Laufzeitkomplexität und die Klassen $\mathcal{P}$ und $\mathcal{NP}$

Prof. Dr. David Sabel

LFE Theoretische Informatik

Letzte Änderung der Folien: 10. Juli 2022

Komplexitätstheorie

Was ist Komplexitätstheorie?

- Teilgebiet der Theoretischen Informatik
- Grobes Thema: Komplexität von **entscheidbaren Problemen**
- Maße zum Messen des Ressourcenbedarfs von Algorithmen:
  - Rechenzeit
  - Platzbedarf
  - ...
- Komplexität eines Problems
  = Komplexität des besten Algorithmus bezüglich des Maßes
- Ziel: Einordnung von Problemen in Komplexitätsklassen

Komplexitätstheorie: Teilbereiche

1. Nachweis von oberen Schranken:
   - Finde möglichst guten Algorithmus für konkretes Problem
   - Analysieren der Laufzeit- und Platzkomplexität
   - z.B.: Der CYK-Algorithmus liefert obere Schranke $O(|P| \cdot n^3)$ für das Wortproblem für CFGs (mit n = Wortlänge, P Produktionen der CFG)
   - Möglichst genaue Schranken (bezüglich der O-Notation)

Komplexitätstheorie: Teilbereiche (2)

2. Nachweis von unteren Schranken:
   - Zeige, dass es keinen besseren Algorithmus gibt.
   - Schwieriger, da man über alle Algorithmen argumentieren muss.
   - Möglichst genaue Schranken (bezüglich der Ω-Notation)
   - Oft ist $\Omega(n)$ (n = Größe der Eingabe) eine untere Schranke für die Laufzeit, da man die Eingabe lesen muss.

Komplexitätstheorie: Teilbereiche (3)

3. Auswirkung der **Maschinenmodelle auf den Ressourcenbedarf**
   - insbesondere **Determinismus vs. Nichtdeterminismus**
   - Wichtige ungelöste Frage: Das $\mathcal{P}$ **vs.** $\mathcal{NP}$**-Problem**
   - Komplexitätsklassen sind i.A. ungenauer (größer) als in den vorher genannten Teilbereichen
   - **Wir beschäftigen uns in diesem Abschnitt hiermit.**

Zeitkomplexität: Annahmen und Festlegungen

Turingmaschinen:

- Wir betrachten nur **entscheidbare** Sprachen
- Deswegen nehmen wir an:

  Die Turingmaschinen, welche diese Sprachen entscheiden, **halten auf jeder Eingabe an**:
- DTMs: Wir nehmen an, dass es „Verwerfe“-Zustände gibt, für die die TM keine Nachfolgekonfiguration besitzen.
- NTMs: Haben solche Zustände von Haus aus

Zeitkomplexität: Annahmen und Festlegungen (2)

Mehrband- vs. Einband-Turingmaschinen:

- Wir nehmen Mehrband-TMs, sie passen zu „normalen“ Rechnern.
- Kein „Vergeuden“ von Rechenzeit, nur um die Eingaben zu suchen.
- Unterschied:
  n-Schritte auf Mehrband-TM können in
  $O(n^2)$ Schritten auf Einband-TM ausgeführt werden
- Unterschied macht sich in Komplexitätsklassen $\mathcal{P}$ und $\mathcal{NP}$ **nicht** bemerkbar (siehe später)

Deterministische Laufzeit

Definition ($time_M$)

Sei M eine stets anhaltende DTM mit Startzustand z_0.
Für Eingabe w definieren wir

$time_M(w) := i$, wenn $z_0 w \vdash_M^i uzw$ und
z ist Endzustand oder Verwerfe-Zustand.

Definition (Klasse $TIME(f(n))$)

Für eine Funktion $f : \mathbb{N} \to \mathbb{N}$ sei die Klasse $TIME(f(n))$ genau die Menge der Sprachen, für die es eine deterministische, stets anhaltende, Mehrband-TM M gibt, mit $L(M) = L$ und $time_M(w) \leq f(|w|)$ für alle $w \in \Sigma^*$

Sprache ist daher in $TIME(f(n))$, wenn sie von einer DTM für jede Eingabe der Länge n in $\leq f(n)$ Schritten entschieden wird.

