

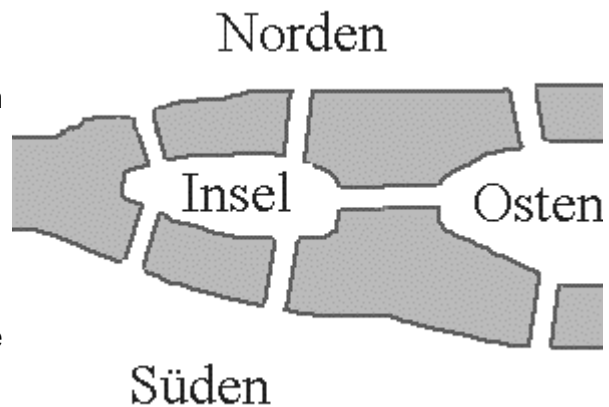
Praktische Grenzen der Berechenbarkeit

Während es im ersten Abschnitt um prinzipiell unlösbare Probleme ging, wenden wir uns nun Aufgaben zu, deren Lösbarkeit praktische Grenzen gesetzt sind. Diese Probleme sind zwar algorithmisch lösbar, aber zu ihrer Lösung sind noch keine effizienten Algorithmen entwickelt worden, d.h. bei Verwendung der bekannten Algorithmen steigt der Aufwand an Betriebsmitteln (Zeit und Speicherplatz) mit wachsender Problemgröße in unverträglicher Weise an. Für eine große Klasse derartiger Probleme ist nicht einmal bekannt, ob eine effiziente Lösung existieren könnte oder nicht.

Die Problemklassen P und NP

Beispiel: Das Königsberger Brückenproblem (L. Euler, 1736)

In der Innenstadt von Königsberg vereinigen sich der Alte Pregel und der Neue Pregel zum Fluss Pregel. Im 18. Jh. führten sieben Brücken über die Flüsse. Jeder Königsberger wusste, dass es keinen Weg gibt, der alle sieben Brücken genau einmal überquert und wieder zum Ausgangspunkt zurückführt, aber keiner hatte eine Erklärung dafür.

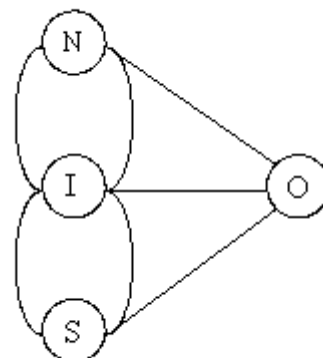


Leonhard Euler (1707 - 1782) verallgemeinerte die Problemstellung:

Gibt es in einem vorgegebenen Graphen einen Weg, der jede Kante des Graphen genau einmal enthält und zum Ausgangsknoten zurückführt (Eulerscher Kreis) ?

Die nächstliegende Lösung besteht in einem erschöpfenden Durchsuchen aller Kanten-Permutationen. Im Königsberger Brückenproblem gibt es $7! = 5040$ Permutationen der Kanten. Jede ist daraufhin zu untersuchen, ob sie zum Ausgangspunkt zurückführt. Da allgemein bei n Kanten $n!$ Permutationen zu untersuchen sind, ist der Zeitaufwand dieses Algorithmus von der Größenordnung $n!$.

Euler fand jedoch eine sehr effiziente Lösung: Ein Weg in einem Graphen, der jede Kante durchläuft und zum Ausgangspunkt zurückführt, muss jeden Knoten genauso oft verlassen, wie er ihn besucht. Soll dabei jede Kante genau einmal durchlaufen werden, so muss von jedem Knoten eine gerade Anzahl von Kanten ausgehen. Dies ist beim Königsberger Brückenproblem nicht der Fall.



