

Komplexitätstheorie

Sommersemester 2013

Prof. Dr. Georg Schnitger
AG Theoretische Informatik

Johann Wolfgang Goethe-Universität Frankfurt am Main

Herzlich willkommen!!!

Kapitel 1: Einführung

Struktur der Veranstaltung

- 1 Der erste Teil der Veranstaltung beschäftigt sich mit **Komplexitätsklassen**. Schwierige algorithmische Probleme für
 - ▶ Speicherplatz: **PSPACE-Vollständigkeit**
 - ★ Wie schwierig ist die Bestimmung von Gewinnstrategien in 2-Personen Spielen?
 - ★ Wie schwierig ist die Bestimmung von minimalen NFAs?
 - ★ Wie mächtig sind probabilistische Berechnungen und Quantenberechnungen?
 - ▶ Parallelität: **P-Vollständigkeit**
 - ★ Welche Probleme in P besitzen keine superschnellen parallelen Algorithmen?
 - ★ Ein Zusammenhang zwischen Speicherplatz und paralleler Zeit.
 - ▶ Effiziente Approximierbarkeit: Eine vollständig neue Sichtweise von **NP**.
- 2 Im letzten Teil werden **untere Schranken für konkrete Probleme** hergeleitet: Wieviele Ressourcen sind für die die Lösung eines Problems notwendig?
 - ▶ Boolesche Funktionen: Wie tief oder wie groß müssen Schaltkreise für das **Clique-Problem** oder für das **Matching-Problem** mindestens sein? Wann müssen Schaltkreise beschränkter Tiefe für das **Paritätsproblem** groß sein?
 - ▶ Kommunikation als Beweismethode: Wieviel muss mindestens kommuniziert werden?
 - ▶ Warum ist die $P \stackrel{?}{=} NP$ Frage so schwer?

- Die Bestimmung von Eigenschaften, die ein algorithmisches Problem schwierig machen.
- Die Entwicklung von Methoden, um die Schwierigkeit eines Problems einschätzen zu können.
 - ▶ Komplexitätsklassen und Vollständigkeit.
 - ▶ Die Herleitung unterer Schranken.

Um angemessene algorithmische Lösungen erarbeiten zu können, muss man die inhärente Schwierigkeit des Problems verstehen.

Worauf wird aufgebaut?

- **Notwendig:** Eine Bachelor-Veranstaltung wie etwa die **Theoretische Informatik 1** für den Entwurf und die Analyse von Algorithmen.
 - ▶ Laufzeitanalyse: O , o , Ω , ω and Rekursionsgleichungen
 - ▶ Traversierung von Graphen
 - ▶ Dynamische Programmierung
 - ▶ NP-Vollständigkeit
 - ▶ Berechenbarkeit
- **Hilfreich, aber nicht notwendig:** Eine Bachelor-Veranstaltung wie etwa die **Theoretische Informatik 2** für formale Sprachen.
- **Spezielles mathematisches Vorwissen** wird nicht verlangt, aber die Fähigkeit, mathematische Beweise verstehen und führen zu können, sollte vorhanden sein.

- (1) S. Arora und B. Barak, Computational Complexity, a Modern Approach, Cambridge University Press, 2009.
- (2) M. Sipser, Introduction to the Theory of Computation, Paperback 3rd edition, Cengage Learning, 2012.
- (3) O. Goldreich, Computational Complexity: A Conceptual Perspective, Cambridge University Press, 2008.
- (4) Skript zur Vorlesung „Komplexitätstheorie“, Goethe-Universität Frankfurt.

