Vorlesung

Theoretische Informatik II

Bernhard Beckert

Institut für Informatik

UNIVERSITÄT KOBLENZ · LANDAU

Wintersemester 2007/2008

Dank

Diese Vorlesungsmaterialien basieren zum Teil auf den Folien zu den Vorlesungen von

Katrin Erk (gehalten an der Universität Koblenz-Landau)

Jürgen Dix (gehalten an der TU Clausthal)

Christoph Kreitz (gehalten an der Universität Potsdam)

Ihnen gilt mein herzlicher Dank.

– Bernhard Beckert, Oktober 2007

Teil VI

Komplexitätstheorie

Welche Arten von Komplexität gibt es?

- Zeit
- Speicher

DTIME und NTIME

Definition 15.1 (NTIME$(T(n))$, DTIME$(T(n))$)

Basismodell: k-DTM $\mathcal{M}$ (ein Band für die Eingabe).
Wenn $\mathcal{M}$ mit jedem Eingabewort der Länge n höchstens $T(n)$ Schritte macht, dann wird sie $T(n)$**-zeitbeschränkt** genannt.
Die von $\mathcal{M}$ akzeptierte Sprache hat **Zeitkomplexität** $T(n)$ (tatsächlich meinen wir $\max(n+1, \lceil T(n) \rceil)$).

- **DTIME**$(T(n))$ ist die Klasse der Sprachen, die von $T(n)$-zeitbeschränkten, DTMn akzeptiert werden.
- **NTIME**$(T(n))$ ist die Klasse der Sprachen, die von $T(n)$-zeitbeschränkten, NTMn akzeptiert werden.

DSPACE und NSPACE

Definition 15.2 (NSPACE$(S(n))$, DSPACE$(S(n))$)

Basismodell: k-DTM $\mathcal{M}$ davon ein spezielles Eingabeband (**offline DTM**).
Wenn $\mathcal{M}$ für jedes Eingabewort der Länge n maximal $S(n)$ Zellen auf den Ablagebändern benutzt, dann heißt $\mathcal{M}$ **$S(n)$-speicherbeschränkt.**
Die von $\mathcal{M}$ akzeptierte Sprache hat **Speicherkomplexität** $S(n)$ (tatsächlich meinen wir $\max(1, \lceil S(n) \rceil)$)

- **DSPACE**$(S(n))$ ist die Klasse der Sprachen, die von $S(n)$-speicherbeschränkten, DTMn akzeptiert werden.
- **NSPACE**$(S(n))$ ist die Klasse der Sprachen, die von $S(n)$-speicherbeschränkten, NTMn akzeptiert werden.

Wichtige Fragen

Wichtige Fragen (1)

Zeit: Wird jede Sprache aus **DTIME**$(f(n))$ von einer DTM entschieden?

Speicher: Wird jede Sprache aus **DSPACE**$(f(n))$ von einer DTM entschieden?

Die Funktionen f sind immer sehr einfache Funktionen, insbesondere sind sie alle berechenbar. In dieser Vorlesung betrachten wir nur Potenzen $f(n) = n^i$.

Wichtige Fragen (2)

Zeit/Speicher: Wie sieht es mit **NTIME**$(f(n))$, **NSPACE**$(f(n))$ aus?

Zeit vs. Speicher: Welche Beziehungen gelten zwischen **DTIME**$(f(n))$, **DSPACE**$(f(n))$, **NTIME**$(f(n))$, **NSPACE**$(f(n))$?

Wichtige Fragen

Halten, hängen nach n Schritten.

Zeitbeschränkt: Was bedeutet es, daß **eine DTM höchstens n Schritte macht**?

Halten?: Streng genommen, müßte sie dann entweder **halten** oder **hängen**. Das bedeutet, im ersten Fall, daß die Eingabe **akzeptiert** wird.

Hängen?: DTM'n auf beidseitig unendlichem Band können aber nicht hängen.

Wichtige Fragen

Stoppen nach n Schritten.

Stoppen: Wir wollen unter **eine DTM macht höchstens n Schritte** folgendes verstehen:

- Sie **hält** (und **akzeptiert** die Eingabe) innerhalb von n Schritten.
- Sie **hängt** (und **akzeptiert** die Eingabe **nicht**) innerhalb von n Schritten.
- Sie **stoppt** innerhalb von n Schritten **ohne in den Haltezustand überzugehen**. Auch hier wird die Eingabe nicht akzeptiert.

