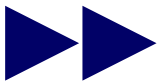
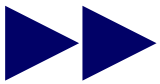
Theoretische Informatik
Kap 3: Komplexitätstheorie



Gliederung der Vorlesung

Theoretische Informatik
Kap 3: Komplexitätstheorie



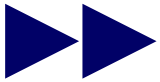
Das P=NP? Problem

Warum ist das P = NP? Problem wichtig?

- Warum ist die Zeitkomplexitätsklasse P interessant?
- Warum ist die Zeitkomplexitätsklasse NP interessant?
- Was hätte man von einer Antwort auf obige Frage?

... und wie könnte man zu einer Antwort gelangen ...

Theoretische Informatik
Kap 3: Komplexitätstheorie

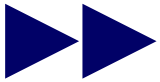


Komplexitätstheorie

Arbeitszeit eines Rechners mit 10^6 Op/sec

Rechenschritte	n = 10	n = 20	n = 30	n = 40	n = 50
n	0,00001 s	0,00002 s	0,00003 s	0,00004 s	0,00005 s
n^2	0,0001 s	0,0004 s	0,0009 s	0,0016 s	0,0025 s
n^3	0,001 s	0,008 s	0,027 s	0,064 s	0,125 s
n^5	0,1 s	3,2 s	24,3 s	1,7 min	5,2 min
2^n	0,001 s	1,0 s	17,9 min	12,7 Tage	35,7 Jahre
3^n	0,059 s	58 min	6,5 Jahre	3855 Jahr-hunderte	$2*10^8$ Jahr-hunderte

Hinweis: n – Maß für die Problemgröße



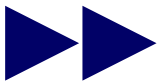
Komplexitätstheorie

Problemgröße, die in einer Stunde bewältigt werden kann

Rechenschritte	heute	100-fache Geschwindigkeit	1000-fache Geschwindigkeit
n	N_1	100 N_1	1000 N_1
n^2	N_2	10 N_2	31,6 N_2
n^3	N_3	4,6 N_3	10 N_3
n^5	N_4	2,5 N_4	3,9 N_5
2^n	N_5	N_5 + 6,6	N_5 + 9,9
3^n	N_6	N_6 + 4,2	N_6 + 6,3

Hinweis: n – Maß für die Problemgröße

Theoretische Informatik
Kap 3: Komplexitätstheorie



Das P=NP? Problem

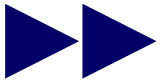
mit Blick auf die beiden Tabellen kommt man zu folgender These

Ein Problem ist dann „praktisch“ lösbar, wenn es einen „schnellen“ Lösungsalgorithmus für dieses Problem gibt.

- „schnell“ heißt, daß es einen polynomiellen Lösungsalgorithmus für dieses Problem gibt, d.h. es gibt Zahlen c,r > 0, so daß dieser Lösungsalgorithmus auf jeder zulässigen Eingabe w nicht mehr als $c*|w|^r$ viele Rechenschritte arbeitet

Hinweis: |w| bezeichnet die Größe der Eingabe, wobei eine „sinnvolle“ Kodierung vorausgesetzt wird

Theoretische Informatik
Kap 3: Komplexitätstheorie



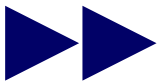
Das P=NP? Problem

Was hat obige These mit der Zeitkomplexitätsklasse P zu tun?

Problemtyp 1: formale Sprachen akzeptieren

- die Zeitkomplexitätsklasse P enthält alle Sprachen L, für die es eine „schnelle“ det. sprachakzeptierende TM M mit L(M) = L gibt

Theoretische Informatik
Kap 3: Komplexitätstheorie



Das P=NP? Problem

Was hat obige These mit der Zeitkomplexitätsklasse P zu tun?

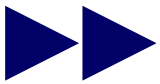
Problemtyp 2: das Wortproblem für formale Sprachen lösen

ohne viel Aufwand kann man zeigen ...