- (1) Lance Fortnow und Bill Gasarch, Computational Complexity Blog, <http://blog.computationalcomplexity.org/>
- (2) The blog of Scott Aaronson, <http://www.scottaaronson.com/blog/>
- (3) R.J. Lipton, Goedel's lost letter and $P = NP$, <http://rjlipton.wordpress.com/>

- Die Webseite der Veranstaltung enthält alle wichtigen Informationen zur Veranstaltung wie Skript, Folien, Übungsblätter, Klausurtermine, usw.

www.thi.informatik.uni-frankfurt.de/KT/SoSe13/

- **Übungsbetrieb: BITTE UNBEDINGT TEILNEHMEN!**
 - ▶ Wöchentliche Ausgabe der Übungszettel vor der Vorlesung.
 - ▶ Rückgabe, nach 1-wöchiger Bearbeitungszeit, zu Beginn der Vorlesung. (Rückgabe auch im Briefkasten neben Büro 312 möglich.)
 - ▶ Werden **alle** Übungspunkte erreicht, dann Verbesserung des Ergebnisses einer mündlichen Prüfung um bis zu einer Note.

- 1 Bitte helfen Sie mir durch
 - ▶ ihre Fragen,
 - ▶ Kommentare
 - ▶ und Antworten!
- 2 Die Vorlesung kann nur durch **Interaktion** interessant werden.
 - ▶ Ich muss wissen, wo der Schuh drückt.
- 3 Sie erreichen mich außerhalb der Vorlesung im Büro 303.
 - ▶ Sprechstunde: Dienstags 10-12.
 - ▶ Kommen Sie vorbei.

Wir untersuchen algorithmische Entscheidungsprobleme nach

- der Laufzeit sequentieller Algorithmen: **Zeitkomplexität**
- dem Speicherplatz sequentieller Algorithmen: **Platzkomplexität**
- der Effizienz paralleler Algorithmen: **Schaltkreiskomplexität** und
- nach der Qualität approximativer Lösungen: **Approximationskomplexität**.

Zeit-Komplexitätsklassen

- Wir wählen Turingmaschinen als Rechnermodell.
 - ▶ Die Architektur einer Turingmaschine M besteht aus einem nach links und rechts unbeschränktem, eindimensionalem Band. Das Band ist in Zellen unterteilt und der Zelleinhalt wird von einem Lese-Schreibkopf modifiziert.
 - ▶ Während der Berechnung darf der Kopf den Inhalt der gegenwärtig besuchten Zelle (mit einem Buchstaben des Bandalphabets) überdrucken und in einem Schritt zur linken oder rechten Nachbarzelle wandern.
- Für eine Eingabe $w = (w_1, \dots, w_n) \in \Sigma^*$ ist
 - ▶ $M(w)$ die Ausgabe der Berechnung.
 - ▶ Wenn die Maschine M für jede Eingabe nur die Symbole 0 oder 1 als Ausgabe produziert, dann sagen wir, dass die Eingabe w **akzeptiert** (Ausgabe 1) bzw. **verworfen** wird (Ausgabe 0).
- M erkennt (oder akzeptiert) die **Sprache**

$$L(M) = \{w \in \Sigma^* \mid M \text{ akzeptiert } w \}.$$

Wir stoppen die Zeit

- Σ sei ein Alphabet, also eine endliche Menge von Buchstaben.
- M sei eine deterministische Turingmaschine.

(a) Wenn M für Eingabe $w \in \Sigma^*$ genau $\text{time}_M(w)$ Schritte ausführt, dann ist

$$\text{time}_M(n) = \max\{\text{time}_M(w) \mid w \in \Sigma^n\}$$

die Laufzeit von M für Eingabelänge n .

(b) Für die Funktion $t : \mathbb{N} \rightarrow \mathbb{N}$ ist

$$\text{DTIME}(t) = \{L \subseteq \Sigma^* \mid \text{es gibt eine deterministische Turingmaschine } M \text{ mit } L(M) = L \text{ und } \text{time}_M = O(t)\}.$$

(c) Die Komplexitätsklasse P besteht aus den mit polynomieller Laufzeit erkennbaren Sprachen,

$$P = \bigcup_{k \in \mathbb{N}} \text{DTIME}(n^k).$$

Die Laufzeit nichtdeterministischer Berechnungen

- M sei eine nichtdeterministische Turingmaschine.
- **Die Macht des Ratens:** M akzeptiert Eingabe w , wenn w von mindestens einer Berechnung akzeptiert wird.

$$L(M) = \{w \mid \text{es gibt eine Berechnung von } M, \text{ die } w \text{ akzeptiert} \}$$

ist die von M erkannte/akzeptierte Sprache.