Antworten

Antworten (informell)

Zeit: Jede Sprache aus $\mathbf{DTIME}(f(n))$ ist entscheidbar: Man warte einfach solange wie $f(n)$ angibt. Falls bis dahin kein “Ja” gekommen ist, ist die Antwort eben “Nein”.

Speicher: Jede Sprache aus $\mathbf{DSPACE}(f(n))$ ist entscheidbar. Denn es gibt nur **endlich viele verschiedene Konfigurationen**. Falls also die DTM nicht terminiert (was passiert), macht man folgendes: man schreibt alle Konfigurationen auf (die komplette Rechnung). Falls die DTM nicht terminiert, muss sie in eine Schleife laufen (eine Konfiguration erreichen, die sie schon einmal hatte). Das lässt sich feststellen.

Antworten

Antworten (informell)

NTM vs. DTM: Trivial sind $\mathbf{DTIME}(f(n)) \subseteq \mathbf{NTIME}(f(n))$ und $\mathbf{DSPACE}(f(n)) \subseteq \mathbf{NSPACE}(f(n))$. Versucht man aber eine NTM durch eine DTM zu simulieren, braucht man (vermutlich) exponentiell mehr Zeit.
Für die Speicherkomplexität kann man zeigen:
$\mathbf{NSPACE}(f(n)) \subseteq \mathbf{DSPACE}(f^2(n))$.

Antworten

Antworten (informell):

Zeit vs. Speicher: Trivial sind $\mathbf{DTIME}(f(n)) \subseteq \mathbf{DSPACE}(f(n))$ und $\mathbf{NTIME}(f(n)) \subseteq \mathbf{NSPACE}(f(n))$.

Aber $\mathbf{DSPACE}(f(n))$, $\mathbf{NSPACE}(f(n))$ sind viel größer.

Bandkompression

Konstante Faktoren werden ignoriert

Nur **die funktionale Wachstumsrate einer Funktion in Komplexitätsklassen zählt**: Konstante Faktoren werden ignoriert.

Theorem 15.3 (Bandkompression)

Für jedes $c \in \mathbb{R}^+$ und jede Speicherfunktion $S(n)$ gilt:

$$\textbf{DSPACE}(S(n)) = \textbf{DSPACE}(cS(n))$$
$$\textbf{NSPACE}(S(n)) = \textbf{NSPACE}(cS(n))$$

Bandkompression

Beweis

Eine Richtung ist trivial. Die andere geht, indem man eine feste Anzahl r ($> \frac{2}{c}$) von benachbarten Zellen auf dem Band als **ein neues Symbol** darstellt. **Die Zustände der neuen Maschine simulieren die alten Kopfbewegungen als Zustandsübergänge** (innerhalb des neuen Symbols). D.h. für r Zellen der alten Maschine werden nun nur maximal 2 benutzt: im ungünstigsten Fall, wenn man von einem Block in den benachbarten geht.

Zeitbeschleunigung

Theorem 15.4 (Zeitbeschleunigung)

Für jedes $c \in \mathbb{R}^+$ *und jede Zeitfunktion* $T(n)$ *mit* $\lim_{n\to\infty} \frac{T(n)}{n} = \infty$ *gilt:*

$$\mathbf{DTIME}(T(n)) = \mathbf{DTIME}(cT(n))$$
$$\mathbf{NTIME}(T(n)) = \mathbf{NTIME}(cT(n))$$

Zeitbeschleunigung

Beweis

Eine Richtung ist trivial. Der Beweis der anderen läuft wieder über die Repräsentation einer festen Anzahl r ($> \frac{4}{c}$) benachbarter Bandzellen durch **neue Symbole**. Im Zustand der neuen Maschine wird wieder kodiert, welches Zeichen und welche Kopfposition (der simulierten, alten Maschine) aktuell ist. **Wenn die alte Maschine simuliert wird, muss die neue nur** 4 **statt** r **Schritte machen**: 2 um die neuen Felder zu drucken und weitere 2 um den Kopf auf das neue und wieder zurück auf das alte (im schlimmsten Fall) zu bewegen.

Wachstumsrate von DTIME und DSPACE

Lemma 15.5 (Wachstumsrate von DTIME, DSPACE)

Es sei $T(n)$ eine Zeitfunktion mit $\lim_{n\to\infty} \frac{T(n)}{n} = \infty$, und es sei $S(n)$ eine Speicherfunktion.

(a) *Es sei $f(n) = \mathbf{O}(T(n))$. Dann gilt:*
$\mathbf{DTIME}(f(n)) \subsetneqq \mathbf{DTIME}(T(n))$.

