?
P = NP
Die 1 Million $ Frage

Prof. Dr. Hans Jürgen Ohlbach

23 Hilbertsche Probleme aus dem Jahr 1900

Derzeit
10 gelöst
5 ungelöst
8 unklar formuliert oder teilweise gelöst

David Hilbert 1886

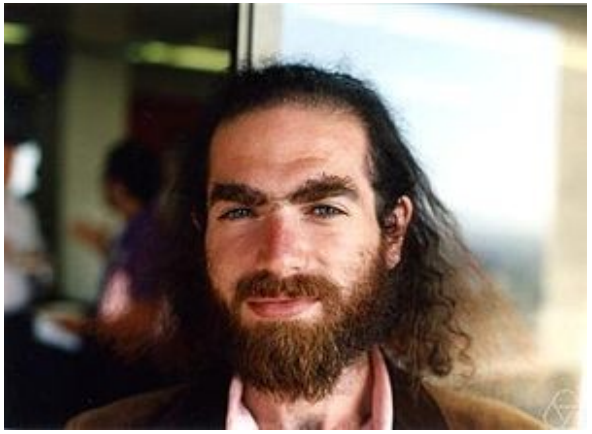
Die Millenium-Probleme

7 ungelöste Probleme aus dem Jahr 2000

Das *Clay Mathematics Institute* lobte ein Preisgeld von 1 Million US $ pro Problem aus.

Nummer 4 ist das P-NP-Problem

Nummer 5 ist gelöst (Perelman 2002)



Grigori Perelman 1993

Worum geht es dabei?

Ganz abstrakt:

Wie schnell können Computer (Algorithmen) Probleme lösen?

Dabei geht es eigentlich nicht um die Computer, sondern um das Lösungsverfahren an sich.

1. Beispiel: Suche in sortierter Liste.

Wie schnell kann man einen Namen in einem Telefonbuch mit 65536 Einträgen finden?

Das optimale Verfahren?

Binäre Suche (Wiederholtes Halbieren)

65536
32768
16384
8192
4096
2098
1024
512
256
128
64
32
16
8
4
2
1

16 Halbierungen
$16 = \log_2(65536)$

Suche in sortierter Liste: Optimalität

Frage:
Könnte es noch ein schnelleres Verfahren geben als Binäre Suche (mit Aufwand $\log_2(n)$)?

Antwort: NEIN!

Beweisskizze:
Um jedes der n-Elemente identifizieren zu können, kann man sie durchnummerieren. Um n Nummern darzustellen, benötigt man ca. $\log_2(n)$ Bits (in Binärdarstellung),

(Bsp.: Für die Zahl 7 braucht man 3 Bits: 111)

Um ein bestimmtes Element zu identifizieren, muss das Verfahren, egal wie es das macht, die $\log_2(n)$ Bits bestimmen.

Suche in sortierter Liste: Komplexität

Wir haben also für das Problem:

Suche in Sortierter Liste

ein ***optimales*** Verfahren gefunden (Binäre Suche),
und für n Elemente benötigt es $\log_2(n)$ Schritte.

Man kann daher sagen:

Die Komplexität des ***Problems***

Suche in Sortierter Liste

ist von der *Ordnung* $\log_2(n)$ (man schreibt $O(\log_2(n))$

Die Aussage bezieht sich jetzt ***auf das Problem selbst***,
nicht mehr auf das Verfahren (den Algorithmus).

Algorithmus versus Problem

Unterscheide:

- Komplexität eines **Verfahrens** (Algorithmus),
  d.h. wieviele Rechenschritte braucht es bei einer Eingabe der Größe n?

- Komplexität des **Problems**,
  d.h. Komplexität des *bestmöglichen* Verfahrens.

Beispiel 2: Sortieren

Wie schnell kann man ein Telefonbuch mit 65536 Einträgen sortieren?

Ein einfaches Verfahren: InsertionSort:
Man fügt nacheinander jedes Element in die schon sortierte Liste ein.

Komplexität $O(n^2)$,
d.h. Für das Telefonbuch über 4 Mrd. Operationen.

Besser: Mergesort

Man

- halbiert die Liste
- wendet Mergesort auf die beiden Hälften getrennt an
- verschmilzt die beiden sortierten Teillisten.

