Werbematerial für Software produziert, die Optimallösungen für alle TSPs versprach. Aber dafür braucht man einen mathematischen Beweis, sonst belügt man seine Kunden!

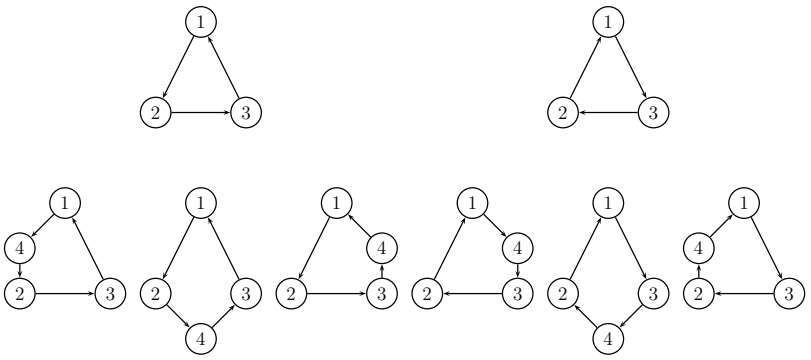


Abbildung 7. Alle verschiedenen Rundreisen durch 3 bzw. 4 Knoten für das ATSP

Exakte Algorithmen: Enumeration

Für einen „klassischen“ Mathematiker sind Travelling-Salesman-Probleme „trivial“. Die Anzahl der verschiedenen Touren ist, bei gegebener Städtezahl, offensichtlich endlich. Also kann man die Touren enumerieren (d. h., eine Tour nach der anderen erzeugen), ihre Länge bestimmen und dann eine Tour kürzester Länge auswählen. Dieses Argument ist prinzipiell korrekt. Dabei wird jedoch vergessen, dass man, wenn, z. B. die Bohrreihenfolge für eine Leiterplatte zu bestimmen ist, gerne eine Lösung hätte, bevor die Leiterplatte in Produktion geht, dass also die benötigte Rechenzeit für den Einsatz in der Praxis eine entscheidende Rolle spielt. Enumeration ist offenbar ein exakter Algorithmus, aber praktisch ist diese Methode unbrauchbar. Warum das so ist, überlegen wir uns anhand unseres 18-Städte-TSP.

- Wie viele verschiedene Touren gibt es in vollständigen Graphen mit n Knoten? Betrachten Sie das symmetrische und das asymmetrische TSP.

Wir beginnen mit dem ATSP und untersuchen zunächst kleine Beispiele: Bei zwei Städten gibt es eine Tour, bei drei Städten 2, bei vier Städten offensichtlich 6 verschiedene Touren. Kurzes Nachdenken liefert eine Konstruktionsmethode. Kenne ich alle Touren über $n-1$ Städte, so erhalte ich alle Touren mit n Städten auf folgende Weise. Ich liste alle Touren $T_1, T_2, \ldots$ mit $n-1$ Städten auf. Aus jeder einzelnen dieser Touren mache ich $n-1$ Touren über n Städte dadurch, dass ich nacheinander jeden Bogen (i,j) durch die zwei Bögen (i,n) und (n,j) ersetze. Aus den 2 Touren durch drei Städte werden so die 6 Touren durch vier Städte, siehe Abbildung 7.

Aus dieser Überlegung folgt, dass die Anzahl $A(n)$ aller Touren durch n Städte gleich der Anzahl aller Touren durch $n-1$ Städte ist, multipliziert mit $n-1$. Aus $A(2) = 1$, $A(3) = 2 = 2(A2)$ folgt dann unmittelbar (durch vollständige

Induktion):

Satz
Die Anzahl $A(n)$ aller ATSP-Touren durch n Städte ist

$$A(n) = (n-1)!$$

Beim symmetrischen TSP unterscheiden wir nicht zwischen einer (gerichteten) Tour und ihrer Rückrichtung. D. h., die Rundreise (1, 2, 3, 4) ist dieselbe wie (1, 4, 3, 2). Also ist die Anzahl $S(n)$ aller Touren des STSP durch n Städte gleich der Hälfte der Anzahl $A(n)$ der Touren des ATSP.

Satz
Beim symmetrischen TSP ist die Anzahl $S(n)$ aller Touren durch n Städte

$$S(n) = \frac{1}{2}A(n) = \frac{1}{2}(n-1)!$$

Die Fakultätsfunktion wächst ungeheuer schnell. Wollen wir unser 18-Löcher-Bohr-
problem lösen, so müssen wir $\frac{1}{2} \cdot 17!$ also 177 843 714 048 000 Touren generieren und ihre jeweilige Länge berechnen. Macht man das auf die offensichtliche Weise, so muss man für jede Tourlänge n Additionen vornehmen, in unserem 18-Städte-Beispiel immerhin rund 3,2 Billiarden Additionen. Es gibt jedoch wesentlich geschicktere Tourengenerierer, die auf Resultaten über hamiltonsche Graphen beruhen, bei denen im Durchschnitt nur jeweils 3 Additionen pro Tour erforderlich sind.

Mein Mitarbeiter Ralf Borndörfer hat so ein „schnelles“ Enumerationsverfahren implementiert. Es braucht auf einem Dell PC mit 2 GB Speicher und mit zwei Intel Pentium 4 Prozessoren, die mit 3,2 GHz getaktet sind, eine Gesamtrechenzeit von 15.233 Minuten und 42,78 Sekunden, also nicht ganz 11 Tage, um Tabelle 1 und die zugehörige Abbildung 8 für das 18-Löcher-Bohrproblem mit max-Entfernung zu erzeugen. (Eine Nebenbemerkung: Beim ersten Durchlauf, der „nur“ 13.219 Minuten und 59,77 Sekunden gedauert hatte, waren mehrere Überläufe bei dem Vektor aufgetreten, der für die möglichen Zielfunktionswerte inkrementell die Anzahl der Touren zählt, die diesen Wert als Lösung haben. Daraufhin mußte im C-Programm der Zählvektor von `int` (4 Bytes) auf `double` (8 Bytes) umdefiniert und die Rechnung mit der oben genannten Laufzeit wiederholt werden.)
Die Tabelle zeigt an, dass bei dieser Zielfunktion die kürzeste Tourlänge 43 und die längste 124 ist, dass also die 177 Billionen Touren nur Längen im Intervall [43,124] haben können. Darunter sind 16 Touren mit Länge 43. Diese sind also

Tabelle 1. Die möglichen Tourlängen des 18-Löcher Bohrproblems bei max-Entfernung und die Anzahl der Touren mit der jeweiligen Länge

43	16	71	172209541500	99	6978559023616
44	100	72	245244658270	100	6244495474690
45	800	73	343149491900	101	5479849473844
46	3534	74	471943012064	102	4715202571782
47	12956	75	638138778488	103	3975504420746
48	42912	76	848619318748	104	3280703432478
49	132438	77	1110034058608	105	2650666075940
50	372592	78	1428613199066	106	2091119982816
51	994256	79	1809054779498	107	1613047093070
52	2521592	80	2254667176880	108	1212028051160
53	6055074	81	2765433680186	109	888250844100
54	13902226	82	3338898869324	110	632612260292
55	30721582	83	3967855667630	111	437558030732
56	64973678	84	4641782959536	112	293477361242
57	132952132	85	5345084813004	113	189853217168
58	262678618	86	6058581971982	114	118785323640
59	502140880	87	6759692128388	115	71056777216
60	933147656	88	7422687410466	116	40793341944
61	1683569970	89	8022048930284	117	22247075616
62	2959612206	90	8530723566270	118	11388817040
63	5072963886	91	8926671324850	119	5540265968
64	8488258830	92	9188149461752	119	5540265968
65	13880473854	93	9303009856806	120	2397103968
66	22212414106	94	9262110464766	121	995323392
67	34793007114	95	9065959154040	122	326822400
68	53417929270	96	8722548741612	123	99496512
69	80409526082	97	8244549048098	124	18745344
70	118767208978	98	7656039959930		

alle optimal. Abbildung 8 zeigt auf einer logarithmischen Skala an, wieviele Touren welche Tourlänge haben.

Bedenkt man, dass man zur Lösung eines 25-Städte-Problems mit dieser Enumerationsmethode eine Zeit benötigt, die $18\times19\times\ldots\times24$-mal so lang ist, dann müsste unser Laptop um den Faktor 1,7 Milliarden schneller sein, wenn wir ein 25-Städte-TSP in rund ein bis zwei Wochen lösen wollten. Enumeration ist also nicht praxistauglich, und wir müssen uns etwas Besseres überlegen.

Exakte Algorithmen: Ganzzahlige Programmierung

Die Methoden, die heutzutage zur Optimallösung von TSPs herangezogen werden, sind äußerst kompliziert und können daher hier nicht im Einzelnen dargestellt werden. Ich will nur kurz das Vorgehen skizzieren.