(b) *Es sei $g(n) = \mathbf{O}(S(n))$. Dann gilt:*
$\mathbf{DSPACE}(g(n)) \subsetneqq \mathbf{DSPACE}(S(n))$.

P, NP, PSPACE

Definition 15.6 (P, NP, PSPACE)

$$\begin{aligned} \mathbf{P} &:= \bigcup_{i \geqslant 1} \mathbf{DTIME}(n^i) \\ \mathbf{NP} &:= \bigcup_{i \geqslant 1} \mathbf{NTIME}(n^i) \\ \mathbf{PSPACE} &:= \bigcup_{i \geqslant 1} \mathbf{DSPACE}(n^i) \end{aligned}$$

P, NP, PSPACE

Intuitiv

- Probleme in **P** sind effizient lösbar, jene aus **NP** können in exponentieller Zeit gelöst werden.
- **PSPACE** ist eine sehr große Klasse, weit größer als **P** oder **NP**.

Komplexitätsklassen für Funktionen

Komplexitätsklassen für Funktionen

Eine Funktion $f : \mathbb{N} \to \mathbb{N}$ ist in **P**, falls es eine DTM $\mathcal{M}$ und ein Polynom $p(n)$ gibt, so daß für jedes n der Funktionswert $f(n)$ in höchstens $p(\text{länge}(n))$ Schritten von $\mathcal{M}$ berechnet wird. Dabei gilt $\text{länge}(n) = \lg n$, denn man braucht $\lg n$ Zeichen um die Zahl n binär darzustellen.

Analog funktioniert dies für alle anderen Komplexitätsklassen.

Beziehungen zwischen den Komplexitätsklassen

Frage

Was sind die genauen Beziehungen zwischen den Komplexitätsklassen P, NP, PSPACE?

$$\mathbf{P} \subseteq \mathbf{NP} \subseteq \mathbf{PSPACE}$$

Komplexitätsklassen

Frage

Wie zeigen wir, daß ein gegebenes Problem in einer bestimmten Klasse ist?

Antwort

Reduktion auf ein bekanntes!

Wir brauchen eines, mit dem wir anfangen können: SAT

Komplexitätsklassen

Frage

Können wir in **NP** Probleme finden, die **die schwierigsten in NP** sind?

Antwort

Es gibt mehrere Wege, ein schwerstes Problem zu definieren. Sie hängen davon ab, welchen **Begriff von Reduzierbarkeit** wir benutzen.

Für einen gegebenen Begriff von Reduzierbarkeit ist die Antwort: **Ja**. Solche Probleme werden **vollständig in der gegebenen Klasse** bezüglich des Begriffs der Reduzierbarkeit genannt.

Reduktion

Definition 15.7 (Polynomial-Zeit-Reduzibilität)

Seien L_1, L_2 Sprachen.

Polynomial-Zeit: L_2 ist Polynomial-Zeit reduzibel auf L_1, bezeichnet mit $L_2 \preceq_{\text{pol}} L_1$, wenn es eine **Polynomial-Zeit beschränkte DTM** gibt, die für jede Eingabe w eine Ausgabe $f(w)$ erzeugt, so daß

$$w \in L_2 \text{ gdw } f(w) \in L_1$$

Reduktion

Lemma 15.8 (Polynomial-Zeit-Reduktionen)

1. *Sei L_2 Polynomial-Zeit-reduzibel auf L_1. Dann gilt*

   L_2 *ist in* **NP** *wenn* L_1 *in* **NP** *ist*
   L_2 *ist in* **P** *wenn* L_1 *in* **P** *ist*

2. *Die Komposition zweier Polynomial-Zeit-Reduktionen ist wieder eine Polynomial-Zeit-Reduktionen.*

Theorem 15.9 (Charakterisierung von NP)

Eine Sprache L is in **NP** *genau dann wenn es eine Sprache L' in* **P** *und ein $k \geq 0$ gibt, so das für alle $w \in \Sigma$ gilt:*

$$w \in L \text{ gdw. es gibt ein } c : \langle w, c \rangle \in L' \text{ und } |c| < |w|^k.$$

c wird ***Zeuge*** *(witness oder Zertifikat/certificate) von w in L genannt. Eine DTM, die die Sprache L' akzeptiert, wird* ***Prüfer*** *(verifier) von L genannt.*

Wichtig:

Ein Entscheidungsproblem ist in **NP** genau dann wenn **jede Ja-Instanz ein kurzes Zertifikat** hat (d.h. seine Länge polynomial in der Länge der Eingabe ist), welche in polynomial-Zeit verifiziert werden kann.