Komplexität für n zu sortierende Elemente

- $\log_2(n)$ Halbierungen
- Das Verschmelzen zweier sortierter Listen geht in n Schritten.

Zusammen:

$O(n \log_2(n))$ Schritte.

d.h. für das Telefonbuch ca. 1 Million Operationen.

Wie schnell kann man sortieren?

Frage:
Geht Sortieren noch schneller als mit **O(n $\log_2$(n))** Schritten?

Antwort:
Nein!

Beweisskizze:
Gesucht ist die *sortierte* Reihenfolge der Elemente,
also eine Folge von **n** Positionsnummern (Zahlen).
Für jede dieser Zahlen braucht man $\log_2$(n) Bits.
Man muss also **n $\log_2$(n)** Bits berechnen.

Die Komplexität des **Sortierproblems** ist also **O(n $\log_2$(n))**
Es kann keinen Algorithmus geben, der noch schneller ist!

Typisches Problem für einen Informatiker

Gegeben:
ein zu lösendes Problem

Gesucht:
ein optimaler Algorithmus

Zu beweisen:
es kann keinen schnelleren Algorithmus geben (bis auf kleine technische Verbesserungen).

Typisches Beweisverfahren für einen Optimalitätsbeweis

Gegeben:
ein *neues* zu lösendes Problem

Gefunden:
ein prima Algorithmus mit einer Komplexität O(f(n))
(z.B. $f(n) = n \log_2(n)$ oder $f(n) = n^2$ oder ...)

Zu beweisen:
es kann keinen schnelleren Algorithmus geben

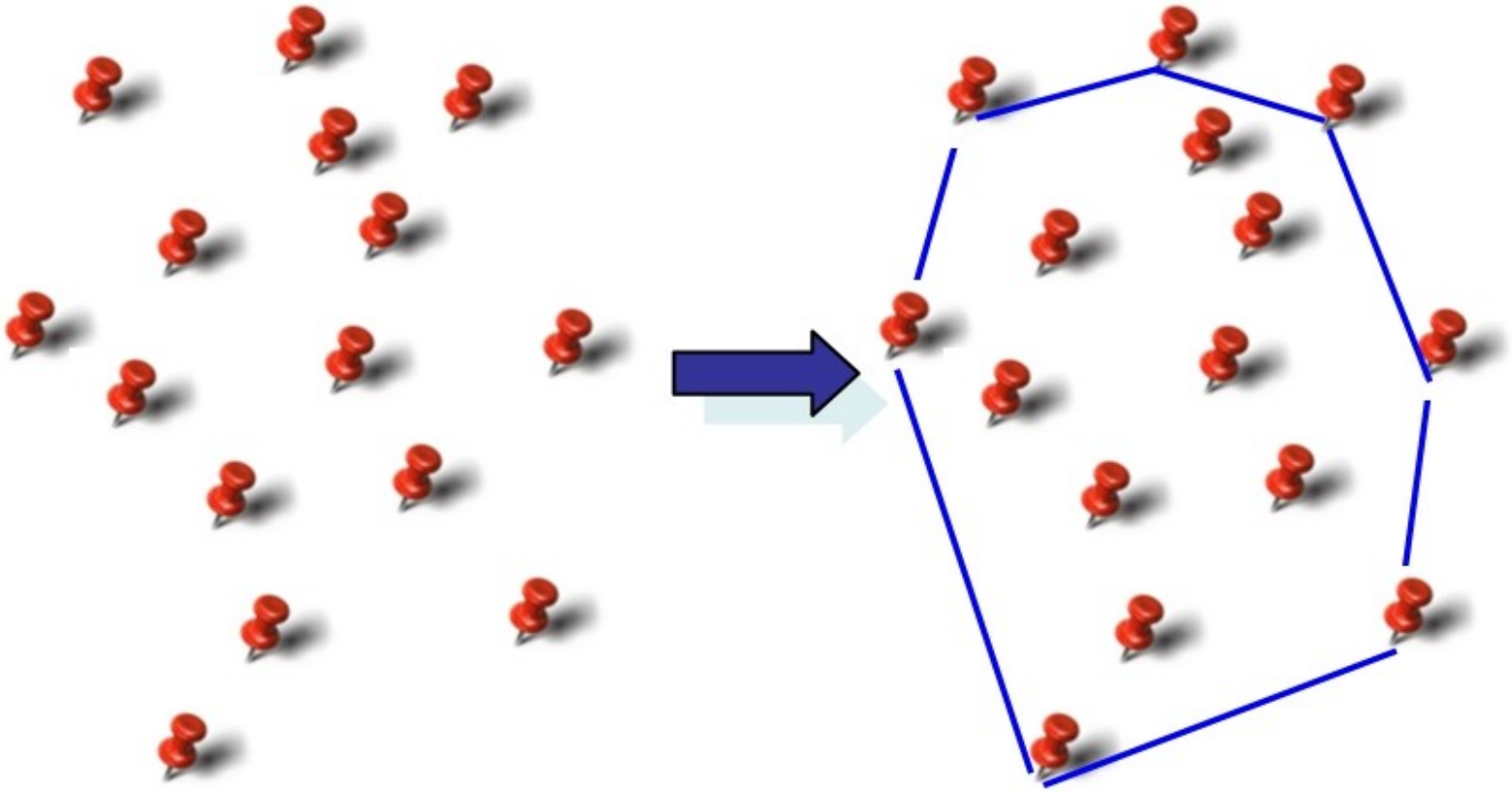
Beweisidee:

- suche bekanntes Problem mit der Komplexität O(f(n))
- finde eine Transformation, die das bekannte Problem auf das neue Problem transformiert, so dass
  - Die Transformation nicht mehr Aufwand als O(f(n)) erfordert
  - Die Lösung des transformierten Problems zurücktransformiert werden kann auf die Lösung des Originalproblems.

Schlussfolgerung:
Wenn es für das neue Problem ein *schnelleres* Verfahren gäbe, dann könnte man das alte Problem mit dem Transformationstrick und dem schnellen Verfahren für das neue Problem schneller als mit O(f(n)) lösen.

Widerspruch!

Beispiel: Konvexe Hülle



Komplexität: $O(n \log_2(n))$

Bild aus: Logofätu: Algorithmen und Problemlösungen mit C++

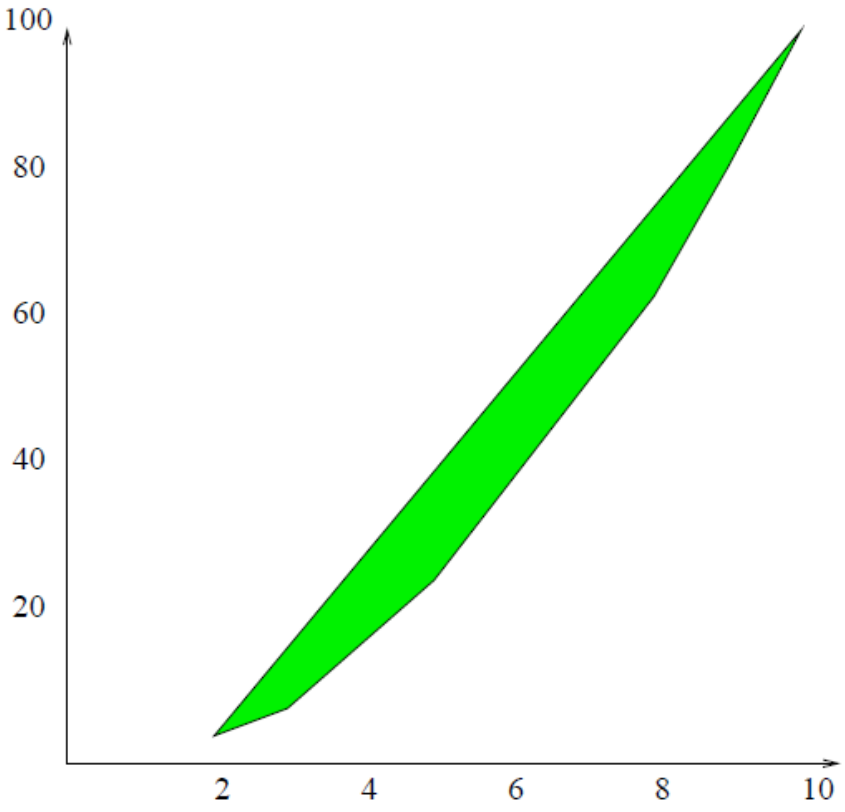
Idee für Optimalitätsbeweis

Transformation des Sortierproblems auf das konvexe Hüllenproblem.