Wir betrachten ein symmetrisches n-Städte-TSP. Jeder der $\frac{1}{2}(n-1)!$ Tou-

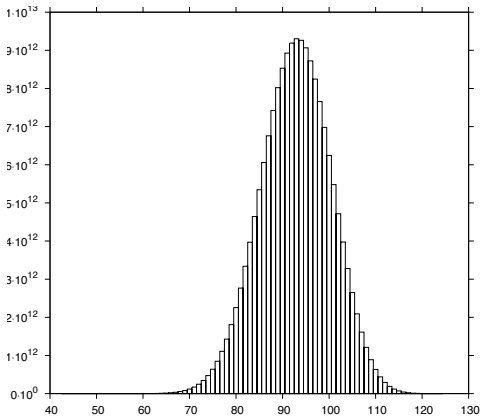


Abbildung 8. Häufigkeitsverteilung der Tourlängen für das 18-Löcher-Bohrproblem bezüglich der max-Entfernung

ren ordnen wir einen Vektor im $\frac{1}{2}n\,(n-1)$-dimensionalen reellen Vektorraum $\mathbb{R}^{\frac{1}{2}n(n-1)}$ zu. Dies geschieht auf folgende Weise. Jede Komponente eines Vektors dieses Raumes bezeichnen wir mit einer Kante des vollständigen Graphen K_n (der K_n hat genau $\frac{1}{2}n\,(n-1)$ Kanten). Jeder Tour T ordnen wir einen Tour-Vektor zu. Der Tour-Vektor zur Tour T enthält in jeder Komponente eine 1, die zu einer Kanten gehört, die in T enthalten ist. Alle übrigen Komponenten sind 0.

Beispiel: Betrachten wir die Tour (1, 3, 2, 5, 4), so enthält diese die Kanten 13, 14, 23, 25 und 45. Der K_5 enthält 10 Kanten, die wir wie folgt anordnen: 12, 13, 14, 15, 23, 24, 25, 34, 35, 45. Der zu der Tour (1, 3, 2, 5, 4) gehörige Tour-Vektor ist dann der folgende:
(0, 1, 1, 0, 1, 0, 1, 0, 0, 1).

Auf diese Weise erhält man $\frac{1}{2}\,(n-1)!$ Vektoren im $\mathbb{R}^{\frac{1}{2}n(n-1)}$. Die konvexe Hülle dieser Vektoren ist ein Polytop, genannt *Travelling-Salesman-Polytop*, dessen Ecken genau die Tour-Vektoren sind. In langjähriger Forschungsarbeit sind viele Ungleichungen entdeckt worden, die Facetten des Travelling-Salesman-Polytops definieren (in drei Dimensionen sind die Facetten die Seitenflächen eines Polytops). Mit diesen Ungleichungen, eingebettet in so genannte *Schnittebenen-Verfahren* der *linearen ganzzahligen Optimierung*, verknüpft mit *Branch&Bound-Methoden* und unter Ausnutzung verschiedener heuristischer Ideen, können symmetrische TSPs mit Tausenden von Städten optimal gelöst werden. Mehr hierzu kann man in den Artikeln [1], [2], [5],[14], [16], [17], [18] und insbesondere auf der Webseite http://www.tsp.gatech.edu/ finden.

Die hier kurz skizzierte Methodik ist eine grundsätzliche Vorgehensweise der ganzzahligen Optimierung. Sie ist insbesondere am TSP weiter entwickelt und verfeinert worden und gehört heute zum Standardrepertoire der ganzzahligen Optimierung, insbesondere wenn man beweisbar optimale Lösungen finden will.

Abbildung 9, mit Zustimmung der Autoren der Webseite http://www.tsp.gatech.

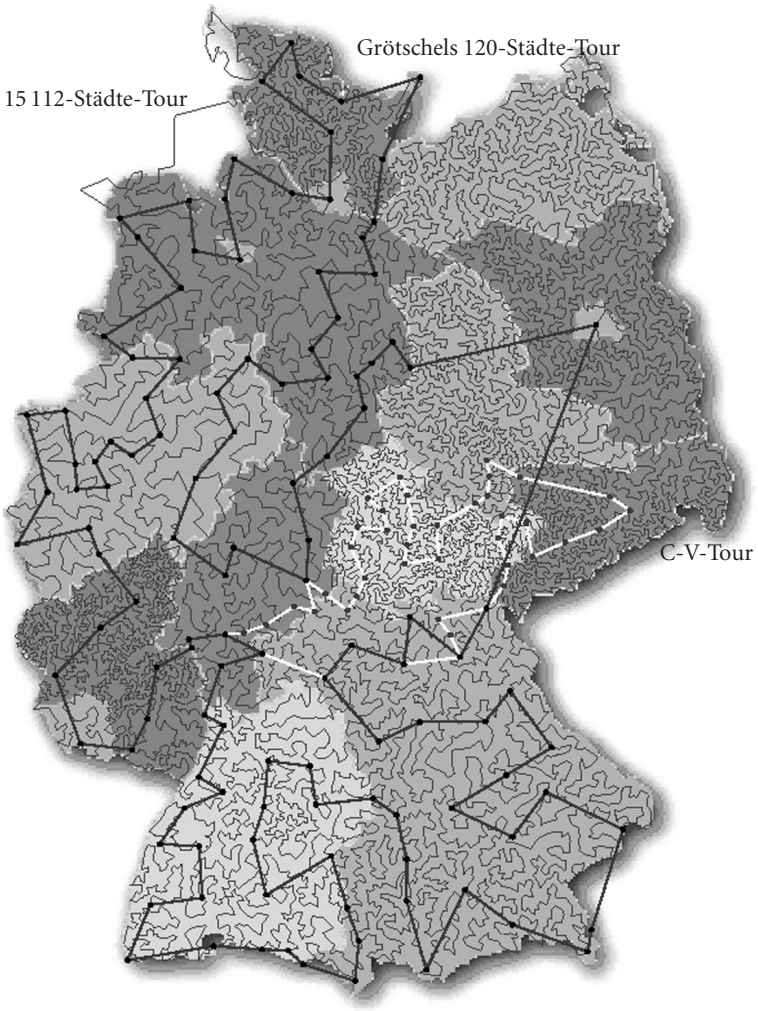


Abbildung 9. Drei verschiedene optimale Rundreisen durch Deutschland

edu/d15sol/dhistory.html entnommen, zeigt drei kürzeste Rundreisen durch Deutschland. Die grüne Tour ist die Optimallösung der Rundreise Frankfurt–Sachsen, aus dem Buch des alten Commis Voyageurs, siehe Abbildung 6. Eine Tour ist die Lösung eines 120-Städte-Problems, das der Autor 1977 gelöst hat, siehe [7]. Damals war das Weltrekord. Die lange Deutschlandreise geht durch 15.112 Orte. Sie wurde 2001 von Applegate, Bixby, Chvátal und Cook gefunden, damals ebenfalls Weltrekord. Der derzeitige Weltrekord (Juni 2005) ist die optimale Lösung eines Leiterplatten-Problems mit 33.810 Bohrlöchern. Sie wurde 2005 von William Cook, Daniel Epsinoza und Marcos Goycoolea gefunden.

Ich möchte hier nicht den Einduck erwecken, dass die Lösung der großen TSPs „mühelos“ ist. Dahinter stecken langjähriger Forschungsaufwand vieler und eine

große Zahl von Mannjahren an Implementationsarbeit. Allein der Rechenaufwand zur Lösung des deutschen 15.112-Städte-TSP war gewaltig. Die Lösung wurde auf 110 Prozessoren an der Rice University in Houston, Texas, und der Princeton University in Princeton, New Jersey, berechnet. Die Gesamtrechenzeit im Jahre 2001 (skaliert auf Compaq EV6 Alpha-Prozessoren mit 500 MHz) betrug 22,6 Jahre, bis ein Beweis für die Optimalität der gefundenen Tour erbracht war. Natürlich wird in der Praxis niemand so viel Aufwand treiben wollen. Aber hier geht es um die wissenschaftliche Entwicklung neuer Lösungsmethoden, und dann versucht man schon gerne einmal, neue Weltrekorde aufzustellen. (Es gibt ja auch Leute, die auf den Mount Everest steigen, obwohl das eigentlich völlig nutzlos ist.) Im Falle des TSP hat die Methodenentwicklung praktische Auswirkungen. Auf linearer Programmierung basierende Schnittebenenverfahren, verknüpft mit Branch&Bound und verschiedenen Heuristiken, haben sich als robuste und vielseitig verwendbare Lösungsmethoden für kombinatorische Optimierungsprobleme etabliert und werden bei vielen Anwendungsfällen wirkungsvoll eingesetzt.