- die Zeitkomplexitätsklasse P enthält alle Sprachen L, für die es eine „schnelle“ det. TM M gibt, die das Wortproblem für die Sprache L löst (/* also, die L „entscheidet“ */)

Hinweis: beim Wortproblem für ein Sprache $L \subseteq \Sigma^*$ geht es darum, für jedes Wort $w \in \Sigma^*$ zu entscheiden, ob $w \in L$ oder $w \notin L$ gilt

Theoretische Informatik
Kap 3: Komplexitätstheorie



Das P=NP? Problem

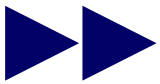
Was hat obige These mit der Zeitkomplexitätsklasse P zu tun?

Problemtyp 3: eine Funktion f: N → N berechnen

ohne viel Aufwand kann man zeigen ...

- wenn es eine „schnelle“ det. TM M gibt, die die Funktion f berechnet, so gehört die Sprache L = { (x,y) | y = f(x) } zur Zeitkomplexitätsklasse P

Theoretische Informatik
Kap 3: Komplexitätstheorie



Das P=NP? Problem

Was hat obige These mit der Zeitkomplexitätsklasse P zu tun?

es sei irgendein Problem der genannten Problemtypen gegeben

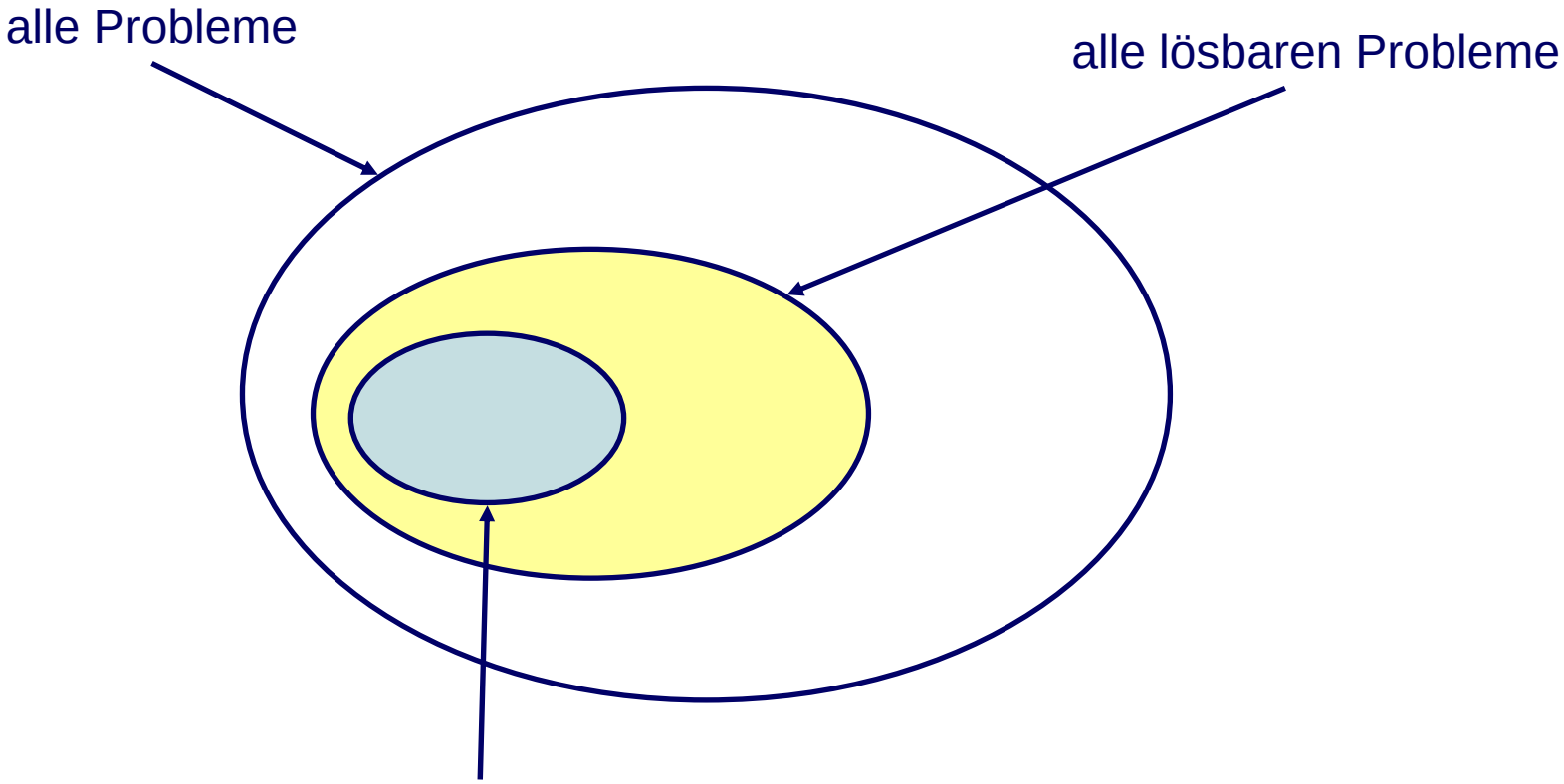
es sei bekannt, daß es einen „schnellen“ Lösungsalgorithmus für dieses Problem gibt (/* der auf einem von-Neumann-Rechner implementierbar ist */)

