Optimale Wege

Anton Schüller[1]
Ulrich Trottenberg[1,2]
Roman Wienands[2]

[1]**Fraunhofer-Institut**
Algorithmen und Wissenschaftliches Rechnen SCAI

[2] **Mathematisches Institut**
der Universität zu Köln

Version 1.1
27.09.2012

Inhaltsverzeichnis

1 Vorbemerkung: Kombinatorische Optimierungsprobleme im Schulunterricht

Aufgabenstellungen aus dem Bereich der kombinatorischen Optimierung spielen in der Schulmathematik nur eine untergeordnete Rolle, obwohl uns derartige Problemstellungen im täglichen Leben vielfach begleiten, z.B.

- Wie packe ich die Spülmaschine so, dass möglichst viel hineinpasst?
- Wie packe ich den Kofferraum so, dass möglichst das ganze Urlaubsgepäck inklusive sperriger Koffer hineinpasst?
- Wie komme ich am schnellsten mit dem PKW von A nach B und welche interessanten Alternativen gibt es, z.B. wenn auf der schnellsten Route ein Stau auftritt?
- Wie komme ich am schnellsten mit öffentlichen Verkehrsmitteln von C nach D?

Dabei ist die kombinatorische Optimierung ein Thema, das sehr gut in den mathematischen Unterricht passt (vgl. z.B. [2, 3, 4, 5]).

Wir werden in diesem Unterrichtsmodul mit einer Aufgabenstellung aus der kombinatorischen Optimierung beginnen, die für die Schüler/innen einfach verständlich ist und sie mit mathematischem und algorithmischem Denken vertraut macht.

Die Schüler/innen gehen frei an die Aufgabe heran, ohne dass vorher Lösungswege oder Lösungsansätze im Unterricht diskutiert worden sind. Dabei macht die Beschäftigung mit dieser Aufgabenstellung für die Schüler/innen deutlich, dass Mathematik durchaus etwas anderes und viel mehr ist als pures Rechnen und dass Mathematik ein starkes kreatives Element hat. Ferner erfahren die Schüler/innen, dass sie in der Lage sind, selbständig und kreativ an völlig neue Problemstellungen heranzugehen, dass sie selber und ohne fremde Hilfe Algorithmen entwickeln können, und sie lernen zudem noch eine ganze Menge über das Lösen von Optimierungsproblemen.

Einen besonderen Reiz macht diese Aufgabenstellung auch dadurch aus, dass sie mathematisch durchaus kompliziert ist.

Wir beschränken uns in diesem Unterrichtsmodul auf das Problem, eine möglichst kurze Rundreise durch eine Reihe vorgegebener Städte zu finden, wobei die Entfernungen zwischen allen Städten bekannt sind (Travelling-Salesman-Problem). Einen guten Überblick über diese Problemstellung und unterschiedliche Ansätze zu ihrer Lösung findet man in [2, 6].

Das zugehörige Unterrichtsmaterial eignet sich für eine Unterrichtsreihe ab etwa Klasse 8 oder 9 und kann im Mathematik-Unterricht, im Differenzierungsbereich Mathematik/Naturwissenschaft der Mittelstufe oder im Rahmen einer Projektwoche eingesetzt werden. Es eignet sich auch als Einführung in detailliertere Untersuchungen zu Optimierungsfragestellungen und entsprechenden Algorithmen in einem Projektkurs der Oberstufe.

Der Einstieg in das Thema erfordert etwa vier Unterrichtsstunden und kann je nach vorhandener Zeit und Interesse in verschiedener Weise ausgebaut werden. Hierzu gibt es unter [2]-[5] eine Fülle von Anregungen und Material.

Die Entwicklung dieses Unterrichtsmoduls wurde unterstützt durch eine Projektförderung durch die WestLB-Stiftung Zukunft NRW, für die wir uns ganz herzlich bedanken.

2 Die einführende Problemstellung

Anhand einer konkreten Problemstellung sollen die Schüler/innen versuchen, selbstständig eine möglichst gute Lösung für das sogenannte *Travelling-Salesman-Problem* (TSP) zu erarbeiten. Die Schüler/innen erhalten keine Vorgaben und keine Vorinformationen zu möglichen Lösungsansätzen, sondern sie sollen sich völlig frei mit dieser für sie neuartigen Problemstellung auseinandersetzen. Allgemein lässt sich dieses Problem folgendermaßen formulieren:

Das Travelling-Salesman-Problem: *Gegeben seien n Städte und die Entfernungen zwischen je zweien dieser Städte. Wir starten in einer Stadt und besuchen alle Städte genau einmal. Am Ende wollen wir zum Startort zurückkehren. Die Aufgabe besteht darin, eine möglichst kurze Rundreise durch diese n Städte zu finden.*

Mathematisch gesehen ist dies ein realistisches und schwieriges Problem, das exakt nur schwer zu lösen ist. Es gehört zu der Klasse der sogenannten *NP-vollständigen Probleme*, die alle folgende (und weitere) charakteristische Eigenschaften aufweisen (vgl. [1]):

- Sie haben eine Lösung; die Zeit zur Berechnung dieser Lösung wächst aber exponentiell mit der Zahl der Unbekannten (beim TSP mit der Zahl der Städte).
- Für kein Problem ist ein Algorithmus bekannt, der eine Lösung in einer Zeit findet, die nur polynomial mit der Anzahl der Unbekannten wächst.
- Es ist unklar, ob überhaupt Lösungsalgorithmen für diese Probleme existieren, deren Rechenaufwand nur polynomial mit der Anzahl der Unbekannten wächst.
- Zwar ist es bei allen Problemen schwierig, eine Lösung zu finden. Die Überprüfung, ob ein Lösungskandidat tatsächlich die richtige Lösung ist, ist aber in polynomialem Zeitaufwand möglich.

NP-vollständige Probleme kommen in der Praxis relativ häufig vor. Ein weiteres Beispiel für ein derartiges Problem, das im Schulalltag immer wieder auftaucht, ist auch das *Stundenplanproblem*.

Optimale Lösungen findet man (theoretisch) mit exponentiell wachsendem Rechen- oder Zeitaufwand. Für die Praxis sind derartige Algorithmen jedoch leider im Allgemeinen nicht brauchbar, weil die Rechenzeiten sehr schnell ins Astronomische wachsen (vgl. Abschnitt 4). Daher muss man sich mit heuristischen Lösungsansätzen und Algorithmen zufrieden geben, die eine möglichst gute Lösung in möglichst kurzer Zeit liefern.