Der TSP-Code *Concorde* der Kollegen Applegate, Bixby, Chvátal und Cook, der zur Lösung des 15.112-Städte-TSP benutzt wurde, kann von der Webseite http://www.tsp.gatech.edu/concorde/index.html heruntergeladen werden. Er liefert in vielen Fällen überraschend schnell eine optimale Lösung, häufig schneller, als eine „selbst gestrickte“ Heuristik eine ordentliche Lösung findet. Eine „praktisch akzeptable“ Laufzeitgarantie kann aber auch Concorde nicht abgeben. Die Kombination der verschiedenen Verfahren, die in Concorde stecken, kann in ungünstigen Fällen zu einer Laufzeit führen, die exponentiell in der Problemgröße ansteigt. Beim Concord-Code wird der (erfolgreiche) Versuch unternommen, die unvermeidbare Laufzeitexplosion durch Einsatz von hoch spezialisierten mathematischen Algorithmen so weit wie möglich hinauszuzögern. Das TSP ist jedoch eines der $\mathcal{NP}$-schweren Probleme (vgl. Kapitel 9), und so wird es wohl nie ein beweisbar schnelles Verfahren zur optimalen Lösung des TSP geben, es sei denn, jemand beweist $\mathcal{P} = \mathcal{NP}$. Dann werden die „algorithmischen Karten völlig neu gemischt werden“ müssen. (Daran glaube ich jedoch nicht.)

Das gegenwärtige „Super-TSP-Projekt“ besteht in dem Versuch, eine optimale Lösung eines symmetrischen TSP mit 1.904.711 Städten der Welt zu finden, siehe http://www.tsp.gatech.edu/world/pictures.html. Die beteiligten Kollegen haben inzwischen eine Tour konstruiert, deren Länge von einer (mit riesigem Aufwand berechneten) unteren Schranke für die optimale Tourlänge nur um 0,076 % abweicht. Für jeden praktischen Zweck ist das natürlich ausreichend, nicht jedoch, wenn man einen Beweis für die Optimalität liefern will.

Nebenbei bemerkt, diese Weltrekord-Geschichte bezieht sich nur auf das symmetrische TSP. Beim asymmetrischen TSP ist man bisher nicht in diese Größenordnungen vorgedrungen. Das ATSP scheint in der Praxis schwerer zu sein. Keiner weiß aber wirklich, warum und wieso.

Heuristiken

In der Praxis möchte man große TSPs meistens relativ schnell „lösen“. Man ist manchmal gar nicht an einer wirklichen Optimallösung interessiert. Eine „gute“ Lösung würde schon reichen, wenn sie in kurzer Zeit bestimmt werden könnte.

Ein Hauptgrund für die Reduzierung des Anspruchs ist, dass die Daten nicht genau ermittelt oder sogar nur geschätzt werden können. Die Fahrtzeit eines Autos zwischen zwei Orten zum Beispiel ist vom Fahrzeug, Verkehr, Wetter, etc. abhängig und nur grob abschätzbar. In solchen Fällen ist das Bestehen auf beweisbarer Optimalität eine überzogene Forderung. Ferner ist das TSP häufig nur eines von mehreren Problemen, die bearbeitet werden müssen. Bei der Produktion von Flachbaugruppen etwa sind die Herstellung der Leiterplatte selber, die Bestückung und die Verlötung viel wichtigere und zeitkritischere Operationen als das Bohren. Darüber hinaus werden Problemdaten aus den verschiedensten Gründen häufig kurzfristig geändert, so dass man immer wieder neue Durchläufe des Planungszyklus machen muss und sich daher der Aufwand zur Bestimmung jeweils bestmöglicher Lösungen nicht notwendig lohnt.

Dies sind die Szenarien, bei denen Heuristiken zum Einsatz kommen. Das Ziel ist hier, Algorithmen zu entwerfen, die gute (und möglichst auch beweisbar gute) Touren produzieren. Dabei ist man nicht so sehr an Verfahren interessiert, die für allgemeine TSPs ordentlich arbeiten. Sie sollen für die in einer Anwendung typischen TSP-Beispiele gute Lösungen liefern und dies in einer für die vorliegenden Anwendungsszenarien angemessenen Laufzeit auf Rechnern, die hierfür zur Verfügung stehen.

Solche Anforderungen sind bei meinen Industriekontakten fast immer gestellt worden. Bei der Optimierung des Bohrens von Vias zum Beispiel wurde ganz konkret vereinbart, dass TSPs in einer gewissen Größenordnung mit max-Abständen und relativ regulärer Bohrlochstruktur auf einem bestimmten PC in weniger als 3 Minuten (inklusive Zeit für Input und Output) zu lösen sind, siehe [9]. Der Entwurf von Heuristiken muss derartige Forderungen berücksichtigen.

Über solche Fälle im Detail zu berichten, würde hier zu weit führen, doch möchte ich einige prinzipielle Ideen vorstellen, die beim TSP ordentlich funktionieren, aber auch bei anderen kombinatorischen Optimierungproblemen in analoger Weise zum Tragen kommen können.

Greedy-Algorithmen

Greedy heißt gefräßig, und simplere Methoden als Greedy-Algorithmen sind kaum denkbar. Man betrachtet ein kombinatorisches Optimierungsproblem und definiert eine „gefräßige Auswahlregel“, der man einfach folgt. Da man viele solcher Regeln wählen kann, gibt es ganz unterschiedliche Greedy-Algorithmen. Bei der

Berechnung von minimalen aufspannenden Bäumen zum Beispiel (siehe Kapitel 2 dieses Buches) sortiert man die Kantengewichte aufsteigend (genauer: nicht absteigend) und arbeitet dann die Kanten in dieser Reihenfolge ab. Kann die nächste zur Auswahl stehende Kante zur Menge der bereits gewählten Kanten hinzugefügt werden, ohne dass ein Kreis entsteht, so nimmt man die Kante, sonst nicht. Erstaunlich ist, dass dieses myopische Verfahren tatsächlich in einem zusammenhängenden Graphen einen aufspannenden Baum mit minimalem Gewicht liefert.

Man kann dieses Verfahren auf das TSP wie folgt übertragen, was Sie eventuell bei Ihren eigenen Experimenten zur Konstruktion optimaler Touren auch in ähnlicher Weise getan haben. Man sortiert die Kanten wie oben und fügt eine zur Entscheidung anstehende Kante (also eine noch nicht verarbeitete Kante, die unter allen noch verfügbaren Kanten das kleinste „Gewicht“ hat) nur dann der Menge der bereits gewählten Kanten hinzu, wenn diese sich dann noch zu einer Tour ergänzen lassen. Das bedeutet, dass die jeweils ausgewählten Kanten aus einer Menge von Wegen bestehen, wobei eine Kante nur dann hinzugefügt wird, wenn entweder zwei Wege verbunden werden, ein Weg verlängert oder ein neuer Weg (bestehend aus nur einer Kante) zur gegenwärtigen Menge hinzugefügt wird. Praktische Tests haben gezeigt, dass dieses Verfahren (genannt *GREEDY-TSP*) keine guten Touren liefert.

Die Greedy-TSP-Heuristik
Eingabe: ein vollständiger gewichteter Graph
Ausgabe: ein Hamiltonkreis
1. Wähle eine noch nicht markierte Kante minimalen Gewichts und markiere sie, falls die markierten Kanten sich dann noch zu einer Tour ergänzen lassen.
2. Wiederhole 1., bis die markierten Kanten einen Hamiltonkreis bilden.

Überlegen Sie sich, wie man testen kann, ob die jeweilige neu gewählte Kante das Kriterium erfüllt und wie man der Menge der gewählten Kanten (ohne sie in den Graphen einzuzeichnen) ansieht, ob sie einen Hamiltonkreis bildet. Arbeiten Sie dabei wieder mit der Vorstellung der „Froschperspektive“ des Computers (vgl. Kapitel 1).