Algorithmus: Eulerkreis

```
FÜR i := 1 BIS Knotenzahl WIEDERHOLE
  Kantenzahl [i] := 0;
ENDE_WIEDERHOLE
FÜR i := 1 BIS Knotenzahl WIEDERHOLE
  FÜR j := 1 BIS Knotenzahl WIEDERHOLE
    WENN Kante von i nach j existiert
      DANN Kantenzahl [i] := Kantenzahl [i] + 1;
    ENDE_WIEDERHOLE
  ENDE_WIEDERHOLE
i := 0;
WIEDERHOLE
  i := i+ 1;
  WENN Kantenzahl [i] gerade
    DANN Eulerkreis := TRUE
    SONST Eulerkreis := FALSE;
  BIS (NOT Eulerkreis) ODER (i = Kantenzahl);
  WENN Eulerkreis
    DANN Ausgabe : "Eulerkreis existiert"
    SONST Ausgabe : "Eulerkreis existiert nicht"
```

Wegen der beiden ineinandergeschachtelten Schleifen ist der Zeitaufwand dieses Algorithmus proportional zum Quadrat der Knotenzahl.

Zusammenfassung:

Als *praktisch durchführbar* gelten Algorithmen, die höchstens polynomialen Aufwand erfordern. Als *praktisch undurchführbar* gelten die Algorithmen mit mindestens exponentiellem Aufwand. Ein Problem p gehört zur Klasse P , wenn es einen praktisch durchführbaren Lösungsalgorithmus für P gibt.

Eine leichte Abwandlung der Fragestellung des führt zu einem viel schwierigeren Problem:

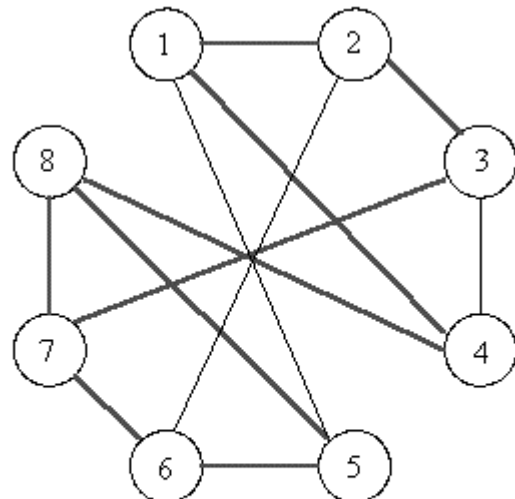
Beispiel : Das Hamilton-Problem

Ein *Hamilton-Zyklus* in einem Graphen ist ein Weg über die Kanten, der im selben Knoten beginnt und endet, und der genau einmal durch jeden Knoten führt.

Das Hamilton-Problem lautet:

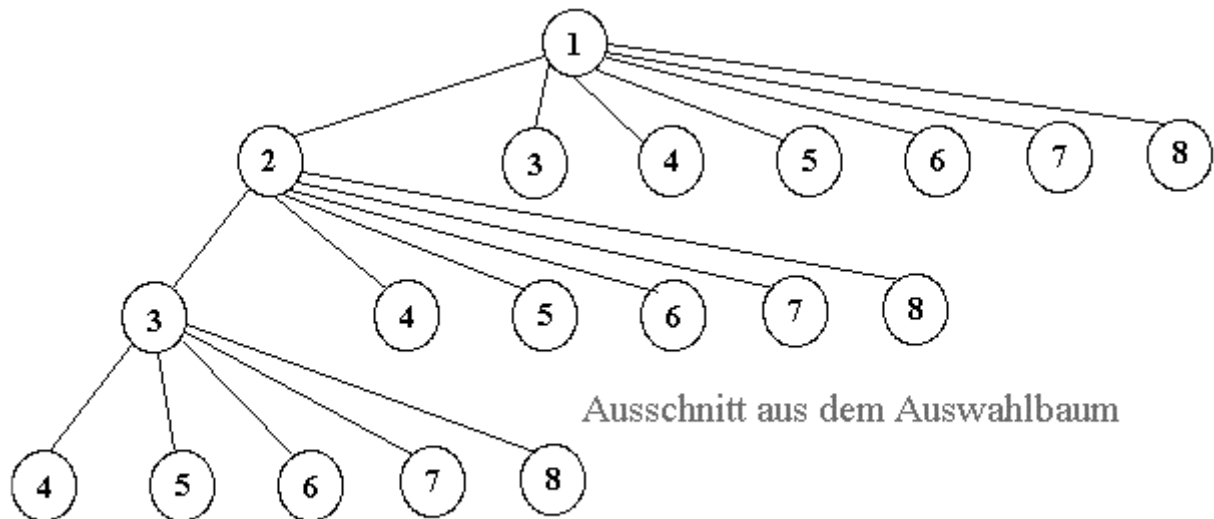
Für einen beliebigen Graphen ist zu entscheiden, ob er einen Hamilton-Zyklus enthält oder nicht.