dann kann man ohne viel Aufwand zeigen ...

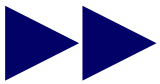
- es gibt eine „schnelle“ det. TM zur Lösung dieses Problems

Theoretische Informatik
Kap 3: Komplexitätstheorie

Das P=NP? Problem



Theoretische Informatik
Kap 3: Komplexitätstheorie



Das P=NP? Problem

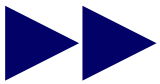
Warum ist die Zeitkomplexitätsklasse NP interessant?

- es gibt eine ganze Reihe von „praktisch“ interessanten Problemen, für die nicht bekannt ist, ob es überhaupt „schnelle“ Lösungsalgorithmen für diese Probleme gibt

- allen diesen Problemen ist gemein, daß es „schnelle“ nicht-det. Lösungsalgorithmen für diese Probleme gibt

  ... die bekannten „schnellen“ nichtdet. Lösungsalgorithmen basieren stets auf derselben algorithmischen Idee

Theoretische Informatik
Kap 3: Komplexitätstheorie



Das P=NP? Problem

Zusammengesetzte Zahl

- Gegeben: eine natürliche Zahl k
- Frage: Gibt es natürliche Zahlen a >1 und b >1 mit a*b = k?

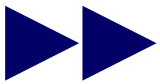
Rundreise-Problem

- Gegeben: eine n×n-Matrix von Entfernungen zwischen n „Städten“ und eine Zahl r
- Frage: Gibt es eine „Rundreise“ der Länge kleiner gleich r?

Erfüllbarkeit

- Gegeben: eine aussagenlogische Formel F
- Frage: Ist F erfüllbar?

Theoretische Informatik
Kap 3: Komplexitätstheorie



Das P=NP? Problem

Zusammengesetzte Zahl

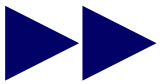
- Gegeben: eine natürliche Zahl k
- Frage: Gibt es natürliche Zahlen a >1 und b >1 mit a*b = k?

wichtig für die Kryptographie (/* SSL, RSA, ... */)

algorithmische Idee zur Lösung des Problems

1. „rate“ zwei Zahlen i und j mit $2 \leq i, j \leq k - 1$
2. überprüfe, ob i*j = k
   - falls ja, so gibt „k ist eine zusammengesetzte Zahl“ aus

Theoretische Informatik
Kap 3: Komplexitätstheorie



Das P=NP? Problem

Rundreise-Problem

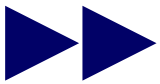
- Gegeben: eine n×n-Matrix von Entfernungen zwischen n „Städten“ und eine Zahl r
- Frage: Gibt es eine „Rundreise“ der Länge kleiner gleich r?

wichtig für Transportoptimierung, Steuerung von Robotern, ...

algorithmische Idee zur Lösung des Problems

1. „rate“ eine Rundreise s1,...,sn
2. überprüfe, ob die Rundreise eine Länge kleiner gleich k hat
   - falls ja, so gib „es gibt eine Rundreise kleiner gleich k“ aus

Theoretische Informatik
Kap 3: Komplexitätstheorie



Das P=NP? Problem

Erfüllbarkeit

- Gegeben: eine aussagenlogische Formel F
- Frage: Ist F erfüllbar?