Viel besser (aber noch längst nicht gut) ist die folgende Variante des Greedy-Algorithmus, auf die die meisten bei ihren Überlegungen sehr schnell kommen. Man wählt einen beliebigen Knoten, sagen wir s, als Startknoten aus. Von diesem geht man zu dem am nächsten gelegenen Knoten. Ist man an einem beliebigen Konten i angelangt, so hat man einen Weg vom Startknoten s zu i gefunden. Nun wird der Weg zu einem Knoten j verlängert, der noch nicht im vorhandenen Weg ist und so nah wie möglich bei i liegt. Sind bereits alle Knoten im Weg vorhanden, so geht man zum Startknoten zurück. Dieses Verfahren heißt „Nächster Nachbar“

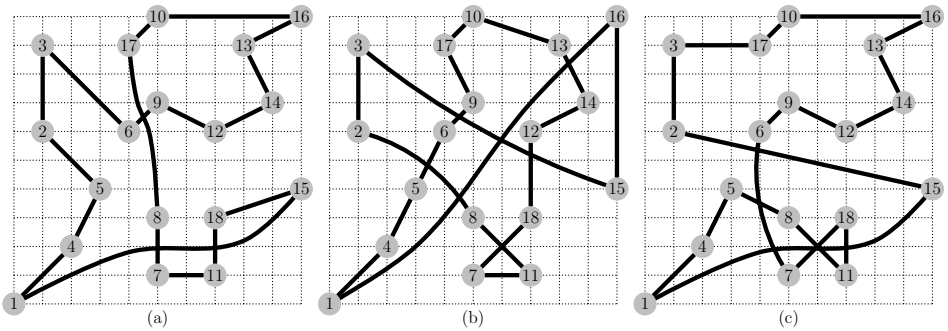


Abbildung 10. Nächster-Nachbar-Heuristik bezüglich $\max$-Abstand mit Startknoten 1: Es entstehen unterschiedliche Touren je nach Wahl des auf Knoten 5 folgenden Knotens.

oder kurz *NN-Heuristik*.

Die Nächster-Nachbar-Heuristik
Eingabe: ein vollständiger gewichteter Graph
Ausgabe: ein Hamiltonkreis
1. Wähle einen Startknoten s.
2. Gehe von s aus zu einem Knoten v mit minimalem Abstand zu s.
3. Gehe von dem eben erreichten Knoten v zu einem noch nicht besuchten Knoten mit minimalem Abstand zu v.
4. Falls es noch unbesuchte Knoten gibt, gehe zu 3., anderenfalls kehre zum Startknoten s zurück.

Betrachten wir unser 18-Städte-TSP mit max-Abstand. Wählen wir 1 als Startknoten, so müssen wir bei der NN-Heuristik zum Knoten 4 gehen, danach zu 5. Beim max-Abstand sind die Knoten 2, 6 und 8 gleich weit von 5 entfernt. Wir wählen dann einen der drei Knoten zufällig oder einfach den ersten der drei Knoten in der Nachbarliste, in diesem Falle wäre das Knoten 2. Sie sehen hier bereits, dass bei nicht eindeutig definiertem nächsten Nachbarn je nach Zufallsauswahl ganz unterschiedliche Touren entstehen können. Spielen Sie die verschiedenen Fälle bis zum Ende durch.

Abbildung 10 zeigt drei verschiedene Touren des 18-Städte-TSP mit $\max$-Entfernung, die bei Anwendung der NN-Heuristik mit Startknoten 1 und jeweils unterschiedlicher Auswahl der nächsten Nachbarn (bei gleicher Entfernung) produziert werden.

Die beiden oben skizzierten Heuristiken sind für das STSP formuliert worden, sie sind offensichtlich auch für das ATSP anwendbar. Greedy-Algorithmen sind beim TSP schnell. Die NN-Heuristik zum Beispiel berührt jede Kante höchstens zweimal, hat also eine Laufzeit von $O(n^2)$. (Die hier verwendete „O-Notation“ gibt eine obere Schranke für die Abschätzung der Laufzeit in Abhängigkeit von

der Größe des Inputs an. $O(n^2)$ bedeutet, dass der Algorithmus bei Eingabe eines Graphen mit n Knoten höchstens $an^2 + bn + c$ Rechenschritte machen muss.) Jeder (vernünftige) Algorithmus zur Lösung eines TSP sollte die Länge jeder Kante mindestens einmal anschauen, schneller als mit NN geht es also kaum. Aber die NN-Touren sind nicht wirklich „konkurrenzfähig". NN-Touren können nämlich beliebig weit vom Optimum entfernt sein.

Das können Sie sich klar machen, indem Sie die NN-Heuristik auf Graphen mit verschiedenen Gewichtungen anwenden. Das Problem ist die zuletzt gewählte Kante. Erfinden Sie Graphen, in denen es passieren kann, dass die NN-Heuristik die teuerste Kante wählen muss!

Eine kleine Modifikation kann die NN-Heuristik etwas verbessern, wobei es dann immer noch vorkommen kann, dass man meilenweit vom Optimum entfernt landet. Anstatt wie bei NN den Kreis schrittweise in eine Richtung wachsen zu lassen, werden hier in beiden Richtungen Kanten hinzugefügt, je nachdem, auf welcher Seite des bereits konstruierten Weges es gerade günstiger geht.

Die Doppelter-Nächster-Nachbar-Heuristik
Eingabe: ein vollständiger gewichteter Graph
Ausgabe: ein Hamiltonkreis
1. Wähle einen Startknoten s.
2. Wähle einen Knoten mit minimalem Abstand zu s und die zugehörige Kante.
3. Suche für beide Endknoten des bereits konstruierten Weges je einen nächstgelegenen Knoten. Füge den mit dem kleineren Abstand und die entsprechende Kante dem bereits konstruierten Weg hinzu.
4. Falls es noch unbesuchte Knoten gibt, gehe zu 3., anderenfalls füge die die beiden Endknoten des bereits konstruierten Weges verbindende Kante hinzu.

Approximationsalgorithmen für das STSP

Heuristiken, für die man (zumindest unter gewissen Umständen) eine Gütegarantie angeben kann, nennt man gelegentlich auch *Approximationsalgorithmen*. Wir unterstellen dabei (häufig ohne es explizit zu sagen), dass ein Approximationsalgorithmus eine polynomiale (bzw. eine in der Praxis brauchbare) Laufzeit haben sollte. Denn wenn wir auch exponentielle Laufzeiten zulassen, dann können wir ja gleich enumerieren und eine optimale Lösung bestimmen. Wie solch eine Gütegarantie definiert ist, finden Sie am Ende dieses Kapitels (Abschnitt „Vertiefung"). Dort wird auch gezeigt, dass es für das allgemeine STSP oder ATSP keine beispielunabhängigen Gütegarantien geben kann, falls sich die (wichtige und immer noch offene) Vermutung $\mathcal{P} \neq \mathcal{NP}$ (vgl. Kapitel 9) bestätigt.

Wenn man TSP-Heuristiken mit Gütegarantien entwickeln will, muss man aufgrund dieser Überlegungen Zusatzannahmen an die Zielfunktion machen. Eine diesbezüglich erfolgreiche Idee ist, geometrische Bedingungen zu fordern, die in der Praxis häufig erfüllt sind. Die beliebteste unter diesen Bedingungen ist die so genannte *Dreiecksungleichung*. Sie besagt, dass der direkte Weg von Stadt i nach Stadt k nicht länger sein darf als der Weg von i nach k über eine andere Stadt j, in Formeln:

$$c(i,k) \leq c(i,j) + c(j,k) \text{ für alle } 1 \leq i < j < k \leq n.$$

Beim Leiterplatten-Bohrproblem ist für alle betrachteten Zielfunktionen die Dreiecksungleichung erfüllt. (Sie sind aus mathematischer Sicht von so genannten Metriken abgeleitet.) Die Entfernungstabellen in Straßenatlanten erfüllen die Dreiecksungleichung hingegen meistens nicht. Das gilt z. B. für die Entfernungsmatrix, die dem deutschen 120-Städte-Problem aus Abbildung 9 zugrunde liegt. Der Grund dafür ist, dass diese Tabellen km-Angaben machen und bei der Entfernungsberechnung Wege auswählen, die einen möglichst hohen Autobahnanteil haben. Häufig sind diese Strecken zeitlich günstiger als die km-mäßig kürzeren über Landstraßen.

Wir betrachten nun folgende ganz einfache Heuristik für STSPs mit Zielfunktionen c, für die die Dreiecksungleichung gilt. Sie heißt *Spanning-Tree-Heuristik*, kurz *ST-Heuristik*.

Wir konstruieren in einem gewichteten vollständigen Graphen mit n Knoten einen minimalen aufspannenden Baum (vgl. Kapitel 2). Die Gesamtlänge dieses minimalen aufspannenden Baumes ist kleiner als die Länge einer optimalen Tour. Prüfen Sie dies an Beispielen nach. Aber warum ist das so? Überlegen Sie erst selbst, bevor Sie weiterlesen.

Satz
Gilt die Dreiecksungleichung, so ist eine optimale Tour in einem gewichteten Graphen immer mindestens so lang wie die Summe der Kantengewichte eines minimalen aufspannenden Baumes desselben Graphen.