Die Gesamtheit der möglichen Wege durch den Graphen lässt sich in einem Auswahlbaum darstellen, wobei jedem Weg ein Zweig entspricht. Die Tiefe des Auswahlbaumes ist dabei gleich der Knotenzahl n . Die Anzahl der zu prüfenden Wege beträgt $(n-1)!$



Hamilton-Zyklus:

1 - 2 - 3 - 7 - 6 - 5 - 8 - 4 - 1



Der folgende Algorithmus führt eine erschöpfende Suche im Auswahlbaum durch und hat damit einen Zeitaufwand der Größenordnung $(n-1)!$.

Algorithmus : Hamilton-Zyklus

Eingabe : Anzahl der Knoten n

Eingabe : $(n \times n)$ -Matrix M mit

$M[i,j] = 1$, wenn eine Kante von Knoten i zu Knoten j existiert

$M[i,j] = 0$, wenn keine Kante von Knoten i zu Knoten j existiert

FÜR $i := 1$ BIS n WIEDERHOLE Besucht[i] := FALSE

Besuchte Knoten := 0;

Besuche(1)

PROZEDUR Besuche(Knotenindex p)

Besuchte Knoten := Besuchte Knoten + 1

WENN Besuchte Knoten = n UND $M[p, 1] = 1$

DANN Ausgabe : "Hamilton-Zyklus existiert"

SONST

Besucht[p] := TRUE

FÜR $q := 1$ BIS n WIEDERHOLE

WENN $M[p, q] = 1$ DANN

WENN NICHT Besucht[q] DANN Besuche [q]

ENDE_WIEDERHOLE

Besucht[p] := FALSE;

Besuchte Knoten := Besuchte Knoten + 1

ENDE_SONST

Leicht ist es dagegen, eine vorgegebene Lösung zu verifizieren, da sich jeder Zweig des Auswahlbaumes in polynomialer Zeit durchlaufen lässt (Zeitaufwand proportional zur Knotenzahl n). Den gleichen geringen Zeitaufwand hätte folgender Algorithmus zur Lösung des Hamilton-Problems:

Algorithmus : Hamilton-Zyklus (2)

```
Eingabe : Anzahl der Knoten n
Eingabe : (n x n)-Matrix M mit
  M[i,j] = 1, wenn eine Kante von Knoten i zu Knoten j existiert
  M[i,j] = 0, wenn keine Kante von Knoten i zu Knoten j existiert
FÜR i := 1 BIS n WIEDERHOLE Besuchte[i] := FALSE
Besuchte Knoten := 0;
p := 1
FÜR i := 1 BIS n WIEDERHOLE
  Besuchte[p] := TRUE
  Wähle Knoten q mit M [p,q] = 1 UND Besuchte[q] = FALSE
  p := q
ENDE_WIEDERHOLE
WENN FÜR ALLE i := 1 BIS n Besuchte[i] := TRUE
  DANN Ausgabe : "Hamilton-Zyklus existiert"
  SONST Ausgabe : "Hamilton-Zyklus existiert nicht"
```

Die Anweisung "Wähle Knoten q ..." lässt offenbar die Wahl zwischen mehreren konkreten Anweisungen zu. Ein Algorithmus mit dieser Eigenschaft heißt *nichtdeterministisch*. Die Ausführung eines nichtdeterministischen Algorithmus kann man sich so vorstellen, dass an jeder Verzweigungsstelle - durch glückliches Raten bzw. kraft höherer Intuition - der richtige Weg gewählt wird. Voraussetzung für die Abarbeitung eines solchen Algorithmus wäre ein nichtdeterministisch arbeitender Computer, der eine unendlich große Zahl von Prozessoren besitzt. Bei jeder Verzweigung teilt sich die Menge der Prozessoren in so viele Teilmengen auf, wie es weiterführende Wege gibt.