Teil VI

Komplexitätstheorie

Vollständige und harte Probleme

Definition 16.1 (NP-vollständig, NP-hart)

- Eine Sprache L heißt **NP-hart (NP-schwer)** wenn jede Sprache $L' \in$ **NP** polynomial-zeit-reduzibel auf L ist.
- Eine Sprache L heißt **NP-vollständig** wenn sie
  1. in **NP** ist ($L \in$ **NP**), und
  2. **NP**-hart ist

Definition 16.2 (PSPACE-vollständig, PSPACE-hart)

- Eine Sprache L heißt **PSPACE-hart (PSPACE-schwer)** wenn jede Sprache $L' \in$ **PSPACE** polynomial-zeit-reduzibel auf L ist.
- Eine Sprache L heißt **PSPACE-vollständig** wenn sie
  1. in **PSPACE** ist ($L \in$ **PSPACE**) und
  2. **PSPACE**-hart ist

Vollständige und harte Probleme

Bemerkenswert

- Wenn gezeigt werden kann, daß auch nur ein einziges **NP**-hartes Problem in **P** liegt, dann ist **P** $=$ **NP**.
- Wenn **P** $\neq$ **NP** gilt, dann ist kein einziges **NP**-vollständiges Problem in polynomieller Zeit lösbar.

  Eine Million Euro für den, der das „P $=$ NP“-Problem löst!
  (Millenium Probleme)

Vollständige und harte Probleme

Wie zeigt man NP-Vollständigkeit?

Um zu zeigen, dass eine Sprache L **NP**-vollständig ist:

Finde bekanntermaßen **NP**-vollständige Sprache L' und reduzieren sie auf L:

$$L' \preceq L$$

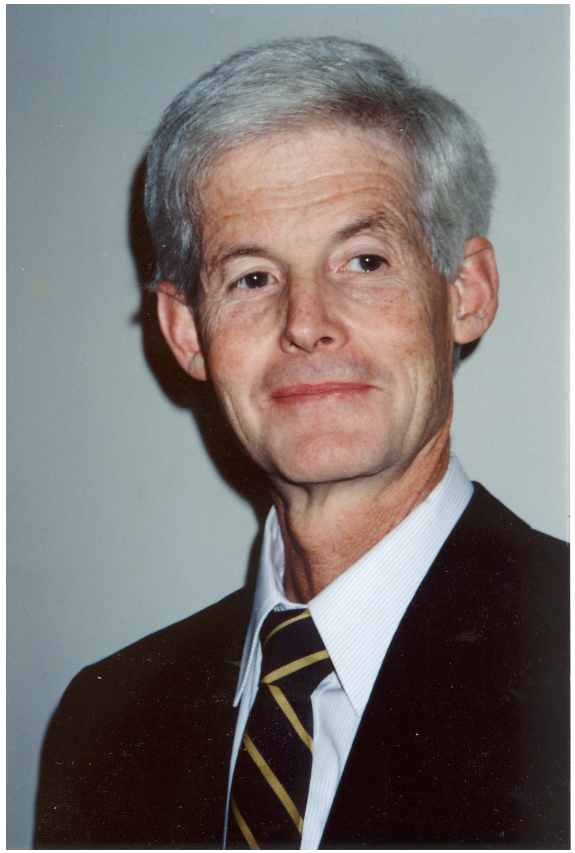
Das genügt, da jede Sprache aus **NP** auf L' reduzierbar ist und wegen $L' \preceq L$ dann auch auf L.

Hierfür häufig verwendet:
SAT-Problem, d.h.

$$L' = L_{\text{sat}} = \{w \mid w \text{ ist eine erfüllbare aussagenlogische Formel}\}$$

Satz von Cook

Stephen A. Cook ⋆1939



- Informatik-Professor and der Universität Toronto
- 1971: „Das Erfüllbarkeitsproblem der Aussagenlogik (SAT) ist NP-vollständig“
- Turing-Preisträger

Abgschlossenheit der Komplexitätsklassen

P, PSPACE abgeschlossen unter Komplement

Alle Komplexitätsklassen, die mittels **deterministischer Turing-Maschinen** definiert sind, sind **abgeschlossen unter Komplement-Bildung**

Denn:
Wenn eine Sprache L dazu gehört, dann auch ihr Komplement
(einfach die alte Maschine ausführen und die Ausgabe invertieren).

Abgschlossenheit der Komplexitätsklassen

Abgeschlossenheit von NP unter Komplement

Frage:

Ist **NP** abgeschlossen unter Komplementbildung?

Antwort:

Keiner weiß es!