Die Begründung ist nicht kompliziert, aber man kommt nicht ohne weiteres darauf: Entfernt man aus einer optimalen Tour T eine Kante, so entsteht ein aufspannender Baum B des Graphen. Dieser aufspannende Baum hat mindestens die gleiche Gesamtlänge wie ein minimaler aufspannender Baum B_{min} desselben Graphen. Es gilt also $c(B) \geq c(B_{min})$. Die Länge der entfernten Kante ist nicht negativ (siehe dazu die nächste Aufgabe). Darum gilt: $c(T) \geq c(B) \geq c(B_{min})$ □

- Zeigen Sie, warum ein Graph, in dem die Dreiecksungleichung erfüllt ist, keine negativen Kanten haben kann.

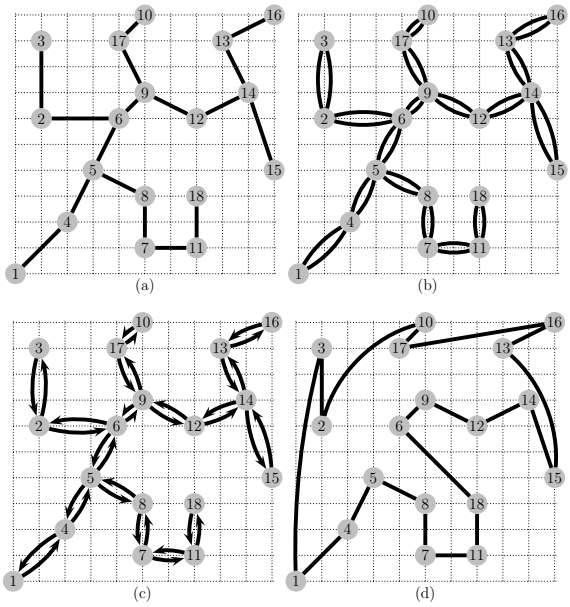


Abbildung 11. Konstruktion einer Tour mit der Spanning-Tree-Heuristik bezüglich der Manhattan-Erntfernung

Darüber hinaus muss ein Graph, in dem die Dreiecksungleichung erfüllt ist, vollständig sein, sonst könnte man nicht zwischen allen möglichen Kombinationen von drei Knoten Dreiecke finden.

Zurück zur Spanning-Tree-Heuristik: Nun wird der minimale aufspannende Baum in zwei Schritten in einen Hamiltonkreis umgebaut. Alle Kanten des Baumes werden verdoppelt. Der neue Graph G enthält somit zu jeder Kante eine parallele Kante. Abbildung 11 zeigt in (a) einen kürzesten aufspannenden Baum bezüglich der Manhattan-Entfernung und in (b) den Graphen G mit der verdoppelten Kantenmenge von B.

Was ist das Besondere an G? Für die Summe der Gewichte aller Kanten in G (hier bezeichnet mit $c(E)$) gilt, dass sie gleich dem doppelten Gewicht des Baumes ist und damit kleiner oder gleich dem doppelten Gewicht einer optimalen Tour T_{opt}.

$$c(E) = 2c(B) \leq 2c(T_{opt}).$$

Und G ist eulersch (siehe Kapitel 3 dieses Buches), d. h. G ist zusammenhängend und jeder Knotengrad ist gerade. Es gibt also eine eulersche Tour in G. Abbildung 11 (c) zeigt eine eulersche Tour, der wir eine (von mehreren möglichen) Orientierungen gegeben haben. Sie beginnt in Knoten 1, führt zu Knoten 4, dann zu 5, zu 8, 7, usw.

Jetzt kommt ein einfacher (aber pfiffiger) Schritt. Wir machen aus der orientierten eulerschen Tour einen hamiltonschen Kreis (= TSP-Tour) T durch Abkürzung. Wir wandern (z. B. von Knoten 1 aus startend) entlang der eulerschen Tour

und folgen der gewählten Orientierung. Gelangen wir vom gegenwärtigen Knoten i über den Bogen der Eulertour zu einem Knoten j, den wir noch nicht berührt haben, so nehmen wir die zugehörige Kante ij in die zu konstruierende TSP-Tour auf. Haben wir den Knoten j vorher schon besucht, so gehen wir entlang der Orientierung der Eulertour so lange weiter, bis wir einen bisher unbesuchten Knoten k gefunden haben, und nehmen die Kante ik in die TSP-Tour auf. Gibt es keinen solchen Knoten k mehr, dann müssen wir irgendwann zum Knoten 1 zurückkehren und schließen den hamiltonschen Kreis mit der Kante $i1$. Dies ist die von der ST-Heuristik konstruierte TSP-Tour T.

Die Spanning-Tree-Heuristik
Eingabe: ein vollständiger gewichteter Graph
Ausgabe: ein Hamiltonkreis
1. Konstruiere einen minimalen aufspannenden Baum B.
2. Verdoppele alle Kanten von B, es entsteht der eulersche Graph G.
3. Konstruiere eine Eulertour in G.
4. Wähle einen Startknoten und gehe die Eulertour ab. Lasse bereits besuchte Knoten weg und füge die direkte Verbindung zwischen den jeweils verbleibenden Knoten ein.

Verfolgen wir das Vorgehen in Abbildung 11 (c) und (d). Wir beginnen in 1, gehen dann zu 4, zu 5, zu 8, zu 7, zu 11 und zu 18. Die den durchlaufenen Bögen zugehörigen Kanten nehmen wir in die zu konstruierende Tour auf. Von 18 führt die Eulertour zu 11 (da waren wir schon), dann zu 7, zu 8, zu 5 (auch diese Knoten haben wir schon besucht), dann zu 6. Wir fügen nun die Kante von 18 nach 6 zur hamiltonschen Tour hinzu.

Was haben wir gemacht? Wir haben den Weg von 18 über 11, 7, 8 und 5, nach 6 durch die Kante von 18 nach 6 ersetzt. Da die Dreiecksungleichung gilt, erhalten wir

$$c(18,6) \leq c(18,11) + c(11,7) + c(7,8) + c(8,5) + c(5,6).$$

Also haben wir die Eulertour „abgekürzt“.

Jetzt geht es in unserem Beispiel weiter. Wir gehen von 6 nach 9, dann nach 12, nach 14 und nach 15. Von 15 geht es zurück nach 14 (da waren wir schon) und nach 13. Wir fügen zur TSP-Tour die Kante von 15 nach 13 hinzu und kürzen den Weg der Eulertour von 15 über 14 nach 13 ab. Und so weiter. Das Ergebnis ist in Abbildung 11 (d) zu sehen.

Das Besondere an diesem Verfahren ist, dass die Länge $c(T)$ der Tour, die so entsteht, durch den Wert $c(E)$, der die Länge der Eulertour bezeichnet, nach oben beschränkt ist, denn T entsteht aus E dadurch, dass Wege in E durch Kanten ersetzt werden. Die Länge einer Kante ik, die einen (i,k)-Weg ersetzt, ist aufgrund der Dreiecksungleichung nie länger als der ersetzte (i,k)-Weg. Wir erhalten die

Abschätzung:

$$c(T) \leq c(E) \leq 2c(T_{opt}).$$

Damit haben wir bewiesen, dass die Spanning-Tree-Heuristik, falls die Dreiecksungleichung gilt, eine Lösung produziert, die maximal 100 % von der optimalen Tourlänge abweicht, also eine sogenannte 1-approximative Heuristik ist. □

Satz
Für Entfernungen, die die Dreiecksungleichung erfüllen, konstruiert die Spanning-Tree-Heuristik Touren, die maximal 100 % von der optimalen Tourlänge abweichen.

Dieses Verfahren hat Christofides durch folgende Idee verbessert. Statt den kürzesten aufspannenden Baum B zu verdoppeln, berechnet man das minimale perfekte Matching M (vgl. Kapitel 3 und 7) auf den Knoten von B, die ungeraden Grad haben. Fügt man diese Matchingkanten dem minimalen aufspannenden Baum hinzu, so entsteht wiederum ein eulerscher Graph, und man konstruiert eine Tour T wie oben durch Abkürzung.

Die Christofides-Heuristik
Eingabe: ein vollständiger gewichteter Graph
Ausgabe: ein Hamiltonkreis
1. Konstruiere einen minimalen aufspannenden Baum B.
2. Suche ein minimales perfektes Matching auf den Knoten von B, die ungeraden Grad haben. Füge diese Kanten zu B hinzu. Es entsteht der eulersche Graph G.
3. Konstruiere eine Eulertour in G.
4. Wähle einen Startknoten und gehe die Eulertour ab. Lasse bereits besuchte Knoten weg und füge die direkte Verbindung zwischen den jeweils verbleibenden Knoten ein.