Bezeichnung:

Ein Problem gehört zur *Klasse NP*, wenn es durch einen nichtdeterministischen Algorithmus mit polynomialem Zeitaufwand gelöst werden kann.

Anmerkungen:

- Weil jeder deterministische Algorithmus ein Sonderfall eines nichtdeterministischen (mit nur einer Möglichkeit) ist, gilt:
P ist Teilmenge von NP.
- Der Zeitaufwand eines nichtdeterministischen Algorithmus zur Lösung der gestellten Aufgabe ist gleich dem Zeitaufwand eines deterministischen Algorithmus zur Überprüfung der Lösung.
- Ist zu einem Problem aus NP eine Lösung gegeben, so kann sie in polynomialer Zeit verifiziert werden.

Das Hamilton-Problem gehört also zur Klasse NP. Bis heute hat noch niemand einen effizienten Algorithmus für dieses Problem gefunden. Es ist aber auch noch niemandem gelungen nachweisen dass das Hamilton-Problem nicht in der Klasse P liegt. Das gleiche gilt für eine ganze Reihe weiterer bekannter Probleme, von denen wir im folgenden noch einige vorstellen werden. Mehr noch: Für kein einziges Problem konnte bisher bewiesen werden, dass es in der Klasse NP aber nicht in der Klasse P liegt. Das heißt:
Ob die Klasse P eine echte Teilmenge der Klasse NP ist oder ob P und NP gleich sind, ist ein bis heute ungelöstes Problem der Informatik.

NP-Vollständigkeit

Beispiel: Das Problem des Handlungsreisenden

Ein Vertreter einer Firma, dessen Bezirk eine bestimmte Anzahl von Städten umfasst, beginnt seine Reise stets von seiner Basis aus, besucht jede Stadt genau einmal und kehrt dann zur Basis zurück. Um die Reisekosten zu minimieren, hat der Vertreter eine Tabelle der gegenseitigen Entfernungen der Städte seines Bezirks zusammengestellt. Es ist ein Algorithmus zu entwickeln, der die Reihenfolge der Städte für den kürzesten Reiseweg des Vertreters und die minimale Streckenlänge ausgibt.

Algorithmus: Handlungsreisender

Eingabe: Anzahl der Städte n , Entfernungstabelle

$p := (1, 2, 3, \dots, n)$

MinTour := p

MinStrecke := Tourlänge(p)

Permutiere(n) {und ermittle neue MinTour und MinStrecke}

Ausgabe: MinTour, MinStrecke

"Permutiere" ist dabei eine rekursive Prozedur (vergl. Skript: Problemlösestrategien der Informatik). Der Algorithmus führt im Aufrufbaum eine erschöpfende Suche durch. Da die Anzahl der Permutationen einer n -elementigen Menge gleich $n!$ ist, erfordert der Algorithmus einen zu $n!$ proportionalen Zeitaufwand.

Es gilt:

Ein effizienter Algorithmus für das Problem des Handlungsreisenden wäre gleichzeitig ein effizienter Algorithmus für das Problem des Hamilton-Zyklus.

Beweis:

Gegeben sei ein Graph G mit n Knoten, der auf die Existenz eines Hamilton-Zyklus untersucht werden soll. Wir konstruieren dazu folgendermaßen einen Graphen G' zum Problem des Handlungsreisenden:

Als Städte verwenden wir die Knoten des Graphen G ; als Entfernungen zwischen den Städten wählen wir 1, wenn zwischen den entsprechenden Knoten im Graphen G eine Kante existiert, sonst 2.