Derartige auf heuristischen Überlegungen basierende Algorithmen werden in aller Regel auch die Schüler/innen entdecken.

3 Einstieg ins Thema mit einem Arbeitsblatt

Das *Arbeitsblatt: Das Rheinlandproblem* dient zum selbständigen Einstieg der Schüler/innen ins Thema. Die konkrete Problemstellung ist aus [2] übernommen. Hier heißt es auch: "*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.*"

Dieses Arbeitsblatt besteht aus zwei Teilen. Zunächst soll sichergestellt werden, dass die Schüler/innen eine Entfernungstabelle korrekt lesen können. Hierzu dienen die beiden einfachen einführenden Aufgaben, die auf jeden Fall besprochen werden sollten, bevor man an die eigentlich interessante Aufgabenstellung (*Das Rheinlandproblem*) herangeht.

Lösungen der einführenden Aufgaben auf dem Arbeitsblatt:

1. (a) 91 km (b) 91 km (c) 236 km (d) 236 km
2. Die Entfernungstabelle ist symmetrisch.

Für die Aufgabe *Rheinlandproblem* benötigen die Schüler/innen mehr Zeit (etwa 15 - 20 min, teilweise wünschen sich die Schüler/innen noch mehr Zeit). Danach geht es darum, die von den Schüler/innen erarbeiteten Vorgehensweisen zusammenzutragen. Dazu berichten *alle* Schüler/innen, wie sie ihre Lösung gefunden haben. Dabei ist Wert darauf zu legen, dass die Vorgehensweisen *für alle* auch nachvollziehbar dargestellt werden. Dieses Zusammentragen der von den Schüler/innen entwickelten Ansätze und Algorithmen ist ein ganz wesentlicher Bestandteil der Arbeit mit diesem Arbeitsblatt und macht die Vielfalt unterschiedlicher Lösungsansätze deutlich. Es wird auch deutlich, dass dieses Problem zwar einfach zu verstehen, aber nicht einfach zu lösen ist. Wir gehen in Abschnitt 3.1 auf Erfahrungen, die wir mit unterschiedlichen Schülergruppen bei der Bearbeitung dieses Arbeitsblatts gemacht haben, näher ein.

Der Einstieg mit dem Arbeitsblatt*Rheinlandproblem* kommt bei den Schüler/innen erfahrungsgemäß sehr gut an. Das Problem ist für jeden verständlich, und auch Schüler/innen, die in anderen Bereichen der Mathematik Defizite aufweisen, können sich mit der Problemstellung erfolgreich auseinandersetzen.

Hierzu trägt bei, dass sich alle Schüler/innen selbständig mit einer neuartigen Problemstellung beschäftigen. Dies geschieht ohne Vorgaben. Gefragt sind individuelle, freie Lösungen und die Kreativität der Schüler/innen.

Die Schüler/innen haben dabei ein doppeltes Erfolgserlebnis:

1. Alle Schüler/innen finden eine (in der Regel nicht die optimale) Rundreise.
2. Beim Erarbeiten dieser Rundreise entwickeln die Schüler/innen selbständig einen (heuristischen) Algorithmus zur Lösung des Problems. Dadurch beweisen sie (auch sich selbst), dass sie über ein gewisses mathematisches Talent verfügen.

Arbeitsblatt: Das Rheinlandproblem

Die folgende Tabelle enthält die direkten Entfernungen der 6 Städte Aachen, Bonn, Düsseldorf, Frankfurt, Köln und Wuppertal untereinander (in km). Wenn man z.B. von Aachen (Bonn, Düsseldorf, ...) aus in eine der Städte reisen möchte, so sind die entsprechenden Entfernungen in der ersten (zweiten, dritten, ...) Zeile der Tabelle enthalten. Will man *nach* Aachen (Bonn, Düsseldorf, ...) reisen, so kann man die entsprechenden Entfernungen aus der ersten (zweiten, dritten, ...) *Spalte* der Tabelle entnehmen.

	Aachen	**Bonn**	**Düsseldorf**	**Frankfurt**	**Köln**	**Wuppertal**
Aachen	0	91	80	259	70	121
Bonn	91	0	77	175	27	84
Düsseldorf	80	77	0	232	47	29
Frankfurt	259	175	232	0	189	236
Köln	70	27	47	189	0	56
Wuppertal	121	84	29	236	56	0

Einführende Aufgaben:

1. Wie groß ist die Entfernung von
   (a) Bonn nach Aachen
   (b) Aachen nach Bonn
   (c) Frankfurt nach Wuppertal
   (d) Wuppertal nach Frankfurt
2. Wenn für alle direkten Wege zwischen zwei Städten Hin- und Rückweg gleich lang sind, sagt man auch, die Entfernungstabelle ist symmetrisch. Ist die obige Entfernungstabelle symmetrisch?

Aufgabe *Rheinlandproblem*: Wir wollen eine Rundreise durch die 6 Städte Aachen, Bonn, Düsseldorf, Frankfurt, Köln und Wuppertal unternehmen. Aufgrund der obigen Tabelle kennen wir die Entfernungen aller 6 Städte voneinander.

Suche eine möglichst kurze Rundreise und beobachte gleichzeitig ganz genau, wie du dabei vorgehst. Formuliere deine Vorgehensweise und deine Überlegungen schriftlich.

3.1 Erfahrungen mit dem Arbeitsblatt *Rheinlandproblem* in Schülergruppen

Wir haben das Rheinlandproblem mehrfach mit Klassen der Stufe 9, mit Oberstufenkursen und auch mit heterogenen Schülergruppen der Jahrgänge 9 bis 13 als Einstieg in die Thematik des TSP eingesetzt. Natürlich sind die Schüler/innen vielfach daran interessiert, einen möglichst kurzen Weg zu finden, und sie vergleichen ihre Wege und Ansätze mit denjenigen von Klassenkameraden. Außerdem wollen sie verstehen, was an ihrem Ansatz eventuell nicht optimal ist.

Die kürzeste Rundreise hat beim Rheinlandproblem eine Länge von 617 km, z.B.

– Aachen – Düsseldorf – Wuppertal – Frankfurt – Bonn – Köln – Aachen –

Nicht in jeder Klasse wird eine optimale Lösung gefunden werden. Manchmal finden Schüler/innen auch Rundfahrten von 619 km, 625 km oder 627 km Länge, die fast optimal sind. Gute Rundfahrten sind auch diejenigen mit einer Länge von 648 km, 652 km oder 654 km. Die längsten Rundreisen sind bei diesem Problem übrigens 777 km lang.