Diese Heuristik hat eine Gütegarantie von 0,5; die *Christofides-Tour* T ist also im schlechtesten Fall 50 % länger als die kürzeste Tour. Um diese Gütegarantie zu beweisen, betrachtet man die ungeraden Knoten und das minimale perfekte Matching. Diese Matchingkanten und die Kanten eines dazu „komplementären“ Matchings bilden einen Kreis, der aufgrund der Dreiecksungleichung höchstens so lang ist wie die optimale Tour. Die Kanten des minimalen perfekten Matchings sind zusammen höchstens halb so lang wie dieser Kreis. Der minimale aufspannende Baum ist, wie bereits gezeigt, höchstens so lang wie die optimale Tour und daher ist die Länge der Christofides-Tour höchstens 150 % der Länge einer optimalen Tour. Versuchen Sie, diese Argumentation mit Papier und Bleistift und vielen Beispielen nachzuvollziehen!

Satz
Für Entfernungen, die die Dreiecksungleichung erfüllen, konstruiert die Christofides-Heuristik Touren, die maximal 50 % von der optimalen Tourlänge abweichen.

Derzeit kennt man keinen Approximationsalgorithmus für STSPs, der eine bessere beweisbare Gütegarantie hat als die Christofides-Heuristik. Macht man jedoch stärkere Annahmen, zum Beispiel, dass das TSP geometrisch durch Punkte in der Ebene gegeben und die Entfernung zwischen zwei Punkten der euklidische Abstand ist, so kann man zu jedem $\epsilon > 0$ eine Heuristik mit polynomialer Laufzeit konstruieren, die die Gütegarantie ϵ besitzt, siehe hierzu [3].

Ein offenes Problem ist die Approximierbarkeit des ATSP. Hier kennt man, wenn man die Dreiecksungleichung z. B. voraussetzt, nur Approximationsalgorithmen, deren Güte von der Anzahl der Städte abhängt. Niemand weiß derzeit, ob unter diesen Voraussetzungen eine Heuristik mit einer konstanten Gütegarantie (sei die erlaubte Abweichung auch 1000 %) konstruiert werden kann.

Verbesserungsverfahren

Die oben skizzierten Heuristiken (GREEDY, NN, ST, Christofides) nennt man auch *Konstruktionsheuristiken*. Sie erzeugen bei ihrer Ausführung jeweils nur eine einzige Tour und hören dann auf.

- Entwickeln Sie Strategien, wie man bereits konstruierte Touren verbessern kann.

Schaut man sich die heuristisch gefundenen Touren in den Abbildungen 10 und 11 an, so sieht man sofort Verbesserungsmöglichkeiten. Ersetzt man z. B. in Abbildung 10(a) die Kanten von 8 nach 17 und 6 nach 9 durch die Kanten 6 nach 17 und 8 nach 9, so erhält man eine kürzere Tour. Einen solchen Tausch von zwei Kanten nennt man *Zweier-Tausch*. Diese Idee kann man erweitern und statt zwei auch drei, vier oder mehr Kanten gleichzeitig so austauschen, dass eine neue Tour entsteht. Man kann auch einen (oder mehrere) Knoten herausnehmen und an anderer Stelle wieder einsetzen, um Verbesserungen zu erzielen. In der Tour 10(c) kann man z. B. aus dem Weg von 2 nach 15 nach 1 den Knoten 15 entfernen, den Weg durch die Kante von 1 nach 2 ersetzen und den Knoten 15 zwischen 12 und 14 schieben, also die Kante 12, 14 durch den Weg von 12 über 15 nach 14 ersetzen. Das ist eine deutliche Verkürzung. Heuristiken, die mit einer Tour beginnen und iterativ die jeweils vorhandene Tour manipulieren, um eine bessere zu erzeugen, nennt man *Verbesserungsheuristiken*.

Es gibt enorm viele „Spielmöglichkeiten“ zum Austauschen von Kanten und Knoten. Man wendet einen solchen Austausch nicht nur einmal an, sondern versucht iterativ, die jeweils neu erzeugte Tour wieder durch einen Austausch zu ver-

122 Martin Grötschel

kürzen. In der Literatur firmieren solche Heuristiken häufig auch unter dem Begriff *local search*. Verbesserungsheuristiken sind die in der Praxis bei weitem erfolgreichsten Verfahren. Das derzeit am besten funktionierende Verbesserungsverfahren dieser Art basiert auf der *Lin-Kernighan-Heuristik*, deren präzise Beschreibung den Rahmen dieses Kapitels sprengen würde. Viele Experimente sind jedoch nötig, um die verschiedenen Parameter dieses Verfahrens gut aufeinander abzustimmen.

Man beobachtet bei praktischen Experimenten, dass Verbesserungsheuristiken in so genannten „lokalen Optima" stecken bleiben. Lokale Optima sind Touren, die durch den betrachteten Austauschmechanismus nicht verkürzt werden können. Dann gibt es zwei „Tricks", um aus einer solchen Falle herauszukommen. Man schaltet auf ein anderes Austauschverfahren um, oder man lässt (gesteuert durch einen gut überlegten Zufallsmechanismus) sogar Verschlechterungen der gegenwärtigen Tour zu in der Hoffnung, später zu noch besseren Touren zu gelangen. Heuristiken, die den Zufall einbauen, Verschlechterungen erlauben und weitere heuristische Ideen verwirklichen, findet man in der Literatur unter den Begriffen *simulated annealing*, *Tabu-Suche*, *genetische* oder *evolutionäre Algorithmen*.

Verbesserungsheuristiken haben hohe Laufzeiten. Schon der simple Zweier-Tausch z. B. benötigt $O(n^2)$ Schritte allein um nachzuweisen, dass eine gegebene Tour durch Zweier-Tausch nicht mehr verbessert werden kann. Manche Implementationen sind sogar im schlechtesten Fall exponentiell. Man muss daher Laufzeitkontrollen und Abbruchkriterien einbauen. Gute Gütegarantien kann man für diese Verfahren fast nie beweisen, aber die Praxis hat gezeigt, dass man mit geeigneten Verbesserungsverfahren sehr nahe an eine Optimallösung herankommt.

Der bereits erwähnte, im Internet für Forschungszwecke frei verfügbare Code *Concorde* stellt u. a. auch einige Heuristiken zum Download bereit, darunter die oben beschriebene NN-Heuristik (Nearest Neighbour) und eine Version des Austauschverfahrens, die *Chained Lin-Kernighan-Heuristik*, siehe http://www.tsp.gatech.edu//concorde/gui/gui.htm.

6 Vertiefung

Die Nichtapproximierbarkeit des TSP

Sei ALG eine TSP-Heuristik, die in polynomialer Laufzeit eine Tour liefert. Ist P ein konkretes TSP-Beispiel, so bezeichnen wir mit $ALG(P)$ den besten Lösungswert, den die Ausführung von ALG auf P erreichen kann. Mit $OPT(P)$ bezeichnen wir den Optimalwert von P. Ohne Beschränkung der Allgemeinheit können wir hier annehmen, dass alle Entfernungen positiv sind. Wir nehmen nun ferner an,

dass es ein $\epsilon > 0$ gibt, so dass für jedes TSP P Folgendes gilt

$$OPT(P) \leq ALG(P) \leq (1+\epsilon)OPT(P)\,.$$

Die Zahl ϵ, falls sie existiert, nennen wir *Gütegarantie*. Ist z. B. $\epsilon = 0,5$, so bedeutet dies, dass für jedes beliebige TSP der Wert der Lösung, die ALG bestimmt, höchstens 50 % größer ist als der Optimalwert des TSP.

ALG bezeichnen wir als Heuristik mit Gütegarantie ϵ oder kurz als ϵ-*approximative* Heuristik.

Zunächst zeigen wir, dass man für das allgemeine STSP oder ATSP überhaupt keine (beispielunabhängige) Gütegarantie beweisen kann, es sei denn $\mathcal{P} = \mathcal{NP}$.

Wir überlegen uns, dass wir, wenn es eine ϵ-approximative Heuristik ALG für das TSP mit polynomialer Laufzeit gibt, ein $\mathcal{NP}$-schweres Problem in polynomialer Zeit lösen können. In unserem Fall werden wir einen polynomialen Algorithmus konstruieren, der entscheiden kann, ob ein Graph hamiltonsch ist oder nicht.

Das geht wie folgt. Sei $G = (V, E)$ ein beliebiger Graph (oder analog ein gerichteter Graph) mit n Knoten. Wir konstruieren ein TSP, das wir G_ϵ nennen wollen. Jeder Kante in E geben wir den Wert 1, jeder Kante des K_n, die nicht in E ist, den Wert $M := n\epsilon + 2$, siehe hierzu auch die Diskussion des Digraphen in Abbildung 1. G_ϵ hat folgende Eigenschaft. Ist G hamiltonsch, so hat G_ϵ den Optimalwert n. Ist G nicht hamiltonsch, so muss eine kürzeste Tour mindestens eine Kante mit Wert M und höchstens $n-1$ Kanten mit Wert 1 enthalten. Also ist der Optimalwert von G_ϵ mindestens $n - 1 + M$.