Die Klasse $\mathcal{P}$

Definition (Polynom)

Ein Polynom ist eine Funktion $p : \mathbb{N} \to \mathbb{N}$ der Form:

$$p(n) = \sum_{i=0}^{k} a_i \cdot n^i = a_k n^k + a_{k-1} n^{k-1} + \cdots + a_1 n + a_0$$

mit $a_i \in \mathbb{N}$ und $k \in \mathbb{N}$.

Definition (Komplexitätsklasse $\mathcal{P}$)

Die Klasse $\mathcal{P}$ ist definiert als

$$\mathcal{P} = \bigcup_{p \text{ Polynom}} TIME(p(n))$$

Nochmal: Zugehörigkeit zu $\mathcal{P}$

Formale Sprache L ist in Klasse $\mathcal{P}$ enthalten, wenn:

Es gibt stets anhaltende DTM M und ein Polynom p mit:

- $L = L(M)$
- $\forall w \in \Sigma^* : time_M(w) \leq p(|w|)$

Polynomielle Komplexität

Nachweis der polynomiellen Komplexität:

Zeige, dass die Komplexität $O(n^k)$ für Konstante k ist.

Beispiele: Algorithmus mit Laufzeit

- $n \log n$ hat polynomielle Komplexität, da $n \log n \in O(n^2)$
- 2^n hat keine polynomielle Komplexität
- $n^{\log n}$ hat keine polynomielle Komplexität

Sprechweisen:

- Algorithmus ist effizient = Algorithmus hat polynomielle Komplexität
- Analog: Problem ist effizient lösbar = Problem in $\mathcal{P}$

Polynomielle Komplexität: Andere Modelle (nur FSK)

Sei M eine DTM mit $time_M(w) \leq f(|w|)$.

Analyse unserer Äquivalenzbeweise zeigt:

- TMs können durch GOTO-Programme simuliert werden, konstant viele Schritte, um 1 TM-Schritt zu simulieren
- GOTO in WHILE: Anzahl der Durchläufe durch die einzige WHILE-Schleife nicht größer als $f(|w|)$
- WHILE kann u.U. durch LOOP ersetzt werden: $y := f(|w|)$; **LOOP** y **DO** ... **END**, wenn f LOOP-berechenbar

Für f LOOP-berechenbar: Funktion mit Laufzeitkomplexität $f(n)$ LOOP-berechenbar.

Daher $TIME(2^n)$ oder sogar $TIME(\underbrace{2^{2^{\cdots^2}}}_{n\text{-mal}})$ noch in der Klasse der LOOP-berechenbaren Funktionen.

Kostenmaße

Konsequenz der vorherigen Aussagen:

I.a. asymptotisch kein Unterschied, ob Turingmaschine, WHILE-Programm oder GOTO-Programm

Vorsicht: Falle!
In $x_i := x_j$ dürfen die Zahlen nicht **zu groß** werden (bei uniformen Kostenmaß)

Uniformes Kostenmaß:
Ausführung von $x_i := x_j$ in konstanter Zeit (1 Schritt)

Logarithmisches Kostenmaß:
Ausführung von $x_i := x_j$ in Anzahl der Bits ($|\log x_j|$-Schritte)
Turingmaschine muss so viele Bits kopieren!
Daher bei großen Zahlen logarithmisches Kostenmaß verwenden.

Beispiel

WHILE-Programm

$x_0 := 2;$
WHILE $x_1 \neq 0$ **DO**
  $x_0 := x_0 * x_0;$
  $x_1 := x_1 - 1;$
END

berechnet $2^{2^{x_1}}$ in x_0.