Der Nearest-Neighbor-Algorithmus, der recht häufig von den Schüler/innen "entdeckt" wird, liefert – je nach Wahl des Startortes – Rundreisen von 658 (Start: Bonn oder Frankfurt), 689 (Start: Wuppertal), 693 (Start: Düsseldorf) 698 (Start: Aachen) oder 702 (Start: Köln) km Länge (vgl. auch Abschnitt 7.2.1).

Bemerkung: Man muss unterscheiden zwischen der Qualität des Algorithmus und der Länge der gefundenen Rundfahrt. Hat also jemand eine sehr kurze, eventuell auch eine optimale Rundfahrt gefunden, so sagt dies noch nichts über die Qualität des eingesetzten Algorithmus aus! So kann ein Algorithmus, der z.B. beim Rheinlandproblem bei der Wahl eines ungünstigen Startortes zu einer relativ langen Rundreise führt, trotzdem in vielen Fällen besser sein als ein Algorithmus, der hier zu einer recht kurzen Rundreise führt. Auf das Problem der Bewertung von Algorithmen kommen wir später noch zu sprechen (vgl. Abschnitt 5).

Viel interessanter als die Länge der von den Schüler/innen gefundenen kürzesten Rundfahrt ist die beeindruckende Vielfalt an Ansätzen und Algorithmen, welche die Schüler/innen in dieser kurzen Zeit entwickeln. Wir stellen einige unserer Erfahrungen im Folgenden kurz vor:

- Erfahrungsgemäß werden ganz unterschiedliche Algorithmen entwickelt.
- Mehrere Schüler/innen werden Varianten des Nearest-Neighbor-Algorithmus entdeckt haben: Man startet in einer Stadt (oder mit der kürzesten vorkommenden Städteverbindung) und geht dann in die nächstgelegene Stadt, von dort aus in die nächstgelegene noch nicht besuchte Stadt usw. und kehrt in die Ausgangsstadt zurück, wenn alle Städte besucht sind.
- Selbst bei demselben Nearest-Neighbor-Algorithmus werden die Schüler/innen in der Regel unterschiedlich lange Rundtouren herausbekommen haben. Vergleicht man

identische algorithmische Ansätze, so wird man dann feststellen, dass der einzige Unterschied in der Wahl der Startorte besteht. Hieraus können die Schüler/innen mehrere Schlüsse ziehen:

  - Offenbar hängt die Länge der gefundenen Rundtour vom gewählten Startort *für den Algorithmus* ab.
    Zur Verdeutlichung: Hat man einmal eine Rundtour gefunden, so ändert sich die Länge der Rundtour nicht, solange man zwar den Startort ändert, ansonsten die Städte aber in der gleichen Reihenfolge besucht (bei 5 Städten A, B, C, D und E sind etwa die Rundtouren ABCDEA und CDEABC gleich lang, was man sofort sieht, wenn man sich dies anhand einer Skizze veranschaulicht). Hiervon zu unterscheiden sind aber die Rundtouren, die man mit einem bestimmten Algorithmus erhält. So ist die Länge der Rundtour, die man mit dem Nearest-Neighbor-Algorithmus erhält, sehr wohl vom für den Algorithmus gewählten Startort abhängig.
  - Wenn die Länge der Route vom Startort abhängt, so liefert der Algorithmus offensichtlich nicht automatisch die kürzeste, möglicherweise aber eine akzeptable Route. Der Nearest-Neighbor-Algorithmus liefert also nicht automatisch die optimale Lösung des Problems. Auch dies ist ein Ergebnis der Arbeit der Schüler/innen mit dem Arbeitsblatt.

- Hinsichtlich der Wahl des Startortes können Schüler/innen unterschiedliche Teil-Algorithmen entwickelt haben, z.B.:

  - Man kann, wie schon oben erwähnt, mit der kürzesten vorkommenden Verbindung (hier: Köln–Bonn) starten.
  - Man kann von der Stadt aus starten, die am weitesten von den anderen entfernt ist (damit man am Ende, wenn nur noch wenige Städte übrig sind, nicht zu einer ungünstigen Wahl gezwungen ist).
  - Man kann in einer Stadt starten, in der die Unterschiede zwischen den kürzesten und den längsten Abständen zu anderen Städten möglichst groß sind (weil man hier am Ende, wenn nur noch wenige Städte übrig sind, am meisten verlieren könnte).

- Manche Schüler/innen entdecken auch den doppelten Nearest-Neighbor-Algorithmus: Ausgehend von einer Startstadt sucht man die *beiden* nächstgelegenen Städte und verbindet sie mit der Startstadt. Dann sucht man an beiden Enden der bisher gefundenen Tour die nächstgelegenen Nachbarn und fügt sie der Tour hinzu. Diesen Prozess wiederholt man, bis die Tour geschlossen ist.

- Häufig ergänzen Schüler/innen auch den einfachen oder doppelten Nearest-Neighbor-Algorithmus um lokale Optimierungsüberlegungen.

- Andere Schüler/innen versuchen, von kürzeren Rundreisen auszugehen, und dann die gefundene Rundreise sukzessive um eine weitere Stadt zu erweitern. Dabei entwickeln die Schüler/innen auch hier verschiedene Startstrategien:

  - Man startet mit drei beliebigen Städten.

  - Man startet mit drei Städten, die nahe beieinander liegen.
  - Man startet mit drei Städten, die möglichst weit voneinander entfernt liegen, denn diese müssen ja auch in die Rundreise integriert werden.

  In allen Fällen werden die weiteren Städte dann nacheinander in die aus drei Städten bestehende Startrundreise so eingebunden, dass die jeweils entstehende Rundreise möglichst kurz wird.