Die Kompexitätsklasse co-NP

Definition 16.3 (co-NP)

co-NP ist die Klasse dr Sprachen deren Komplemente in **NP** liegen:

$$\textbf{co-NP} = \{L \mid \overline{L} \in \textbf{NP}\}$$

Beziehungen zwischen Komplexitätsklassen

Die folgenden Beziehungen sind momentan noch unbekannt

1. $\mathbf{P} \stackrel{?}{\neq} \mathbf{NP}$.
2. $\mathbf{NP} \stackrel{?}{\neq} \mathbf{co\text{-}NP}$.
3. $\mathbf{P} \stackrel{?}{\neq} \mathbf{PSPACE}$.
4. $\mathbf{NP} \stackrel{?}{\neq} \mathbf{PSPACE}$.

Teil VI

Komplexitätstheorie

NP-vollständige Probleme

Beispiel 17.1 (NP vollständige Probleme)

- Ist ein (un-) gerichteter Graph hamiltonsch? (**Hamiltonian circle**)
- Ist eine logische Formel erfüllbar? (**Satisfiability**)
- Gibt es in einem Graphen eine Clique der Größe k? (**Clique of size k**)
- Ist ein Graph mit drei Farben zu färben? (**3-colorability**)
- Gibt es in einer Menge von ganzen Zahlen eine Teilmenge mit der Gesamtsumme x? (**Subset Sum**)
- Rucksackproblem (**knapsack**)
- Multiprozessor-Scheduling

NP-vollständige Probleme

Definition 17.2 (Hamilton Circle)

Hamilton-Kreis:
Weg entlang der Kanten in einem Graphen, der jeden Knoten genau einmal besucht

$L_{\mathbf{Ham}_{\mathbf{undir}}}$: Die Sprache, die aus allen ungerichteten Graphen besteht, in denen es einen Hamilton-Kreis gibt.

$L_{\mathbf{Ham}_{\mathbf{dir}}}$: Die Sprache, die aus allen gerichteten Graphen besteht, in denen es einen Hamilton-Kreis gibt.

NP-vollständige Probleme

Definition 17.3 (Maximale Clique: L_{Clique_k})

Eine **Clique** in einem Graphen ist ein **vollständiger Teilgraph von** G.

Für $k \in \mathbb{N}$:

L_{Clique_k} Die Sprache, die aus allen ungerichteten Graphen besteht, die eine Clique der Größe k enthalten.

NP-vollständige Probleme

Definition 17.4 (k-colorability: $L_{\text{Color}_{\leq k}}$)

Ein (ungerichteter) Graph heißt **k-färbbar**, falls jeder Knoten mit einer von k Farben so gefärbt werden kann, daß benachbarte Knoten verschiedene Farben haben.

Für $k \in \mathbb{N}$:

$L_{\text{Color}_{\leq k}}$ Die Sprache, die aus allen ungerichteten, mit höchstens k Farben färbbaren Graphen besteht.

Definition 17.5 (SAT, k-CNF, k-DNF)

DNF: Eine Formel ist in **disjunktiver Normalform**, wenn sie von folgender Form ist:

$$(l_{11} \wedge \ldots \wedge l_{1n_1}) \vee \ldots \vee (l_{m1} \wedge \ldots \wedge l_{mn_m})$$

CNF: Eine Formel ist in **konjunktiver Normalform**, wenn sie von folgender Form ist:

$$(l_{11} \vee \ldots \vee l_{1n_1}) \wedge \ldots \wedge (l_{m1} \vee \ldots \vee l_{mn_m})$$

NP-vollständige Probleme

Definition (Fortsetzung)

k-DNF: Eine Formel ist in k-*DNF* wenn sie in DNF ist und jede ihrer Konjuntionen genau k Literale hat.

k-CNF: Eine Formel ist in k-*CNF* wenn sie in CNF ist und jede ihrer Disjunktion genau k Literale hat.

NP-vollständige Probleme

Theorem 17.6 (NP-vollständige Probleme)

Die folgenden Probleme liegen in **NP** *und sind* **NP***-vollständig:*

- L_{sat}
- *CNF*
- *k-CNF für* $k \geq 3$

Theorem 17.7 (Probleme in P)

Die folgenden Probleme liegen in **P***:*

- *DNF*
- *k-DNF für alle k*
- *2-CNF*

NP-vollständige Probleme

Definition 17.8 (Rucksackproblem (knapsack))

Ein Rucksackproblem besteht aus

- n Objekten mit Gewichten $g_1, \ldots, g_n$ und Nutzwerten $a_1, \ldots, a_n$
- einem Gewichtsmaximum G
- einem Nutzwertminimum A

Das Rucksackproblem ist lösbar, wenn es eine Teilmenge der gegebenen Objekte gibt, deren Gewicht nicht größer als G und deren Nutzen nicht kleiner als A ist.