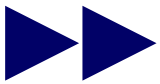
wichtig für die KI (/* Wissensverarbeitung */)

algorithmische Idee zur Lösung des Problems

1. „rate“ eine n-Bit Zahl $a = a_1 a_2 ... a_n$
2. überprüfe, ob die Belegung β mit $\beta(x_1) = a_1$, $\beta(x_2) = a_2$, ..., $\beta(x_n) = a_n$ die Formel F erfüllt
   - falls ja, so gib „F ist erfüllbar“ aus

Hinweis: es habe F die Variablen $x_1, x_2, ..., x_n$

Theoretische Informatik
Kap 3: Komplexitätstheorie



Das P=NP? Problem

Gemeinsamkeiten (/* algorithmische Idee */)

alle Algorithmen implementieren folgende Idee

- rate zu einer gegebenen Probleminstanz einen Lösungskandidaten
- verifiziere für diesen Lösungskandidaten, ob er Lösung für die gegebene Probleminstanz ist
  - falls ja, beantworte die Frage für die gegeben Probleminstanz

... es wird das Prinzip „Durchprobieren bis zum ersten Erfolg“ realisiert

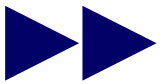
Das P=NP? Problem

Gemeinsamkeiten (/* Laufzeit */)

es gibt Zahlen r, r‘ ≥ 0 mit folgenden Eigenschaften

- zur Beschreibung eines Lösungskandidaten für eine gegebene Probleminstanz genügt Platz in $O(n^r)$
- um zu verifizieren, ob ein Lösungskandidat, Lösung der gegebenen Probleminstanz ist, reichen $O(n^{r'})$ viele Rechenschritte

Hinweis: es sei n die Größe der Beschreibung einer jeweiligen Probleminstanz



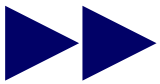
Das P=NP? Problem

Zwischenfazit

es seien L(zZ) (/* L(Rr) bzw. L(Ef) */) die zum Entscheidungs-problem „zusammengesetzte Zahl“ (/* „Rundreise“ bzw. „Erfüllbarkeit“ */) gehörenden Sprachen

dann gilt:

- es gibt eine „schnelle“ nicht-det. sprachakzeptierende TM für die Sprache L(zZ) (/* L(Rr) bzw. L(Ef) */)
- die Sprache L(zZ) (/* L(Rr) bzw. L(Ef) */) gehört zur Zeitkomplexitätsklasse NP



Das P=NP? Problem

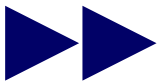
Verallgemeinerung

es sei Π irgendein Entscheidungsproblem, für das es einen „schnellen“ nicht-det. Lösungsalgorithmus gibt, der das Prinzip „Durchprobieren bis zum ersten Erfolg“ realisiert

dann gilt:

- es gibt eine „schnelle“ nicht-det. sprachakzeptierende TM für die Sprache L(Π)
- die Sprache L(Π) gehört zur Zeitkomplexitätsklasse NP

Theoretische Informatik
Kap 3: Komplexitätstheorie



Das P=NP? Problem

für „Rundreise“ und „Erfüllbarkeit“ und tausende andere Entscheidungsprobleme Π gilt ...

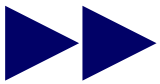
- es ist bekannt, daß es einen „schnellen“ nicht-det. Lösungsalgorithmus gibt, der das Prinzip „Durchprobieren bis zum ersten Erfolg“ realisiert
- es ist offen, ob es überhaupt einen „schnellen“ det. Lösungsalgorithmus gibt

Optimierung, Logik, Zahlentheorie, Graphentheorie, algorithmische Geometrie, Theorie der formalen Sprachen, Kryptographie ...