Bsp.:
Sortieren von (10,5,2,3,9,8).

Transformation:
$n \rightarrow (n,n^2)$ (Parabel, ist konvex)

Die konvexe Hülle erzeugt die *sortierte* Reihenfolge der Zahlen.



Schlussfolgerung:

Da Sortieren nicht schneller als $O(n \log_2(n))$ geht, kann auch die konvexe Hüllenberechnung nicht schneller als $O(n \log_2(n))$ gehen.

Stand soweit

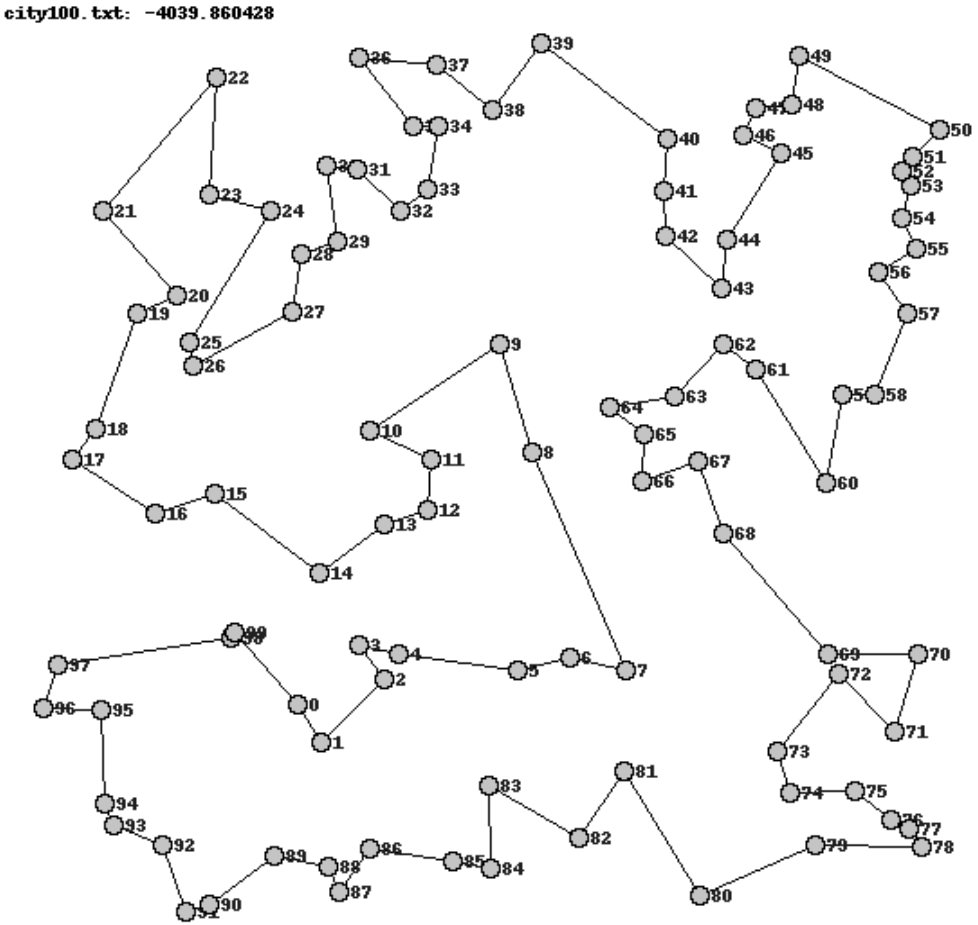
- Viele Probleme wurden untersucht, deren Komplexität O(f(n)) ist, wobei f(n) ein Polynom ist (Bsp.: f(n) = n oder f(n) = n^2 oder f(n) = n^3 oder ...)
- optimale Algorithmen wurden gefunden
- Bis auf technische Feinheiten, oder Varianten für Sonderfälle, ist die Forschung weitgehend abgeschlossen.