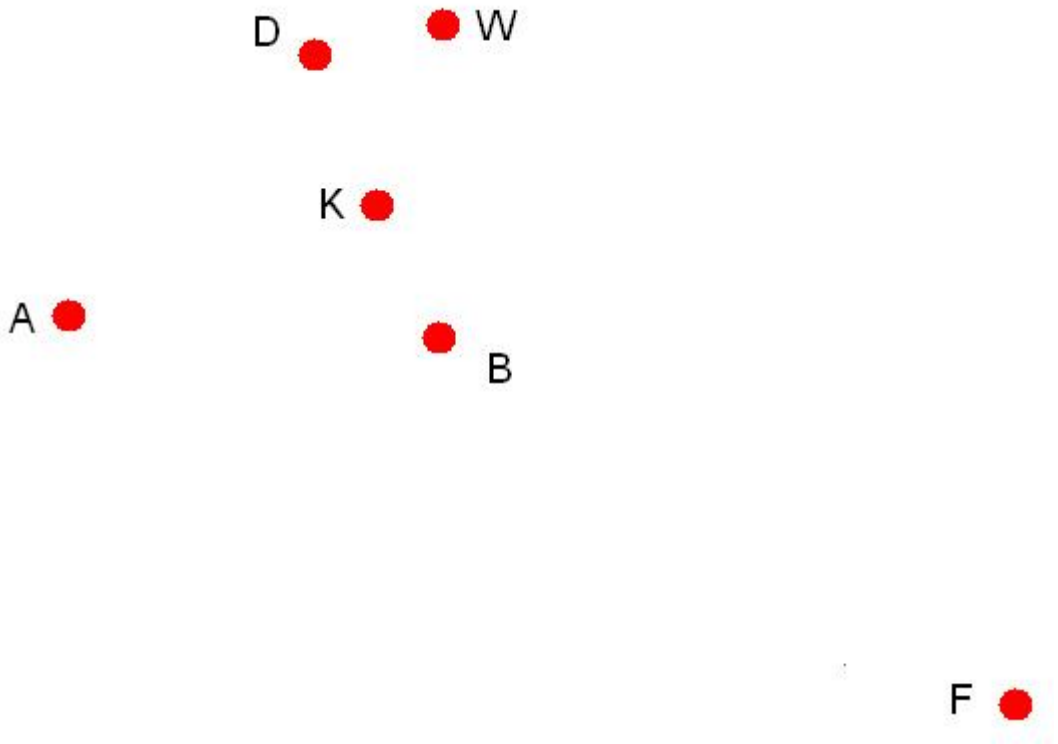
- Sollten einige Schüler/innen versuchen, alle Möglichkeiten auszuprobieren, so werden sie feststellen, dass hierfür die Zeit nicht reicht.
- Manche Schüler/innen verfolgen einen grafischen Ansatz. Sie haben die geografische Lage der Städte ungefähr im Kopf und skizzieren sich zunächst eine Landkarte, auf der sie die Position der Städte eintragen. Dann versuchen sie grafisch, eine möglichst kurze Rundreise zu finden.

Man kann den Schüler/innen ***nach*** der Bearbeitung des Arbeitsblattes *Das Rheinlandproblem* auch noch eine Karte mit den Positionen der Städte zur Verfügung stellen und ihnen aufgeben, auch grafisch eine Lösung des Problems zu entwickeln (vgl. das nachfolgende Arbeitsblatt *Das Rheinlandproblem, grafische Lösung*). Nach unserer Erfahrung kommen einige Schüler/innen mit dem grafischen Ansatz zu besseren Lösungen, andere jedoch zu schlechteren.

In den beigefügten Unterrichtsmaterialien befindet sich auch ein Computerprogramm, welches zwei unterschiedliche algorithmische Ansätze auf das Rheinlandproblem anwendet (vgl. Abschnitt 7.2.1).

Arbeitsblatt: Das Rheinlandproblem, grafische Lösung

Versuche die in der nachfolgenden Skizze durch Kreise markierten 6 Städte Aachen, Bonn, Düsseldorf, Frankfurt, Köln und Wuppertal so miteinander zu verbinden, dass eine möglichst kurze Rundreise entsteht. Erläutere, wie du dabei vorgehst.



Ermittle danach anhand der folgenden Tabelle, welche die direkten Entfernungen dieser 6 Städte untereinander (in km) enthält, die Länge der gefundenen Rundreise.

	Aachen	**Bonn**	**Düsseldorf**	**Frankfurt**	**Köln**	**Wuppertal**
Aachen	0	91	80	259	70	121
Bonn	91	0	77	175	27	84
Düsseldorf	80	77	0	232	47	29
Frankfurt	259	175	232	0	189	236
Köln	70	27	47	189	0	56
Wuppertal	121	84	29	236	56	0

4 Das Ausprobieren aller Möglichkeiten

Auf den ersten Blick scheint die Lösung des Travelling-Salesman-Problems kein grundsätzliches Problem zu sein. Die Schüler/innen könnten vermuten, dass es gerade vor dem Hintergrund seit Jahrzehnten ständig steigender Computerleistungen möglich sein sollte, die Längen aller Rundreisen vom Computer berechnen zu lassen.

Diese Idee berücksichtigt aber nicht das rasante Anwachsen aller Rundreisemöglichkeiten mit der Anzahl der Städte. Die Menge aller Rundreisen umfasst nämlich bei n Städten $n!$ Möglichkeiten. Bei drei Städten A, B und C sieht man dies durch einfaches Ausprobieren:
A B C
A C B
B A C
B C A
C A B
C B A
Dies sind $6 = 3!$ Möglichkeiten. Kommt jetzt eine vierte Stadt D hinzu, so kann man sie vor die erste, vor die zweite, vor die dritte und nach der dritten Stadt positionieren und erhält so viermal so viele, also $4!$ verschiedene Rundreisen. Entsprechend argumentiert man, wenn weitere Städte hinzu kommen.

Was dies für die Rechenzeiten eines Programms bedeutet, kann man sich leicht anhand folgender Überlegung klarmachen: Nehmen wir an, ein schnelles Computerprogramm benötige bei $n = 8$ Städten für das Ausprobieren aller Möglichkeiten lediglich 1 msec$= 10^{-3}$ sec. Dann können wir die Rechenzeiten für eine größere Anzahl von Städten leicht schätzen. Beim Übergang von n auf $n+1$ Städte erhöht sich die Anzahl der Möglichkeiten von $n!$ auf $(n+1)!$, also um den Faktor $n+1$. Entsprechendes können wir auch für die Rechenzeiten eines einfachen Programms erwarten (vgl. Tab. 4.1).

n	Rechenzeit
8	0,001 sec
9	0,009 sec
10	0,09 sec
11	0,99 sec
12	> 10 sec
13	> 2 min
14	>28 min
15	>7 h
16	>4,5 Tage
17	>2 Monate
18	>3 Jahre

Tabelle 4.1: Geschätzte Rechenzeiten beim Ausprobieren aller Möglichkeiten, wenn dies bei 8 Städten lediglich 1 msec an Zeit erfordern würde.