... für die zugehörigen Sprachen L(Π) ist bekannt, daß sie zur Zeitkomplexitätsklasse NP gehören und offen, ob sie zur Zeitkomplexitätsklasse P gehören

Hinweis: man weiß inzwischen, daß das Problem „Zusammengesetzte Zahl“ „schnell“ lösbar ist...

Theoretische Informatik
Kap 3: Komplexitätstheorie



Das P=NP? Problem

Zielstellung „schwierigste“ Sprachen in der Zeitkomplexitätsklasse NP identifizieren (Klasse NPV)

- für Sprachen $L \in NPV$ besteht die Hoffnung, daß ein Nachweis für $L \notin P$ am ehesten zu finden ist
- es gibt sehr viele Sprachen in NPV ...

- wenn der Nachweis gelingt, daß ein Sprache $L \in NPV$ nicht zu P gehört, so gehört keine Sprache aus NPV zu P
- wenn eine Sprache $L \in NPV$ zu P gehört, so gilt P = NP

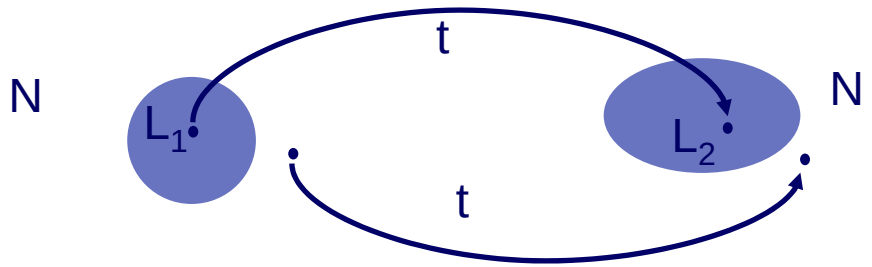
Theoretische Informatik
Kap 3: Komplexitätstheorie

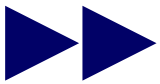
Das P=NP? Problem

Es seien $L_1, L_2 \subseteq N$. Die Sprache L_1 ist auf die Sprache L_2 polynomiell reduzierbar, falls es eine einstellige mit einer deterministischen $O(n^r)$-zeitbeschränkten TM berechenbare Funktion t: N → N gibt, so daß für alle $x \in N$ gilt:

- t(x) ist definiert
- $x \in L_1$ genau dann, wenn $t(x) \in L_2$

Bezeichnung: $L_1 \leq_p L_2$





Das P=NP? Problem

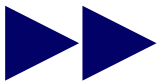
Konsequenzen

Es seien $L_1, L_2 \subseteq N$. Dann gilt:

1. Wenn $L_1 \leq_p L_2$ und $L_2 \in P$, so gilt: $L_1 \in P$.
2. Wenn $L_1 \leq_p L_2$ und $L_1 \notin P$, so gilt: $L_2 \notin P$.

Nachweis von (1): „Komposition“
Nachweis von (2): folgt unmittelbar aus (1)

Theoretische Informatik
Kap 3: Komplexitätstheorie

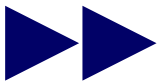


Das P=NP? Problem

der zentrale Begriffsbildung

Eine Sprache $L_1 \subseteq N$ heißt NP-vollständig (/* Bezeichnung: $L_1 \in$ NPV */) genau dann, wenn gilt:

- $L_1 \in$ NP
- für alle $L \in$ NP gilt: $L \leq_p L_1$



Das P=NP? Problem

Konsequenzen

Angenommen, es gibt ein $L \in NPV$ mit $L \in P$.
Dann gilt: $P = NP$.

Angenommen, es gibt ein $L \in NPV$ mit $L \notin P$.
Dann gilt für alle $L' \in NPV$: $L' \notin P$.