Weiterhin problematisch:

Probleme, deren Aufwand exponentiell ist, z.B. O(2^n).

Brauchbare Rechenzeiten hat man dann nur für kleines n.

Problem: Finde die kürzeste Route durch alle Städte



Variante des TSP

Problem:
Gibt es eine Route durch *alle* Städte, für die meine Tankfüllung reicht?

Das ist ein typisches **NP-Problem.**

N: Nichtdeterministisch
P: Polynomiell.

NP-Verfahren:

- *Rate* eine Route
- Teste, ob die Tankfüllung ausreicht.
  Der Test geht in linearer Zeit (P).

Wenn man beim ersten mal *optimal* geraten hat, dann ist man in polynomieller Zeit fertig.

Verfahren für TSP

Ein praktisches NP-Verfahren für TSP:

- *Rate* eine Route
- Teste, ob die Tankfüllung ausreicht.
- Falls nicht, probiere eine Alternative.

Es gibt leider ***exponentiell*** viele Alternativen.

Wenn man immer erst die falsche probiert, braucht das Verfahren ***exponentiell*** viel Zeit.

Das P-NP Problem

Ein praktisches NP-Verfahren für TSP:

- *Rate* eine Route
- Teste, ob die Tankfüllung ausreicht.
- Falls nicht, probiere eine Alternative.

Kann man in ***polynomieller*** Zeit die richtige Alternative auswählen???

Vermutung: nein!

Es ist aber noch nicht bewiesen!

Falls jemand tatsächlich ein *polynomielles* Auswahlverfahren findet, dann ist P = NP, er bekommt 1 Million US$, und es lassen sich viele derzeit schwierige Problem effizient lösen.

Bis auf weiteres: P ≠NP

Ein praktisches NP-Verfahren für TSP:

- *Rate* eine Route
- Teste, ob die Tankfüllung ausreicht.
- Falls nicht, probiere eine Alternative.

Benutze Heuristiken, um eine aussichtsreiche Alternative auszuwählen.

Es gibt viele Varianten, die für unterschiedliche Beispiele unterschiedlich gut funktionieren.

Optimale Heuristiken sind nicht in Sicht.

Ein weites Feld für Forschungen!

Geht es noch schlimmer?

Ja!

Für manche Probleme gilt:

Falls es eine Lösung gibt, findet der Computer sie,

Falls nicht, kann er unendlich lange suchen.

Das Halteproblem

Kann man ein Programm schreiben, welches beliebige Programme daraufhin untersucht, ob sie immer halten, oder in Endlosschleifen kommen können.

Programm HT (Halte-Test)

HT(P) = „ja“, falls P immer hält

HT(P) = „nein“, falls P in Endlosschleifen geraten kann.

Das gibt es nicht!

Beweisskizze

Angenommen, so ein Programm HT gibt es:

HT(P):

Falls hält(P) drucke „ja“

ansonsten drucke „nein“

Definiere HT'(P):

Falls HT(P) = „ja“ **Endlosschleife**

ansonsten drucke „ja“

Jetzt teste: HT'(HT')

Falls HT(HT') „ja“ sagt, gerät HT' in die Endlosschleife, d.h. HT lag falsch.

Falls HT(HT') „nein“ sagt, sagt HT' „ja“, d.h. HT' liegt falsch.

Widerspruch!

Unvollständigkeit

Geht es noch schlimmer?

Ja!

Für manche Problem gilt:

Es gibt eine Lösung, aber kein Computer kann sie finden.

Beispiel:

Beweise für mathematische Theoreme

(Gödels Unvollständigkeitsbeweis der Arithmetik)

Fazit

Unterscheide:
Komplexität des Algorithmus und Komplexität des Problems.

- Polynomielle Probleme sind weitgehend erforscht.
- NP-Probleme sind auf Heuristiken angewiesen, solange nicht doch P = NP bewiesen ist.
- Mit jeder neuen Anwendung scheinen neue Heuristiken notwendig zu werden. Das scheint eine endlose Geschichte zu werden.
- Viele Probleme sind nur semi-entscheidbar
- Manche Problem sind sogar unlösbar.

Für Informatiker gibt es noch beliebig viele sehr anspruchsvolle Probleme zu lösen
(und nicht nur immer neue Programme zu schreiben).