Diese Tabelle macht deutlich: Die Rechenzeiten explodieren geradezu. Daher ist dieser Ansatz nur bei Problemen mit sehr kleinem n praktikabel. Für größere Werte von n sind andere Ansätze notwendig.

Bis heute ist allerdings kein Algorithmus bekannt, der die optimale Rundreise in einer Laufzeit ermitteln kann, die nur polynomial mit n wächst; mit anderen Worten: bei allen bekannten Algorithmen, die eine optimale Rundreise finden, wächst die Laufzeit exponentiell, d.h. sie sind nur für relativ kleine n geeignet. Es ist ferner unklar, ob überhaupt ein Algorithmus mit polynomialer Laufzeit existiert, der immer das Optimum findet.

Daher verwendet man in der Praxis häufig Algorithmen, die auf unterschiedlichen Heuristiken basieren und für die Vielfalt der unterschiedlichen Problemstellungen jeweils möglichst kurze Rundreisen finden. Derartige heuristische Ansätze haben auch die Schüler/innen mit dem Aufgabenblatt *Rheinlandproblem* entwickelt.

Für das Rheinlandproblem mit $n = 6$ kann man noch problemlos alle Möglichkeiten ausprobieren (vgl. hierzu Abschnitt 7.2.1). Dabei ist es nicht erforderlich, die erste Stadt, mit der man die Rundreise beginnt, zu variieren. Hat man alle verschiedenen Rundreisen bei ein und derselben Startstadt, so erhält man hieraus alle Rundreisen mit allen möglichen Startstädten durch zyklische Vertauschung. Eine zyklische Vertauschung der Städte ändert natürlich die Länge der Rundreise nicht.

5 Bewertung von unterschiedlichen Algorithmen

Die optimale Lösung des TSP kann man zwar im Prinzip durch das Ausprobieren aller Möglichkeiten finden, aber der Aufwand hierfür wird, wie wir in Abschnitt 4 gesehen haben, schnell so groß, dass dieser Ansatz nicht weiterhilft. Man braucht also viel schnellere Lösungsansätze. Dies zeigt auch, dass wir neben der Qualität der von einem Algorithmus gelieferten Rundreisen auch den erforderlichen Rechenaufwand mit berücksichtigen müssen, der wiederum auch von der Größe der Problemstellung abhängt.

Die Bewertung eines heuristischen Algorithmus ist nicht einfach, da die gefundene Qualität der Lösung häufig von der konkreten Problemstellung abhängt. So können auch Algorithmen, die häufig relativ schlechte Ergebnisse erzielen, in einem Spezialfall eine optimale Lösung finden.

Der Nearest-Neighbor Algorithmus ist viel schneller als das Ausprobieren aller Möglichkeiten, liefert aber in der Regel nicht die optimale Lösung. Die Abweichung vom Optimum kann sogar sehr groß werden. Sein Rechenaufwand wächst proportional zu n^2 (bzw. n^3 , wenn man zusätzlich unterschiedliche Startstädte berücksichtigt).

Es gibt auch heuristische Algorithmen, für die man Qualitätsgarantien abgeben kann: Die *Spanning-Tree-Heuristik* liefert eine Rundtour, die höchstens um 100 % länger sein kann als die optimale Tour. Diese Schranke wird durch die *Christofides-Heuristik* auf 50 % reduziert. Der Rechenaufwand für die Christofides-Heuristik wächst proportional zu n^3. Eine Beschreibung dieser beiden heuristischen Algorithmen findet man z.B. in [2] oder in [6].

6 Potentielle Erweiterungen dieses Unterrichtsmoduls

Man kann im Unterricht natürlich auch auf die Problematik der Bewertung von heuristischen Algorithmen eingehen. Die Spanning-Tree- und die Christofides-Heuristik mit ihren Qualitätsgarantien [2, 6] oder auch die k-Opt-Heuristiken [7] stellen weitere mögliche Erweiterungen zur Lösung des TSP dar. Bei den k-Opt-Heuristiken geht man von irgendeiner Rundtour aus und versucht diese zu verbessern. Im einfachsten Fall ($k = 2$) werden zwei Kanten aus der Rundtour entfernt und kreuzweise wieder eingeführt. Ist die neue Tour kürzer als die alte, so führt man diesen Prozess mit der neuen Tour fort. Ist die Tour länger, versucht man es mit zwei anderen Kanten.

Eine andere Möglichkeit ist, die Problemstellung dahingehend zu verändern, dass man nicht mehr eine Rundtour sucht, sondern eine Verbindung von einer Startstadt zu einer Zielstadt, wobei dazwischen eine Reihe weiterer Städte besucht werden muss. Auch hier ist dann wieder eine optimale Strecke gesucht. Dieses Problem ist viel einfacher zu lösen als das TSP. Hierzu könnte z.B. der Dijkstra-Algorithmus eingeführt werden [8]. Andere hiermit verwandte Verfahren sind Breiten- und Tiefensuche.

Im Bereich der kombinatorischen Optimierung gibt es eine Vielfalt weiterer interessanter Problemstellungen, die für den Einsatz im Schulunterricht geeignet und in [3, 4, 5] ausführlich beschrieben sind.

7 Unterrichtsmaterialien

Neben einigen Arbeitsblättern sind diesem Unterrichtsmodul auch einige Computerprogramme beigefügt, die von der Lehrkraft zu Demozwecken oder von den Schüler/innen zum Ausprobieren oder Analysieren verwendet werden können. Die vorliegende Unterrichtseinheit kommt jedoch auch völlig ohne diese Computerprogramme aus und ist in keiner Weise hiervon abhängig.

Beginnend mit den Computerprogrammen, werden die Materialien im Folgenden dargestellt und erläutert.

7.1 Folien

Die Dateien rheinland.ppt und rheinland.pdf enthalten jeweils eine Folie zum Rheinland-Problem (als Powerpoint- oder pdf-Datei), die während der Besprechung des zugehörigen Arbeitsblattes die Entfernungsmatrix für alle sichtbar macht.

7.2 Computerprogramme

Alle beigefügten Computerprogramme zu diesem Modul sind in der Programmiersprache Python geschrieben (Versionen 2.6.x oder 2.7.x) und wurden im September 2012 vollständig neu geschrieben. Sie sind jetzt wesentlich besser strukturiert und auch deutlich schneller als die früheren Versionen, die noch von Schüler/innen entwickelt worden waren.