- (a) M benötigt Zeit höchstens $t(n)$, wenn alle Berechnungen von M für Eingaben der Länge n höchstens $t(n)$ Schritte benötigen.
- (b) Wir setzen

$$\text{NTIME}(t) = \{L \subseteq \Sigma^* \mid \text{es gibt eine nichtdet. TM } M \text{ mit } L(M) = L \text{ und } L \text{ benötigt höchstens } O(t) \text{ Schritte} \}.$$

- (c) Die Komplexitätsklasse NP besteht aus den mit polynomieller Laufzeit erkennbaren Sprachen,

$$\text{NP} = \bigcup_{k \in \mathbb{N}} \text{NTIME}(n^k).$$

Die Methode der Diagonalisierung, Eine Zeithierarchie

Können Berechnungen **mehr Probleme** lösen,
wenn **mehr Zeit** zur Verfügung steht?

- Wir benutzen die **Diagonalisierungsmethode** von **Cantor**.
 - ▶ Cantor hat diese Methode erstmalig angewandt um zu zeigen, dass die Menge der reellen Zahlen überabzählbar groß ist.
 - ▶ In der Informatik wird die Diagonalisierung z. B. für den Nachweis der Unentscheidbarkeit der Diagonalsprache oder des Halteproblems benutzt.

Entwerfe eine Maschine mit Laufzeit $O(t(n))$, die sich von allen Maschinen mit Laufzeit $O(t(n)/\log_2 t(n))$ unterscheidet.

Problem: Wir möchten Turingmaschinen simulieren, solange die Zeitschranke $t(n)$ nicht überschritten ist.

Lösung: $t : \mathbb{N} \rightarrow \mathbb{N}$ heißt **zeitkonstruierbar**, falls $t(n) \geq n \cdot \log_2 n$ und falls es eine det. TM gibt, die für jede Eingabe x die Binärdarstellung von $t(|x|)$ in Zeit höchstens $O(t(|x|))$ berechnet.

Zeitkontrolle

- ▶ Zur Berechnung der Binärdarstellung von $t(n)$ steht Zeit $O(t)$ und damit exponentielle Zeit in der Länge der Binärdarstellung von t zur Verfügung.
- ▶ Initialisiere einen Zähler mit Wert $t(n)$ in Zeit $O(t(n))$.
- ▶ Halte den Zähler stets in der Nähe des Kopfes.
⇒ **Zeitkontrolle in Zeit $O(\log_2 t(n))$ pro Schritt.**

- (a) Die Funktion t sei zeitkonstruierbar.
Dann ist $\text{DTIME}(o(\frac{t}{\log_2 t}))$ eine echte Teilmenge von $\text{DTIME}(t)$.
- (b) \mathbf{P} ist eine echte Teilmenge von $\mathbf{E} = \bigcup_{k \in \mathbb{N}} \text{DTIME}(n^k)$.

Wir bauen eine Maschine M^ mit Laufzeit $O(t)$, die sich von allen Maschinen M mit Laufzeit $o(\frac{t}{\log_2 t})$ unterscheidet.*

- 1 M^* bestimmt die Länge n der Eingabe w und speichert die Binärdarstellung von $\frac{t(n)}{\log_2 t(n)}$ in einem Zähler ab.
/* Dies ist mit Laufzeit $O(t(n))$ möglich, da t zeitkonstruierbar ist. */
- 2 Wenn $w \neq \langle M \rangle 0^k$ für eine Turingmaschine M ist, verwirft M^* .
/* $\langle M \rangle$ bezeichnet die Gödelnummer der Turingmaschine M . */
- 3 M^* simuliert M auf der Eingabe $\langle M \rangle 0^k$ und verwirft, wenn die Simulation mehr als $\frac{t(n)}{\log_2 t(n)}$ Schritte benötigt.
- 4 M^* **akzeptiert** w , wenn M **verwirft**.
Wenn M hingegen **akzeptiert**, dann wird M^* **verwerfen**.