Nun lösen wir das so definierte TSP G_ϵ mit ALG und erhalten eine Tour T mit dem Wert $ALG(G_\epsilon)$. Gilt $ALG(G_\epsilon) = n$, so wissen wir, dass der Graph G hamiltonsch ist. Enthält T eine Kante, die nicht aus G ist, so gilt

$$ALG(G_\epsilon) \geq (n-1) + M.$$

Aufgrund der (gut gewählten) Definition von M können wir daraus schließen, dass G nicht hamiltonsch ist. Das geht so. Wenn G hamiltonsch ist, so hat das TSP G_ϵ den Optimalwert n. Da wir angenommen haben, dass ALG ϵ-approximativ ist, gilt die Gütegarantie $ALG(G_\epsilon) \leq (1+\epsilon)OPT(G_\epsilon)$ natürlich auch für das TSP G_ϵ, und das heißt:

$$(n-1) + M = (n-1) + n\epsilon + 2 \leq ALG(G_\epsilon) \leq (1+\epsilon)OPT(G_\epsilon) = (1+\epsilon)n.$$

Hieraus folgt $n\epsilon + 1 \leq n\epsilon$, ein Widerspruch.

Dies bedeutet, dass ALG, wenn G hamiltonsch ist, aufgrund seiner Gütegarantie immer eine Tour mit Länge n, also einen hamiltonschen Kreis, finden muss. Da ALG polynomiale Laufzeit hat, findet ALG in polynomialer Zeit einen Beweis dafür, dass ein gegebener Graph hamiltonsch ist oder nicht. Daraus aber folgt $\mathcal{P} = \mathcal{NP}$.

Wir schließen daraus, dass es, falls $\mathcal{P} \neq \mathcal{NP}$ gilt, keine Heuristik für das TSP mit einer Gütegarantie ϵ gibt. Dies gilt sowohl für das STSP als auch für das ATSP. □

Zufall und das TSP

Es gibt noch eine Forschungsrichtung zum TSP, die bisher nicht angesprochen wurde: TSP und Stochastik. Was könnte man da forschen? Ich will einige interessante Themen kurz anreißen.

Bei den bisher betrachteten Heuristiken hatten wir „worst-case-Abschätzungen“ gemacht, denn beim Beweis einer Gütegarantie muss man den schlechtest möglichen Fall untersuchen. Wie wäre es, stattdessen den „average case“ zu untersuchen, also Aussagen darüber zu beweisen, wie gut eine Heuristik „im Durchschnitt“ ist?

Wir hatten gesehen, dass bei einigen Heuristiken, wenn mehrere gleichwertig erscheinende Auswahlmöglichkeiten bestehen, eine zufällige Auswahl der nächsten in die Tour einzubauenden Kante getroffen werden kann. Man kann diese Idee konsequenter ausnutzen und den Zufall als algorithmisches Instrument an verschiedenen Stellen einsetzen. Das Ergebnis eines solchen „randomisierten Verfahrens“ ist dann nicht mehr deterministisch sondern ein Zufallsereignis. Wenn man tatsächlich „richtig würfelt“, kommt bei ein und demselben TSP-Beispiel in fast jedem Lauf eine andere Tour heraus. Kann man für einen solchen Algorithmus die erwartete Tourlänge bestimmen?

Resultate dieser Art können nur innerhalb eines adäquaten stochastischen Modells bewiesen werden. Man muss dazu einen Wahrscheinlichkeitsraum wählen und annehmen, dass TSP-Beispiele nach einer zu spezifizierenden Wahrscheinlichkeitsverteilung aus diesem Raum gezogen werden. Wenn man solch eine Annahme gemacht hat, kann man sogar noch kühnere Fragen stellen. Was sind beispielsweise typische Eigenschaften von TSPs dieser Art?

Die stochastische Analyse selbst einfacher Sachverhalte in diesem Zusammenhang ist kompliziert und bedarf „harter“ Stochastik und Analysis. Ich möchte lediglich zwei Ergebnisse beschreiben, die die stochastische Analyse des TSP erbracht hat und die zeigen, welch überraschende Resultate (unter gewissen Vorraussetzungen) manchmal mit diesen Techniken bewiesen werden können.

Wir stellen uns folgende Situation vor. Wir betrachten das Einheitsquadrat. Die Entfernung zwischen zwei Punkten sei hierbei der euklidische Abstand. Auf diese Weise haben wir einen Wahrscheinlichkeitsraum definiert. Ein zufälliges TSP ist durch die zufällige Wahl von n Punkten definiert. Hier die Frage: Wie lang ist die kürzeste Tour durch zufällig gewählte n Punkte im Einheitsquadrat?

Auf den ersten Blick erscheint die Frage absurd. Natürlich wird man die optimale Tourlänge, wir wollen sie L_n nennen, nicht genau bestimmen können. Aber

kann man vielleicht etwas über den Erwartungswert, wir bezeichnen ihn mit $E(L_n)$, aussagen? Und das kann man in der Tat sehr präzise. Beardwood, Halton und Hammersley haben 1959 bewiesen, dass es zwei positive, von n unabhängige Konstante k_1 und k_2 gibt, so dass Folgendes gilt

$$k_1\sqrt{n} \leq E(L_n) \leq k_2\sqrt{n}\,.$$

Mehr noch, es gibt eine von n unabhängige Konstante k mit

$$\lim_{n\to\infty}\left(\frac{E(L_n)}{\sqrt{n}}\right) = k\,.$$

Mit anderen Worten: Zieht man n zufällige Punkte aus dem Einheitsquadrat, dann ist die Länge der kürzesten Rundreise durch die n Punkte (bei euklidischer Distanz) mit großer Wahrscheinlichkeit $\sqrt{n}$ multipliziert mit einer Konstanten. Analytisch hat man die Konstanten bisher nicht bestimmen können. Rechenexperimente legen nahe, dass k ungefähr 0,7124 ist, siehe [13]. Es gibt ferner polynomiale Algorithmen, die für diese Art von TSPs dieselbe Asymptotik haben, d. h., sie liefern bei großen TSPs mit großer Wahrscheinlichkeit nahezu optimale Touren.

Wir betrachten ein zweites stochastisches Modell. Jeder Kante des vollständigen Graphen K_n weisen wir einen Wert aus dem Intervall $[0, 1]$ als Kantenlänge zu. Dabei wird jede Kantenlänge unabhängig von den anderen Kantenlängen aus $[0, 1]$ gezogen. Jedes Element des Intervalls $[0, 1]$ hat dabei die gleiche Wahrscheinlichkeit, gezogen zu werden. Es ist bekannt, dass man ein minimales perfektes 2-Matching in polynomialer Zeit berechnen kann. Ein perfektes 2-Matching ist eine Menge von Kanten, so dass jeder Knoten auf genau 2 Kanten liegt. Perfekte 2-Matchings sind somit Kantenmengen, die aus lauter Kreisen bestehen, wobei jeder Knoten in genau einem Kreis vorkommt. Touren sind also spezielle perfekte 2-Matchings. Hieraus folgt, dass die minimale Länge eines perfekten 2-Matchings eine untere Schranke für die mimimale Tourlänge ist. Kürzlich hat Alan Frieze [6] gezeigt, dass – für diese spezielle Art von symmetrischen TSPs – mit großer Wahrscheinlichkeit die kürzeste Tourlänge nur geringfügig von der minimalen Länge eines perfekten 2-Matchings abweicht und dass man eine Tour in polynomialer Zeit konstruieren kann, die diese Abweichung mit hoher Wahrscheinlichkeit realisiert.

Genaue Formulierungen dieser und ähnlicher Resultate erfordern hohen technischen Aufwand. Der interessierte Leser findet mehr dazu in Kapitel 6 von [15] und Kapitel 7 von [12].

Sind Resultate dieser Art für TSPs aus der Praxis relevant? Das ist nicht so klar. Das zweite Modell, bei dem jede Kante eine Länge im Wertebereich $[0, 1]$ erhält, scheint kein geeigneter stochastischer Rahmen für TSPs zu sein, jedenfalls nicht für

die, die mir in der Praxis begegnet sind. Das erste Modell könnte man durchaus beim Leiterplattenbohren anwenden. Wenn die Bohrlöcher zufällig auf dem Einheitsquadrat verteilt sind und die Anzahl der Bohrlöcher groß ist, so müßte bei euklidischer Distanz eine Tourlänge im Bereich $0,6\sqrt{n}$ bis $0,8\sqrt{n}$ herauskommen. Schauen wir unser 18-Städte-TSP auf einem 11×11-Gitter an. Die bezüglich des euklidischen Abstands optimale Tourlänge ist 48,5 (siehe die Tabelle auf Seite 104). Wenn das Problem zufällig gewählt wäre, sollte man einen Optimalwert von rund 33,3 erwarten. Aber das Beispiel ist nicht zufällig (allein 4 der 18 Löcher liegen auf dem Rand) und $n = 18$ ist „zu klein". Die Asymptotik zieht hier noch nicht. Dennoch, in [13] wurde die Asymptotik mit der optimalen Tourlänge für große reale TSPs empirisch verglichen, und es kam dabei eine durchaus akzeptable Übereinstimmung heraus.