Python ist frei verfügbar und kann über die Web-Seite www.python.org frei heruntergeladen werden. Dort finden sich auch entsprechende Tutorials etc.

Python wurde mit dem Ziel entwickelt, das Programmieren für den Programmierer möglichst einfach zu machen. Daher kann man Python mit relativ wenig Aufwand so weit lernen, dass man alle in der Schule vorkommenden Algorithmen problemlos programmieren kann. Wir haben dies mehrfach mit (heterogenen) Gruppen von Schülerpraktikanten der Klassen 9 - 13 erprobt und sehr gute Erfahrungen mit einer zwei- bis dreitägigen Einführung in Python gemacht.

Auf Wunsch können wir auch entsprechendes Material zur Verfügung stellen.

Python ist nicht die schnellste Programmiersprache und nach unseren Erfahrungen in der Ausführung etwa einen Faktor 30 langsamer als entsprechende optimierte C-Programme. Der Aufwand, C oder eine vergleichbare Programmiersprache zu lernen oder einen komplizierten Algorithmus in C zu programmieren, ist dafür aber wesentlich höher als bei Python.

7.2.1 Computerprogramm: Ausprobieren aller Möglichkeiten und Nearest-Neighbor-Algorithmus für das Rheinlandproblem

Das Python-Programm rheinland.py kann sowohl von der Lehrkraft zu Demozwecken als auch von den Schüler/innen eingesetzt werden, um die Ergebnisse des Ausprobierens aller Möglichkeiten bzw. des Nearest-Neigbor-Algorithmus bei unterschiedlicher Wahl des Startortes im Detail zu analysieren. Die Entfernungstabelle des Rheinlandproblems wird vom Programm über die Datei rundreise_input_6.txt eingelesen.

Im Python-Programm rheinland.py sind zwei verschiedene Algorithmen zur Lösung des Rheinlandproblems implementiert. Der Einfachheit halber sind die Städte Aachen, Bonn, Düsseldorf, Frankfurt, Köln, Wuppertal in dieser Reihenfolge mit den Ziffern 0 bis 5 codiert.

1. Beim Ausprobieren aller Möglichkeiten werden die Längen aller möglichen Rundreisen mit dem Startort 0 (Aachen) berechnet und die Rundreisen mitsamt den Längen ausgegeben. Da man hieraus alle anderen Rundreisen mit anderen Startstädten durch zyklische Vertauschung gewinnen kann und sich bei einer zyklischen Vertauschung der Städte die Länge der Rundreise nicht ändert, findet man auf diese Weise eine kürzeste Rundreise.

2. Beim Nearest-Neighbor-Algorithmus werden hingegen in diesem Programm die

Startstädte variiert. Die Ergebnisse zeigen, dass die Länge der vom Algorithmus gefundenen Rundreise vom Startort abhängt. Für das Rheinlandproblem findet der Nearest-Neighbor-Algorithmus für keinen Startort die optimale Rundreise.

7.2.2 Computerprogramm zum Ausprobieren aller Möglichkeiten

Das Python-Programm tsp_allerouten.py ermittelt für $2 < n < 14$ Städte eine möglichst kurze Rundreise durch Ausprobieren aller Möglichkeiten. Es kann von der Lehrkraft und/oder von den Schüler/innen zur systematischen Analyse der Rechenzeiten des Ausprobierens aller Möglichkeiten bei steigender Anzahl n von Städten eingesetzt werden. Mit diesem Programm kann die Lehrkraft demonstrieren oder die Schüler/innen können selbst herausfinden, wie schnell der Rechenaufwand bei diesem Algorithmus der Komplexität $(n-1)!$ anwächst. (Die Anzahl aller Möglichkeiten beträgt in diesem Programm nur $(n-1)!$, weil in diesem Programm die erste Stadt der Rundreise nicht variiert wird; dies ist sinnvoll, weil die Länge einer Rundreise nicht davon abhängt, in welcher Stadt man startet.) Dabei ist für die Schüler/innen sehr überraschend, dass die Rechenzeiten schon für $n < 15$ explodieren.

Zugrundegelegt wird eine Entfernungstabelle der 40 größten deutschen Städte, die über die Datei rundreise_input_40.txt eingelesen wird. Die Eingabe wurde auf Werte mit $n < 14$ eingeschränkt, weil die Rechenzeiten auf üblichen PCs sonst weit in den Stundenbereich gehen. Für $n = 13$ liegen sie – je nach Leistungsfähigkeit des Rechners – schon in der Größenordnung einer halben bis zu einer Stunde oder mehr. Diese Eingabe sprengt in aller Regel den Einsatz in einer Unterrichtsstunde. Hierauf sollte die Lehrkraft explizit verweisen.

Will man größere Werte von n zulassen, reicht es, in dem Befehl
`while n > 13 or n < 2:`
die Zahl 13 durch eine entsprechend höhere zu ersetzen und bei der vom Programm verlangten Eingabe die gewünschte höhere Zahl einzugeben.

Lässt man dieses Programm für $2 \leq n \leq 12$ laufen, so können die ausgegebenen Rechenzeiten gerundet z.B. folgendermaßen aussehen:

n	Rechenzeit (sec)
≤ 7	0,00
8	0,02
9	0,13
10	1,16
11	11,59
12	128,68

Was wäre für $n = 13$ zu erwarten? Antwort: Für $n = 12$ hat man etwa 129 sec, also mehr als 2 Minuten Rechenzeit; für $n = 13$ sind etwa 12-mal so viel, d.h. mehr als 24 Minuten zu erwarten, und für $n = 14$ erhöht sich die Rechenzeit bereits auf über 5 Stunden! Dies macht unmittelbar deutlich, dass dieser Ansatz für größere Anzahlen von Städten nicht praktikabel ist.

7.2.3 Computerprogramm zum Nearest-Neighbor-Algorithmus mit Variation der Startstädte

Das Python-Programm tsp_nn_mit_startvariation.py wendet den Nearest-Neigbor-Algorithmus an, um das Travelling Salesman Problem zu lösen. In diesem Programm variiert der Algorithmus auch die Startstädte. Sein Rechenaufwand ist daher proportional zu n^3.