Zeitkomplexität: Die wichtigen Fragestellungen

? $P \stackrel{?}{=} NP$?

- (*) In welchem Ausmaß kann die Methode der Diagonalisierung benutzt werden?
 - (*) Ist die Frage möglicherweise mit heutigen Methoden nicht beantwortbar?
 - (*) In welchen eingeschränkten Modellen von P und NP können wir die Frage beantworten?
- ? Die NP-Vollständigkeit ist unzureichend, um die **Approximationskomplexität** wichtiger Optimierungsprobleme zu klären. Wir brauchen eine neue Sichtweise von NP.
- ? Wie sehen Querbezüge zwischen **Zeit- und Platzklassen** aus? (Mehr später.)
- ? Wie sehen Querbezüge zwischen **sequentiellen und parallelen** Zeitklassen aus? (Mehr später.)

Speicherplatz-Komplexität

I-O Turingmaschinen

Eine **I-O Turingmaschine** M besitzt drei ein-dimensionale Bänder mit jeweils einem Kopf, wobei jeder Kopf in einem Schritt (höchstens) zur linken oder rechten Nachbarzelle der gegenwärtig besuchten Zelle wandern darf.

- 1 Das erste Band ist das **read-only Leseband**, das die Eingabe speichert.
- 2 Das zweite Band ist das **read-write Arbeitsband**, das aus $s(n)$ Zellen besteht.
- 3 Das dritte Band ist das **write-only Ausgabeband**. (Wir interpretieren das Drucken einer 0 (1), als das Verwerfen (Akzeptieren) der Eingabe).

Die maximale Anzahl der während der Berechnung für Eingabe w besuchten Zellen bezeichnen wir mit **$DSPACE_M(w)$** .

Wir messen den Speicherplatz

M sei eine I-O Turingmaschine.

- (a) Der Speicherplatzbedarf von M für Eingabelänge n ist

$$\text{DSPACE}_M(n) = \max\{\text{DSPACE}_M(w) \mid w \in \Sigma^n\}.$$

- (b) Für $s : \mathbb{N} \rightarrow \mathbb{N}$ ist

$$\text{DSPACE}(s) = \{L \subseteq \Sigma^* \mid L(M) = L \text{ für eine I-O TM } M \text{ mit } \text{DSPACE}_M = O(s)\}.$$

die Klasse aller auf Platz $O(s(n))$ lösbaren Entscheidungsprobleme.

- (c) Die Komplexitätsklasse DL besteht aus allen mit logarithmischem Speicherplatzbedarf erkennbaren Sprachen, also

$$\text{DL} = \text{DSPACE}(\log_2 n).$$

- (d) Die Komplexitätsklasse PSPACE besteht aus allen mit polynomielltem Speicherplatzbedarf erkennbaren Sprachen, also

$$\text{PSPACE} = \bigcup_{k \in \mathbb{N}} \text{DSPACE}(n^k).$$

Platzbedarf nichtdeterministischer Rechnungen

Für eine nichtdeterministische TM M ist $\text{NSPACE}_M(w)$ für eine Eingabe w der **maximale** Speicherplatzbedarf einer Berechnung für w .