$L_{\mathbf{KP}}$ Die Sprache, die aus allen lösbaren Rucksackproblemen besteht.

NP-vollständige Probleme

Definition 17.9 (Multiprozessor-Scheduling-Problem)

Ein Scheduling-Problem besteht aus

- n Prozesse mit Laufzeiten $t_1, \ldots, t_n$
- m Prozessoren
- eine Maximallaufzeit (Deadline) D

Das Schedulinig-Problem ist lösbar, wenn es eine Verteilung der Prozesse auf die Prozessoren gibt, so dass alle Prozesse beendet sind bevor D vergangen ist.

L_{schedule} Die Sprache, die aus allen lösbaren Scheduling-Problemen besteht.

Beispiel 17.10 (CNF-SAT $\preceq_{\text{pol}}$ Clique$_{\leq k}$)

Gegeben eine Instanz von CNF
(eine Konjunktion von Klauseln $C_1 \wedge C_2 \wedge \ldots \wedge C_k$)

Wir konstruieren daraus einen Graphen:

Knoten: die Paare (x, i), so dass x ein Literal ist, dass in der Klausel C_i vorkommt.

Kanten: Es gibt eine Kante zwischen (x, i) und (y, j) falls:
(1) $i \neq j$, und
(2) x, y sind nicht komplementär.

Es gilt dann:
Die CNF-Formel ist erfüllbar genau dann,
wenn der zugeordnete Graph eine Clique der Größe k hat.

Teil VI

Komplexitätstheorie

Beispielhafte Rechenzeiten

Rechenzeiten auf 3.3 Ghz Prozessor

	Größe n							
	10	20	30	40	50	60	10^3	10^6
$\log_n$	1ns	2ns		3ns			10ns	100ns
n	3ns	6ns	9ns	12ns	15ns	18ns	300ns	300μs
n^2	30ns	120ns	270ns	480ns	750ns	1.1μs	300μs	300s
n^3	300ns	2.4μs	8.1μs	19.2μs	37.5μs	64μs	300ms	9.5y
2^n	300ns	300μs	300ms	300s	83.3h	9.5y		
3^n	17.8μs	1.1s	17.3h	116y	$2.5 * 10^9$y			

Beispielhafte Rechenzeiten

Hilft ein schnellerer Computer?

Vergrößerung des in gleicher Zeit lösbaren Problems, wenn der Computer 1000 mal schneller wird

	Steigerung
$\log_n$	10^{300}-fach
n	1000-fach
n^2	32-fach
n^3	10-fach
2^n	plus 10
3^n	plus 6

Beispielhafte Rechenzeiten

Schlussfolgerungen

- Polynomielle Lösbarkeit ist entscheidend
- Exponentieller Aufwand für die Praxis (in der Regel) unakzeptabel
- Bessere Hardware für schlechte Komplexitätsklassen keine Lösung

Ungeklärte Fragen

- Macht Parallelismus / Nichtdeterminismus Probleme handhabbar?
- Zusammenhang zwischen Platzbedarf und Laufzeitverhalten?
  (Bisher nur grobe Abschätzungen bekannt)

Teil VI

Komplexitätstheorie

co-NP

Definition 19.1 (Aussagenlogische Tautologien)

L_{TAUT} Die Sprache, die aus allen aussagenlogischen Tautologien besteht

NP und co-NP

- Es ist unbekannt, ob **NP** $=$ **co-NP**
- Wenn **P** $=$ **NP**, dann **NP** $=$ **co-NP**
  (ob die Umkehrung davon gilt, ist unbekannt)
- Das komplement eines **NP**-vollständigen Problems ist **co-NP**-vollständig
- Wenn es ein Problem gibt, das sowohl **NP**-vollständig als auch **co-NP**-vollständig ist, dann ist **NP** $=$ **co-NP**
- Also: L_{TAUT} ist vermutlich nicht in **NP**
  (da es dem Komplement von SAT entspricht und also **co-NP**-vollständig ist)

Ein PSPACE-vollständiges Problem

Quantified Boolean Formulas (QBF)

Syntax Erweiterung des Aussagenlogik um Quantifizierung über aussagenlogische Variablen