Dann bestimmen wir mit Hilfe des Algorithmus für das Problems des Handlungsreisenden eine minimale Tour in G' . Ist die minimale Tourlänge kleiner oder gleich der Knotenzahl n , so entspricht die minimale Tour in G' einem Hamilton-Zyklus in G .

Das Hamilton-Problem ist also auf das Problem des Handlungsreisenden zurückführbar.

Definition:

Ein Problem p_1 ist auf p_2 in Polynomialzeit *zurückführbar* ($|p_1| \leq |p_2|$), wenn es eine Transformation f gibt, die jeder Eingabe von p_1 eine Eingabe von p_2 in Polynomialzeit so zuordnet, dass jeder Lösungsalgorithmus von p_2 mittels f in einen Lösungsalgorithmus von p_1 überführt werden kann.

Anmerkung:

Ist $|p_1| \leq |p_2|$, so bedeutet das anschaulich, dass das Problem p_1 nicht schwieriger als p_2 ist. Ein effizienter Algorithmus für p_2 wäre gleichzeitig ein effizienter Algorithmus für p_1 .

Beispiel: Das Erfüllbarkeitsproblem

Ein boolescher Term heißt *erfüllbar*, wenn es eine Belegung der Variablen gibt, für die der Term den Wert TRUE annimmt. Das Erfüllbarkeitsproblem lautet:

Für einen beliebigen booleschen Term ist zu entscheiden, ob er erfüllbar ist oder nicht.

Ein deterministischer Lösungsalgorithmus erzeugt systematisch alle 2^n Variablenbelegungen und berechnet für jede Belegung den Wert des Terms. Der Zeitaufwand ist daher proportional zu 2^n , also exponentiell. Ein polynomialer deterministischer Algorithmus ist nicht bekannt.

Satz von Cook (Stephen Cook, 1971)

Jedes Problem aus NP lässt sich auf das Erfüllbarkeitsproblem zurückführen, d.h.:

$$\text{Aus } (p \text{ aus NP}) \text{ folgt } |p| \leq |\text{Erfüllbarkeitsproblem.}|$$

(ohne Beweis)

Definition:

Ein Problem p heißt *NP-vollständig*, wenn das Erfüllbarkeitsproblem auf p zurückführbar ist, d.h.:

$$p \text{ ist NP-vollständig genau dann, wenn } |\text{Erfüllbarkeitsproblem.}| \leq |p|$$

Satz:

Das Problem des Hamilton-Zyklus ist NP-vollständig.

(ohne Beweis)

Folgerung:

Damit ist auch das Problem des Handlungsreisenden NP-vollständig, da gilt:

$$|\text{Erfüllbarkeitsproblem.}| \leq |\text{Hamiltonproblem}| \leq |\text{Problem des Handlungsreisenden}|$$

Anmerkung:

Ist p NP-vollständig, d.h. es gilt: $|\text{Erfüllbarkeitsproblem}| \leq |p|$, so gilt wegen des Satzes von Cook :

$$|p| = |\text{Erfüllbarkeitsproblem}|.$$

Alle NP-vollständigen Probleme sind also gleich schwierig. Ließe sich von nur einem einzigen NP-vollständigen Problem zeigen, dass es zur Klasse P gehört, so gehörten alle NP-vollständigen Probleme zu P. In diesem Fall wären nach dem Satz von Cook die Problemklassen P und NP gleich.

Dies ist ein überzeugendes Argument dafür, dass $P \neq NP$ ist; diese Vermutung ist aber noch nicht bewiesen. Derzeit sind über tausend NP-vollständige Probleme bekannt. Auch das Problem der Zerlegung einer natürlichen Zahl in Primfaktoren ist NP-vollständig. Kryptosysteme mit öffentlichen Schlüsseln basieren auf der Hoffnung, dass $P \neq NP$ gilt.