Dieses Programm kann von der Lehrkraft und/oder von den Schüler/innen zur systematischen Analyse der erforderlichen Rechenzeiten bei steigender Anzahl n von Städten eingesetzt werden. Mit diesem Programm kann die Lehrkraft demonstrieren oder die Schüler/innen können selbst herausfinden, wie schnell der Rechenaufwand bei diesem Algorithmus der Komplexität n^3 anwächst. (Beispiel: Auf einem handelsüblichen PC misst das Programm mit $n = 200$ etwa 0,5 sec, mit $n = 400$ etwa 4 sec für den Nearest-Neighbor-Algorithmus mit Variation der Startstadt; die gesamte Laufzeit ist etwas höher, weil das Aufstellen der Entfernungstabelle mit Zufallszahlen ebenfalls Zeit kostet, die aber nicht mit gemessen wird.) Interessant ist natürlich auch der Vergleich der erforderlichen Rechenzeiten mit denen anderer Algorithmen. Dabei beachte man, dass dieser Algorithmus zwar sehr viel schneller als das Ausprobieren aller Möglichkeiten ist, aber man erhält auch nicht immer die optimale Rundreise.

Ist die Zahl n der zu besuchenden Städte kleiner oder gleich 40, so wird in diesem Programm eine Entfernungstabelle der 40 größten deutschen Städte, die über die Datei rundreise_input_40.txt eingelesen wird, zugrundegelegt. Für $n > 40$ wird die Entfernungstabelle mit zufällig ausgewählten Werten zwischen 1 und 500 belegt.

7.2.4 Computerprogramm zum Nearest-Neighbor-Algorithmus ohne Variation der Startstädte

Das Python-Programm tsp_nn_ohne_startvariation.py wendet den Nearest-Neigbor-Algorithmus an, um das Travelling Salesman Problem zu lösen. In diesem Programm variiert der Algorithmus die Startstädte nicht, sondern startet immer in der ersten Stadt (Stadt 0). Sein Rechenaufwand ist daher proportional zu n^2.

Dieses Programm kann von der Lehrkraft und/oder von den Schüler/innen zur systematischen Analyse der erforderlichen Rechenzeiten bei steigender Anzahl n von Städten eingesetzt werden. Mit diesem Programm kann die Lehrkraft demonstrieren oder die Schüler/innen können selbst herausfinden, wie schnell der Rechenaufwand bei diesem Algorithmus der Komplexität n^2 anwächst. (Beispiel: Auf einem handelsüblichen PC misst das Programm mit $n = 1000$ knapp 0,1 sec, mit $n = 2000$ etwa 0,3 sec für den Nearest-Neighbor-Algorithmus ohne Variation der Startstadt; die gesamte Laufzeit ist etwas höher, weil das Aufstellen der Entfernungstabelle mit Zufallszahlen ebenfalls Zeit kostet, die aber nicht mit gemessen wird.) Vergleicht man die erforderlichen Rechenzeiten auch mit anderen Ansätzen wie dem Ausprobieren aller Möglichkeiten oder dem Nearest-Neigbor-Algorithmus *mit* Variation der Startstädte, so sind hier die Rechenzeiten naturgemäß deutlich kleiner. Dies erkauft man jedoch mit einer in der Regel längeren Rundreise.

Ist die Zahl n der zu besuchenden Städte kleiner oder gleich 40, so wird auch in diesem Programm eine Entfernungstabelle der 40 größten deutschen Städte, die über die Datei rundreise_input_40.txt eingelesen wird, zugrundegelegt. Für $n > 40$ wird die Entfernungstabelle mit zufällig ausgewählten Werten zwischen 1 und 500 belegt.

7.2.5 Computerprogramme zu den k-Opt-Heuristiken

In den Computerprogrammen TSP_TwoOpt.py und TSP_TwoOpt_best.py sind zwei Varianten einer k-Opt-Heuristik implementiert (vgl. Abschnitt 6). Während in TSP_TwoOpt.py jedes Paar von Kanten akzeptiert wird, das eine kürzere Rundtour liefert, wird in TSP_TwoOpt_best.py nur jeweils das Kantenpaar ausgewählt, dass die größte Verbesserung liefert. In der Praxis führen diese Algorithmen zu zufriedenstellenden Ergebnissen, allerdings kann in Einzelfällen der Zeitaufwand sehr stark anwachsen. Man kann diese Programme z.B. mit verschiedenen Werten von n mit $2 \leq n \leq 40$ starten und die Rechenzeit sowie die Qualität der gefundenen Rundtouren systematisch mit denen vergleichen, die man mit den Nearest-Neighbor-Programmen erzielt.

7.3 Weitere Arbeitsblätter

Zwei Arbeitsblätter zum *Rheinlandproblem* wurden bereits in Abschnitt 3 vorgestellt. Ein weiteres Arbeitsblatt beschäftigt sich mit dem Ansatz, alle Möglichkeiten auszuprobieren.

Arbeitsblatt: Ausprobieren aller Möglichkeiten

Aufgaben:

1. Gegeben seien drei Städte A, B und C. Man will eine Rundreise durch diese drei Städte machen, indem man in einer Stadt startet, jede andere Stadt genau einmal besucht und dann in die Ausgangsstadt zurückkehrt. Rundreisen mit unterschiedlichem Startort gelten als verschieden.

   (a) Wie viele verschiedene derartige Rundreisen gibt es?

   (b) Ist es denkbar, dass diese Rundreisen unterschiedlich lang sind? (Tipp: Zeichne dir ein Beispiel für die Lage der Städte A, B und C und male die Rundreisen hinein.)

2. Gegeben seien vier Städte A, B, C und D. Man will eine Rundreise durch diese vier Städte machen, indem man in einer Stadt startet, jede andere Stadt genau einmal besucht und dann in die Ausgangsstadt zurückkehrt. Rundreisen mit unterschiedlichem Startort gelten als verschieden.

   (a) Wie viele verschiedene derartige Rundreisen gibt es?

   (b) Ist es denkbar, dass diese Rundreisen unterschiedlich lang sind? (Tipp: Zeichne dir ein Beispiel für die Lage der Städte A, B, C und D und male verschiedene Rundreisen hinein.)

   (c) Zum Knobeln: Wie viele verschiedene Längen von Rundreisen gibt es höchstens?

3. Gegeben seien fünf Städte A, B, C, D und E. Man will eine Rundreise durch diese fünf Städte machen, indem man in einer Stadt startet, jede andere Stadt genau einmal besucht und dann in die Ausgangsstadt zurückkehrt. Rundreisen mit unterschiedlichem Startort gelten als verschieden.