- Uniformes Kostenmaß: Laufzeit $O(x_1)$
- Turingmaschine: mindestens 2^{x_1} Schritte alleine um das Ergebnis auf das Band zu schreiben.
- Hier also: Logarithmisches Kostenmaß verwenden!

Unser Vorgehen:
logarithmisches Kostenmaß wenn notwendig, sonst uniformes Kostenmaß

Nichtdeterminismus

Auf für NTMs nehmen wir an, dass sie auf allen Berechnungspfaden anhalten.
Beachte: Ein Wort wird **nicht akzeptiert**, wenn auf allen Berechnungspfaden verworfen wird

Definition ($ntime_M$)
Sei M eine Mehrband-NTM, die für jede Eingabe anhält. Dann definieren wir die Laufzeit von M als

$$ntime_M(w) = \max\{i \mid z_0 w \vdash_M^i uzw \text{ und } uzw \nvdash \ldots\}$$

Beachte: Schöning verwendet eine andere Definition (mit Minimum), es macht für die Definition der Klasse $\mathcal{NP}$ aber keinen Unterschied.

$NTIME$ und $\mathcal{NP}$

Definition
Für eine Funktion $f : \mathbb{N} \to \mathbb{N}$ bezeichne $NTIME(f(n))$ die Klasse aller Sprachen L, für die es eine nichtdeterministische Mehrband-TM M gibt mit $L(M) = L$ und für alle $w \in \Sigma^*$ gilt $ntime_M(w) \leq f(|w|)$.

Definition (Klasse $\mathcal{NP}$)
Die Klasse $\mathcal{NP}$ ist definiert als

$$\mathcal{NP} = \bigcup_{p \text{ Polynom}} NTIME(p(n))$$

$\mathcal{P} \subseteq \mathcal{NP}$

Lemma

Es gilt $TIME(f(n)) \subseteq NTIME(f(n))$ und damit auch $\mathcal{P} \subseteq \mathcal{NP}$.

Beweis:

- DTM als NTM auffassen
- ergibt NTM mit einem möglichen Berechnungspfad
- $time_M(w) = ntime_M(w)$
- $L \in TIME(f(n)) \implies L \in NTIME(f(n))$
- $\mathcal{P} \subseteq \mathcal{NP}$

$\mathcal{P}$-vs.-$\mathcal{NP}$

Die Frage

Gilt $\mathcal{P} = \mathcal{NP}$ oder $\mathcal{P} \neq \mathcal{NP}$?

ist **ungelöst**!

- Das $\mathcal{P}$-vs-$\mathcal{NP}$-Problem ist eines der sieben sogenannten Millennium-Probleme, die vom Clay Mathematics Institute im Jahr 2000 als Liste ungelöster Probleme der Mathematik herausgegeben wurde
- Für dessen Lösung wurde ein Preisgeld von einer Million US Dollar ausgelobt

$\mathcal{P}$-vs.-$\mathcal{NP}$ (2)

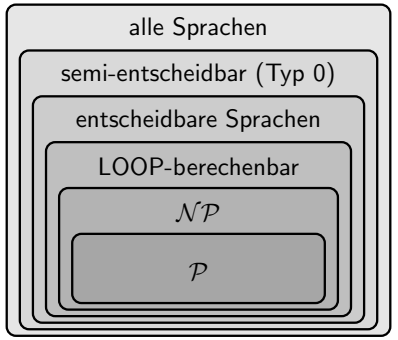
Wesentlicher Grund, der für $\mathcal{P} \neq \mathcal{NP}$ spricht:

> Man müsste für $\mathcal{P} = \mathcal{NP}$ einen Polynomialzeitalgorithmus finden, für ein Problem, für das bisher nur (deterministische) Exponentialzeitalgorithmen bekannt sind. (*)

Viele haben gesucht, keiner hat einen solchen Algorithmus gefunden.

(*) Begründung: Folgt später

Lage der Komplexitätsklasse (Schema)



Dabei unklar, ob $\mathcal{NP} = \mathcal{P}$