Semantik

$$(\forall P)\, F \equiv F[\mathrm{TRUE}/P] \wedge F[\mathrm{FALSE}/P]$$
$$(\exists P)\, F \equiv F[\mathrm{TRUE}/P] \vee F[\mathrm{FALSE}/P]$$

L_{QBF} Die Menge aller allgemeingültigen QBF-Formeln

QBF ist das prototypische PSPACE-vollständige Problem

Ein EXPSPACE-vollständiges Problem

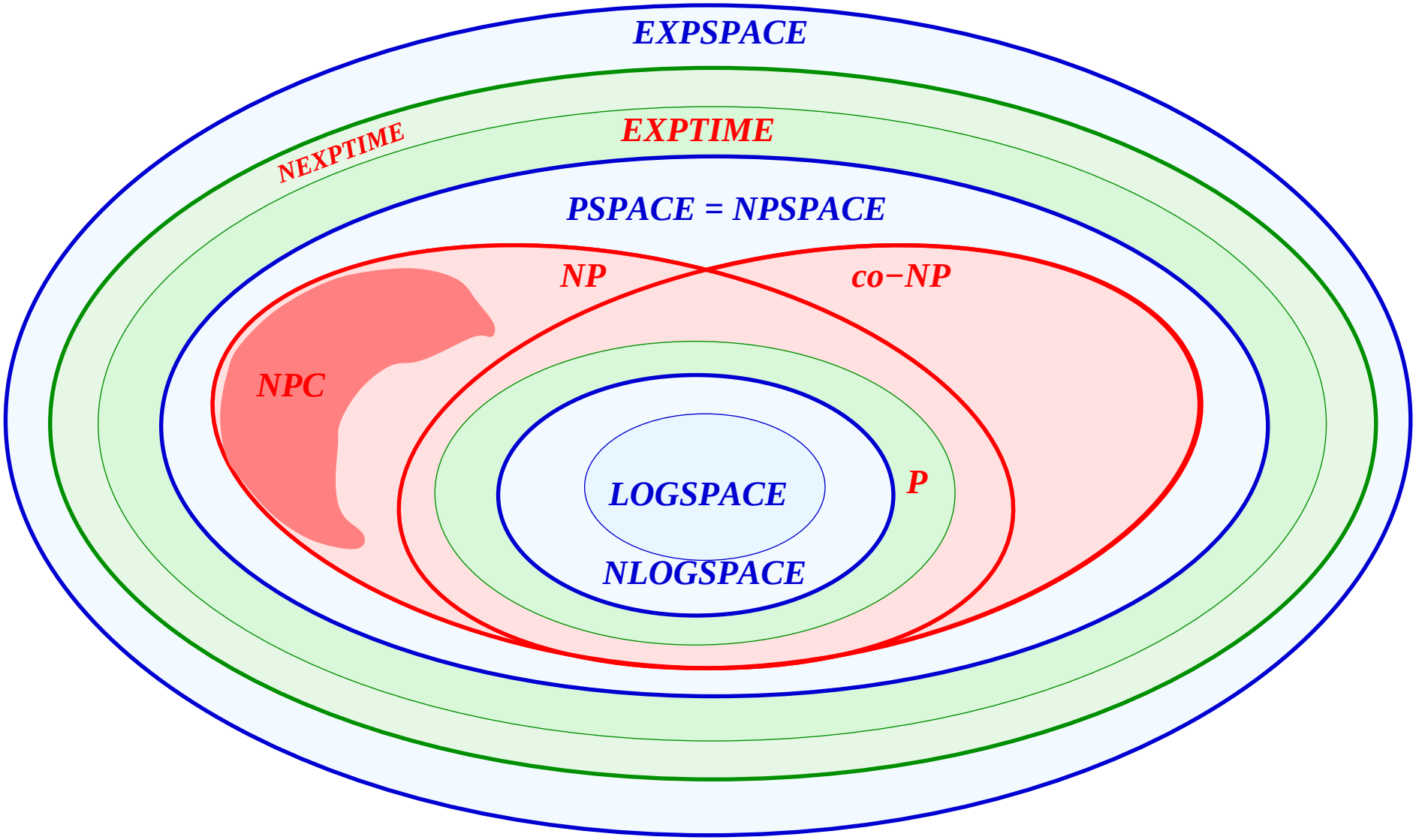
Reguläre Ausdrücke mit Iteration

Erweiterung regulärer Ausdrücke um Iteration
(neben Vereinigung, Konkatenation Kleene-Stern)

Wenn E ein regulärer Ausdruck ist, dann auch E^k $(I \in \mathbb{N})$

Das Problem zu entscheiden, ob zwei reguläre Ausdrücke mit Iteration äquivalent sind, ist EXPSPACE-vollständig.