- (a) Der Speicherplatzbedarf von M für Eingabelänge n ist

$$\text{NSPACE}_M(n) = \max\{\text{NSPACE}_M(w) \mid w \in \Sigma^n\}.$$

- (b) Die Klasse aller mit Speicher $O(s)$ lösbaren Probleme ist

$$\text{NSPACE}(s) = \{L \subseteq \Sigma^* \mid \text{es gibt eine nichtdeterministische I-O TM } M \text{ mit } L(M) = L \text{ und } \text{NSPACE}_M = O(s)\}.$$

- (c) Die Komplexitätsklasse NL besteht aus allen nichtdeterministisch, mit logarithmischem Speicherplatzbedarf erkennbaren Sprachen, also

$$\text{NL} = \text{NSPACE}(\log_2 n).$$

- (d) Die Komplexitätsklasse NPSPACE besteht aus allen nichtdeterministisch, mit polynomielltem Speicherplatzbedarf erkennbaren Sprachen, also

$$\text{NPSPACE} = \bigcup_{k \in \mathbb{N}} \text{NSPACE}(n^k).$$

Eine Speicherplatz-Hierarchie

Können Berechnungen **mehr Probleme** lösen,
wenn **mehr Platz** zur Verfügung steht?

- Unser Vorgehen für die Zeit-Hierarchie lässt sich mit Leichtigkeit für die Speicherplatz-Komplexität verwenden.

Entwerfe eine Maschine mit Speicherplatz $O(s(n))$, die sich von allen Maschinen mit Speicherplatz $o(s(n))$ unterscheidet.

- Wir müssen einen passenden Begriff für Platzkonstruierbarkeit einführen:

$s : \mathbb{N} \rightarrow \mathbb{N}$ heißt genau dann **platzkonstruierbar**, wenn es eine det. TM gibt, die für eine jede Eingabe der Länge n höchstens $O(s(n))$ Speicherplatz benötigt, um $s(n)$ Zellen zu markieren.

Die Funktion s sei platzkonstruierbar.

Dann ist $DSPACE(o(s))$ eine echte Teilmenge von $DSPACE(s)$.

Wir bauen eine Maschine M^ mit Speicherplatz $O(s)$, die sich von allen Maschinen M mit Speicherplatz $o(s)$ unterscheidet.*

- 1 Für Eingabe w steckt M^* auf dem Arbeitsband $2s(|w|)$ Zellen ab.
/* Dies gelingt mit Speicherplatz $O(s(n))$, denn s ist platzkonstruierbar. */
- 2 Wenn $w \neq \langle M \rangle 0^k$ für I-O Turingmaschinen M und $k \in \mathbb{N}$ ist, dann verwirft M^* .
/* $\langle M \rangle$ bezeichnet die Gödelnummer der Turingmaschine M . */
- 3 M^* simuliert M auf Eingabe $w = \langle M \rangle 0^k$. Zu Anfang ist der Kopf in der Mitte des abgesteckten Bereichs. M^* verwirft, wenn M irgendwann den abgesteckten Bereich verlässt oder mehr als $2^{s(n)}$ Schritte benötigt.
/* M^* kann auf Speicherplatz $s(n)$, bei entsprechend vergrößerten Alphabet, gleichzeitig bis $2^{s(n)}$ zählen und M simulieren. */
- 4 M^* akzeptiert w genau dann, wenn M verwirft.

Speicherkomplexität: Die wichtigen Fragestellungen

? Ist **DL** eine echte Teilmenge von **NL**?

- (*) Wie sehen die für **DL** schwierigsten Probleme in **NL** aus?
- (*) Für welches kleinste s gilt

$$NL \subseteq DSPACE(s).$$

- (*) Wenn $L \in NL$, ist dann auch $\bar{L} \in NL$, also
ist **NL abgeschlossen unter Komplement**?

? Ist **NL** eine Teilmenge von **P**, und wenn ja, ist **NL** eine echte Teilmenge?

? Ist **P** eine echte Teilmenge von **PSPACE**?

- (*) Wie sehen die für **P** schwierigsten Probleme in **PSPACE** aus?
- (*) Ist **NP** \subseteq **PSPACE**? Kann man allgemeine probabilistische Berechnungen, die in polynomieller Zeit ablaufen, in **PSPACE** simulieren?
- (*) Lassen sich Quantenberechnungen in **PSPACE** simulieren?