   (a) Wie viele verschiedene derartige Rundreisen gibt es?

   (b) Wie viele verschiedene Längen von Rundreisen gibt es höchstens?

4. Gegeben seien n Städte A_1, A_2, ..., A_n. Man will eine Rundreise durch diese n Städte machen, indem man in einer Stadt startet, jede andere Stadt genau einmal besucht und dann in die Ausgangsstadt zurückkehrt. Rundreisen mit unterschiedlichem Startort gelten als verschieden.

   (a) Wie viele verschiedene derartige Rundreisen gibt es?

   (b) Wie viele verschiedene Längen von Rundreisen gibt es höchstens?

Arbeitsblatt: Ausprobieren aller Möglichkeiten

Infos für Lehrkräfte:

Eine gute Ergänzung zu diesem Arbeitsblatt ist das Computerprogramm zum Ausprobieren aller Möglichkeiten (vgl. Abschnitt 7.2.2).

Bemerkung zu den Fragen des Arbeitsblatts, wie viele verschiedene Längen von Rundreisen es höchstens gibt: hier ist eigentlich eine obere Schranke für die Anzahl verschiedener Längen von Rundreisen gesucht. Obere Schranken sind aber nicht eindeutig festgelegt. Auch die Anzahl aller Rundreisen wäre jeweils eine vernünftige Antwort. Allerdings kann man diese obere Schranke durch zwei voneinander unabhängige Überlegungen reduzieren. Stellen Schüler/innen nur eine dieser beiden Überlegungen an, so stellt auch die daraus resultierende Antwort jeweils eine gute eigenständige Leistung dar.

Lösungen der Aufgaben:

1. 3 Städte A, B und C

   (a) Es gibt 6 verschiedene Rundreisen: ABC, ACB, BAC, BCA, CAB und CBA.

   (b) Die Rundreisen sind alle gleich lang. Sie unterscheiden sich nur durch den Startort bzw. durch die Richtung. Mit anderen Worten: Hat man eine Rundreise, so gehen alle anderen dadurch hervor, indem man den Startort und/oder die Richtung modifiziert.

2. 4 Städte A, B, C und D

   (a) Es gibt 24 verschiedene Rundreisen. Dies kann man auf verschiedene Art und Weise sehen:

   - Zum Beispiel kann ich von der vorherigen Aufgabe ausgehen und überlegen, an welche Positionen in den Rundreisen mit 3 Städten die vierte Stadt gelegt werden kann (vor der ersten, nach der ersten, nach der zweiten und nach der dritten Stadt). Dies sind 4 Positionen; man erhält also viermal so viele verschiedene Rundreisen.
   - Man kann auch wieder alle Möglichkeiten explizit hinschreiben und abzählen.

   (b) Es ist denkbar, dass diese Rundreisen unterschiedlich lang sind. In der Regel werden z.B. die Rundreisen ABCDA und ACBDA unterschiedlich lang sein. Sie können nicht auseinander hervorgehen, wenn man Startort oder Richtung der Rundreise ändert. Man kann sich dies auch geometrisch überlegen, etwa wenn man die vier Städte als Eckpunkte eines Quadrates mit Kantenlänge a wählt und die Längen dieser beiden Rundreisen berechnet.

   (c) Es gibt 24 verschiedene Rundreisen. Um herauszufinden, wie viele verschiedenen Längen von Rundreisen es höchstens gibt, kann man zwei sich ergänzende Überlegungen anstellen:

   - Zu jeder Rundreise gibt es drei weitere, bei denen die Städte in der gleichen Reihenfolge besucht werden und nur die Startstadt modifiziert wird

(zyklische Vertauschung der Städte). Daraus folgt, dass (mindestens) je vier derartige Rundreisen gleich lang sind.

- Man kann jede Rundreise auch in der entgegengesetzten Richtung durchfahren. Dies reduziert die Anzahl der Rundreisen unterschiedlicher Länge um einen Faktor 2.

Weil diese beiden Überlegungen unabhängig voneinander sind, kann ich immer Gruppen von 8 Rundreisen finden, die gleich lang sind. $24 : 8 = 3$ Es gibt also höchstens drei unterschiedliche Längen von Rundreisen.

3. 5 Städte A, B, C, D und E

   (a) Analog zur vorherigen Aufgabe kann man zeigen, dass es $5! = 120$ verschiedene Rundreisen gibt.

   (b) Analog zur vorherigen Aufgabe liefert die Variation der Startstadt hier einen Faktor 5, die Fahrtrichtung wieder einen Faktor 2. Es gibt also höchstens $120 : (5 \cdot 2) = 12$ unterschiedliche Längen von Rundreisen.

4. n Städte A_1, A_2, ..., A_n

   (a) Analog zur vorherigen Aufgabe kann man sich überlegen, dass es $n!$ Möglichkeiten gibt.

   (b) Analog zur vorherigen Aufgabe liefert die Variation der Startstadt hier einen Faktor n, die Fahrtrichtung wieder einen Faktor 2. Es gibt also höchstens $n! : (n \cdot 2) = (n-1)!/2$ unterschiedliche Längen von Rundreisen.

Literatur

[1] Horst Gierhardt. http://www.gierhardt.de/informatik/info13/TheorieIV.pdf

[2] Martin Grötschel. *Schnelle Rundreisen: Das Travelling-Salesman-Problem.* In [3], 95-129. Auch unter http://opus4.kobv.de/opus4-zib/files/890/ZR-05-57.pdf

[3] Stephan Hußmann, Brigitte Lutz-Westphal (Hrsg.). *Kombinatorische Optimierung erleben.* Vieweg, Wiesbaden, 2007.

[4] Brigitte Lutz-Westphal. *Die Mathematik der kürzesten Wege.* Inhalte und Methoden für den Unterricht, ZIB-Report 04-36, Konrad-Zuse-Institut, Berlin, 2004.

[5] Brigitte Lutz-Westphal. *Kombinatorische Optimierung – Inhalte und Methoden für einen authentischen Mathematikunterricht.* Dissertation, TU Berlin, 2006.

[6] http://de.wikipedia.org/wiki/Problem_des_Handlungsreisenden

[7] http://http://de.wikipedia.org/wiki/K-Opt-Heuristik

[8] http://de.wikipedia.org/wiki/Dijkstra-Algorithmus