7 Lösungen und Literaturhinweise

Die Länge einer optimalen Tour für das Rheinlandproblem beträgt 617.

Der gerichtete Graph in Abbildung 1 enthält keinen gerichteten hamiltonschen Kreis. Er hat aber eine interessante Eigenschaft. Wenn man irgendeinen beliebigen der 12 Knoten und die mit ihm inzidierenden Bögen aus dem Graphen entfernt, so ist der dadurch entstehende gerichtete Graph hamiltonsch. Graphen und Digraphen, die nicht hamiltonsch sind, und bei denen das Entfernen eines beliebigen Knotens jeweils zu einem hamiltonschen Graphen oder Digraphen führt, nennt man *hypohamiltonsch*, siehe [11].

Zum Travelling-Salesman-Problem gibt es eine sehr umfangreiche Literatur. Sie richtet sich jedoch vornehmlich an professionelle Mathematiker und Informatiker. Eine allgemein verständliche Einführung ist z. B. [10].

Gute Sammelbände, in denen jeweils in rund 15 Kapiteln verschiedene Aspekte des TSP (Geschichte, Anwendungen, leicht lösbare Spezialfälle, Heuristiken, exakte Algorithmen, Polyedertheorie des TSP, Verallgemeinerungen) abgehandelt werden, sind [12] und [15]. Hier finden sich „unendlich viele" Referenzen zum TSP und zu verwandten Fragestellungen der kombinatorischen Optimierung, knapp 500 in [15] und fast 750 in [12].

In [15] behandeln z. B. die Kapitel 5, 6 und 7 heuristische Lösungsmethoden und deren Analyse, während in [12] die Kapitel 5, 6, 7, 8, 9 und 10 dem Thema Heuristiken gewidmet sind. Exakte Methoden und die dahinter stehende Theorie (lineare und ganzzahlige Optimierung, Schnittebenenverfahren, Branch&Bound, polyedrische Kombinatorik) werden in [15] in den Kapiteln 8, 9 und 10 und in [12] u. a. in den Kapiteln 2, 3, 4 beschrieben. Ein weiterer Übersichtsartikel, der sich an eine breitere (jedoch mathematisch geschulte) Leserschaft richtet und speziell auf algorithmische und rechentechnische Apekte eingeht, ist [14].

Es gibt mehrere Webseiten zum TSP. Die bereits mehrfach erwähnte http://www.tsp.gatech.edu/, die von Bill Cook (Georgia Tech, Atlanta, Georgia, USA) gepflegt wird, offeriert nicht nur einen geschichtlichen Überblick u. a. mit Bildern der jeweiligen Weltrekorde, sie beschreibt auch das konkrete Vorgehen bei der Lösung von TSPs und bietet exzellente TSP-Software zum Download an (sehr empfehlenswert).

TSPLIB, eingerichtet von Gerhard Reinelt (Universität Heidelberg), siehe http://www.iwr.uni-heidelberg.de/groups/comopt/software/TSPLIB95/, ist eine Sammlung von TSP-Beispielen (und Varianten des TSP), vornehmlich aus konkreten Anwendungsfällen. Wer heute TSP-Algorithmen entwirft, sollte seine Methoden an dieser Datensammlung testen, um seine Ergebnisse mit dem „Rest der Welt" vergleichen zu können.

Kapitel 16 des Sammelbandes [12], geschrieben von A. Lodi und A. P. Punnen, behandelt „TSP Software". Die Autoren haben die folgende Webseite aufgelegt: http://www.or.deis.unibo.it/research_pages/tspsoft.html, derzeit gepflegt von Matteo Boccafoli (Università di Bologna, Bologna, Italien), auf der Links zu im Augenblick im Internet verfügbarer TSP-Software (exakte Verfahren, Heuristiken, Java-Applets, etc.) zu finden sind. Die Webseite „TSPBIB Home Page", http://www.ing.unlp.edu.ar/cetad/mos/TSPBIB_home.html, entworfen und gepflegt von Pablo Moscato (Universidade Estadual de Campinas, Campinas, Brasilien) enthält eine umfangreiche Auflistung von Artikeln, Software, Preprints und Links zu anderen Webseiten, die sich mit dem TSP und verwandten Themen beschäftigen.

Literaturverzeichnis

[1] David Applegate, Robert Bixby, Vasek Chvátal, and William Cook. On the Solution of Traveling Salesman Problems. *Documenta Mathematica, Extra Vol. ICM Berlin 1998*, III:645–656, 1998.

[2] David Applegate, Robert Bixby, Vasek Chvátal, and William Cook. Implementing the Dantzig-Fulkerson-Johnson algorithm for large traveling salesman problems. *Mathematical Programming, Series B*, 97(1-2):91–153, 2003.

[3] Sanjeev Arora. Approximation Algorithms for Geometric TSP. In Gregory Gutin and Abraham P. Punnen, editors, *The Traveling Salesman Problem and Its Variations*, pages 207–221. Kluwer, Dordrecht, 2002.

[4] George Dantzig, Ray Fulkerson, and Selmer Johnson. Solution of a Large-Scale Traveling Salesman Problem. *Operations Research*, 2:393–410, 1954.

[5] Matteo Fischetti, Andrea Lodi, and Paolo Toth. Exact Methods for the Asymmetric Traveling Salesman Problem. In Gregory Gutin and Abraham P. Punnen, editors, *The Traveling Salesman Problems and Its Variations*, pages 169–205. Kluwer, Dordrecht, 2002.

[6] Alan Frieze. On Random Symmetric Travelling Salesman Problems. *Mathematics of Operations Research*, 29(4):878–890, 2004.

[7] Martin Grötschel. *Polyedrische Charakterisierungen kombinatorischer Optimierungsprobleme*. Anton Hain, Meisenheim am Glan, 1977.

[8] Martin Grötschel, Michael Jünger, and Gerhard Reinelt. Via Minimization with Pin Preassignments and Layer Preference. *ZAMM – Zeitschrift für Angewandte Mathematik und Mechanik*, 69(11):393–399, 1989.

[9] Martin Grötschel, Michael Jünger, and Gerhard Reinelt. Optimal Control of Plotting and Drilling Machines: A Case Study. *Zeitschrift für Operations Research*, 35(1):61–84, 1991.

[10] Martin Grötschel and Manfred Padberg. Die optimierte Odyssee. *Spektrum der Wissenschaft*, 4:76–85, 1999.

[11] Martin Grötschel and Yoshiko Wakabayashi. Hypohamiltonian Digraphs. *Methods Oper. Res.*, 36:99–119, 1980.

[12] Gregory Gutin and Abraham P. Punnen, editors. *The Traveling Salesman Problem and Its Variations*. Kluwer, Dordrecht, 2002.

[13] D. S. Johnson, L. A. McGeoch, and E. E. Rothberg. Asymptotic Experimental Analysis for the Held-Karp Traveling Salesman Bound. In *Proceedings of the 7th Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 341–350, 1996.

[14] Michael Jünger, Gerhard Reinelt, and Giovanni Rinaldi. The Traveling Salesman Problem. In M. O. Ball, T. L. Magnanti, C. L. Monma, and G. L. Nemhauser, editors, *Network Models*, pages 225–330. North-Holland, Amsterdam, 1995.

[15] E. L. Lawler, J. K. Lenstra, A. H. G. Rinnooy Kan, and D. B. Shmoys, editors. *The Traveling Salesman Problem. A Guided Tour of Combinatorial Optimization*. Wiley, Chichester, 1985.

[16] Denis Naddef. Polyhedral Theory and Brunch-and-Cut Algorithms for the Symmetric TSP. In Gregory Gutin and Abraham P. Punnen, editors, *The Traveling Salesman Problems and Its Variations*, pages 29–116. Kluwer, Dordrecht, 2002.

[17] Manfred Padberg and Martin Grötschel. Polyhedral computations. In E. L. Lawler, J. K. Lenstra, A. H. G. Rinnooy Kan, and D. B. Shmoys, editors, *The Traveling Salesman Problem. A Guided Tour of Combinatorial Optimization*. Wiley, Chichester, 307–360, 1985.

[18] Manfred Padberg and G. Rinaldi. A branch-and-cut algorithm for the resolution of large-scale symmetric traveling salesman problems. *SIAM Rev.*, 33(1):60–100, 1991.