Übersicht Komplexitätsklassen



Teil VI

Komplexitätstheorie

Pseudopolynomielle Probleme

Definition 20.1 (Pseudopolynomielles Problem)

Sei $\mathrm{MAX}(w)$ die größte der in einer Eingabe w kodierten Zahlen.

Ein Problem P heißt pseudopolynomiell, wenn es eine DTM gibt, die P löst und deren Laufzeit

$$\text{polynomiell in } |w| \text{ und } \mathrm{MAX}(w)$$

beschränkt ist.

Eigenschaft

Pseudopolynomielle Probleme sind für große Eingaben noch effizient lösbar, wenn diese nur (viele) kleine Zahlen enthalten

Pseudopolynomielle Probleme

Beispiel: Das Rucksackproblem (knapsack)

Ein Rucksackproblem P besteht aus

- n Objekten mit Gewichten $g_1, \ldots, g_n$ und Nutzwerten $a_1, \ldots, a_n$
- einem Gewichtsmaximum G
- einem Nutzwertminimum A

Das Rucksackproblem ist lösbar, wenn es eine Teilmenge der gegebenen Objekte gibt, deren Gewicht nicht größer als G und deren Nutzen nicht kleiner als A ist.

Es gibt eine DTM, die dies mit Aufwand

$$O(n * G)$$

entscheidet.

Also ist das Rucksackproblem pseudopolynomiell

Pseudopolynomielle Probleme

Beachte

Daraus, dass das Rucksackproblem in

$$O(n * G)$$

lösbar ist, folgt nicht etwa, dass es in **P** ist.

Denn die Größe des Eingabewortes, das die Instanz des Rucksackproblems kodiert, ist in

$$O(n + \log G + \log A)$$

Teil VI

Komplexitätstheorie

Motivation

Viele relevante Probleme nachweislich schwierig

Unentscheidbar Terminierung, Korrektheit von Programmen
Allgemeingültigkeit prädikatenlogischer Formeln

PSPACE-vollständig Spiele, Marktanalysen

NP-vollständig SAT, Navigation, Scheduling

Motivation

Lösungsansätze

Heuristische Lösung Nutze zusätzliches Wissen über die Struktur der in einer bestimmten Anwendung auftretenden Probleminstanzen; Verzichte auf Lösung im Allgemeinen zugunsten eines guten Average Case in der bestimmten Anwendung

Approximation Bestimme Näherungslösung
Verzichte auf optimale Antwort zugunsten kürzerer Laufzeit

Probabilistisch Bestimme richtige Lösung mit bestimmer (hoher) Wahrscheinlichkeit
Verzichte auf sichere Korrektheit der Antwort zugunsten kürzerer Laufzeit

Approximation

Viele NP-harte Probleme haben Optimierungsvariante

Knapsack Finde möglichst kleines Gewicht für festen Minimalnutzen

CLIQUE Finde möglichst große Clique in einem Graphen

Nicht alle NP-harten Probleme lassen sich gut approximativ mit polynomiellem Aufwand lösen

Knapsack Es ist möglich, eine Näherungslösung in

$$O(n^3 * \varepsilon^{-1})$$

zu finden, wobei ε der relative Fehler ist

CLIQUE Keine gute polynomielle Approximation möglich
(es sei denn **P** = **NP**)

Probabilistische Algorithmen

Idee

- Indeterministische, zufallsgesteuerte Rechnung
- Ziel dabei: Falsche Entscheidung möglich aber unwahrscheinlich
- Verringerung der Fehlerwahrscheinlichkeit durch Wiederholung der Rechnung
- Merke: 2^{-100} liegt schon unter der Wahrscheinlichkeit von Hardwarefehlern

Probabilistische Algorithmen

Beispiel: Probabilistischer Algorithmus für 3-CLIQUE

NB: 3-CLIQUE ist polynomiell lösbar (anders als allgemein CLIQUE)

Gegeben: Graph $G = (V, E)$

Wiederhole folgendes k mal:

- Wähle zufällig $v_1 \in V$ und $\{v_2, v_3\} \in E$
- Teste, ob v_1, v_2, v_3 eine 3-CLIQUE bilden

Fehlerwahrscheinlichkeit

$k = (|E| \cdot |V|)/3$: Fehlerwahrscheinlichkeit $< 0,5$

$k = 100 \cdot (|E| \cdot |V|)/3$: Fehlerwahrscheinlichkeit $< 2^{-100}$