? Wie ordnen sich die Klassen regulärer, kontextfreier und kontextsensitiver Sprachen in die Speicherplatzklassen ein?

- ▶ Wann ist ein algorithmisches Problem parallelisierbar?
- ▶ Was soll „parallelisierbar“ überhaupt bedeuten?

- Ein Schaltkreis S wird beschrieben durch

$$S = (G, Q, R, \text{gatter}, n, \text{eingabe}).$$

- ▶ $G = (V, E)$ ist ein gerichteter, azyklischer Graph.
 - ▶ Q ist die Menge der Quellen und R die Menge der Senken von G .
 - ▶ Die Funktion **eingabe**: $Q \rightarrow \{1, \dots, n\}$ füttert Quellen mit Eingabebits.
 - ▶ **gatter**: $V \setminus Q \rightarrow \{\neg, \vee, \wedge\}$ weist jedem „inneren“ Knoten ein Gatter zu.
- Der Schaltkreis berechnet die Boolesche Funktion

$$f_S : \{0, 1\}^n \rightarrow \{0, 1\}^{|R|},$$

- ▶ Die n Eingabebits werden an die Quellen in G angelegt.
- ▶ Jeder Knoten leitet das Ergebnis seines Gatters weiter.
- ▶ Das Resultat wird an den Senken von G abgelesen.

Größe und Tiefe von Schaltkreisen

(a) Für $S = (G, Q, R, \text{gatter}, n, \text{eingabe})$ ist

- die **Tiefe** von S die Länge des längsten Weges in G
- und die **Größe** von S die Anzahl der Knoten von G .

(b) Für eine **Boolesche Funktion** $f : \{0, 1\}^n \rightarrow \{0, 1\}$ definieren wir die minimale Tiefe

$$\text{DEPTH}(f) = \min\{ d \mid \text{Es gibt einen Schaltkreis der Tiefe } d \text{ für } f \}$$

und die minimale Größe

$$\text{SIZE}(f) = \min\{ s \mid \text{Es gibt einen Schaltkreis der Größe } s \text{ für } f \},$$

wobei wir jeweils nur Schaltkreise mit den Gattern \wedge, \vee, \neg vom Fanin zwei betrachten.

Schaltkreiskomplexität: Die wichtigen Fragestellungen

- ? Hat jede Funktionsfamilie $f_n : \{0, 1\}^n \rightarrow \{0, 1\}$ in **P** Schaltkreise polynomieller Größe?
 - (*) Wenn „ja“ und wenn $\text{SIZE}(f_n)$ stärker als polynomiell für irgendeine Funktionsfamilie in **NP** wächst, dann $P \neq NP$.
- ? Wie weit ist man? Bisher ist keine Funktionsfamilie f_n in **NP** bekannt mit
 - (*) $\text{DEPTH}(f_n) = \omega(\log_2 n)$ oder
 - (*) $\text{SIZE}(f_n) = \omega(n)$.

Die **meisten** Funktionen benötigen sowohl $\text{DEPTH}(f_n) = \Omega(n)$ wie auch $\text{SIZE}(f_n) = \omega(2^n/n)$!

- ? Eingeschränkte Schaltkreismodelle:
 - (*) Erhalten wir bessere untere Schranken für **monotone Schaltkreise**, wenn wir also die Negation verbieten?
 - (*) Und wenn wir nur **Schaltkreise sehr kleiner Tiefe**, aber mit eingeschränktem Fanin betrachten? (Zusammenhang zur Ausdrucksstärke logischer Kalküle).
- ? Welche Probleme in **P** lassen sich parallelisieren?
 - (*) Gibt es im Hinblick auf die Parallelisierung schwierigste Probleme in **P**?
 - (*) Gibt es einen Zusammenhang zwischen Parallelisierbarkeit und Speicherkomplexität?