Komplexitätstheorie – eine erste Übersicht

KTV bedeutet: Details erfahren Sie in der **K**omplexitäts**t**heorie-**V**orlesung.

Probleme

Problem = Menge von unendlich vielen konkreten **Einzelfragen** (**Instanzen**) F_n,
(genauer: von abzählbar unendlich vielen KTV)

Beispiele

- Wie viel ist n^2
  für n=0,1,2,… ?
- Ist φ erfüllbar
  für $\varphi \in$ Menge aller abgeschlossenen Prädikatenlogik-Formeln erster Stufe?

Einzelfragen fragen nach einem **Funktionswert** f(n) für ein **Funktionsargument** a_n.

Eine Variante
Es könnte **mehrere zulässige** Antworten $f_i(n)$ geben, i=1,2,… .
Dann ist f keine Funktion sondern eine linkstotale Relation.

Berechenbarkeitstheorie

… untersucht für **Problemklassen**

Ist ein gegebenes Problem (oder jedes Problem aus einer gegebenen Klasse) **algorithmisch lösbar**?

Das heißt: Gibt es einen **Algorithmus**, der für alle Einzelfragen des Problems (natürlich in jeweils endlich vielen Schritten) die Antwort berechnet?

Bei der **Variante**: Der Algorithmus soll stets eine der zulässigen Antworten finden.

Komplexitätstheorie

… untersucht: Mit welchem Aufwand Ist ein gegebenes lösbares Problem lösbar?

Aufwand

Platzbedarf Anzahl der Rechenschritte
Raumbedarf Anzahl der Speicherplätze
bei der algorithmischen Lösung von Probleminstanzen
in Abhängigkeit von der Instanzengröße.

Größe einer Instanz
eine „plausible Größenangabe“, z.B.

- für n=0,1,2,…: n oder
  $\log_2(n) \approx$ Länge der Binärzahl n KTV
- für die Formel φ: die Länge $|\varphi|$.

Praktisches Ziel der Komplexitätstheorie
Identifikation der praktisch effizient lösbaren Probleme.

Entscheidungsprobleme (1)

Traditionell untersucht KT meist **Entscheidungsprobleme**,
deren Instanzen eine Ja/Nein-Antwort erfordern.
Warum ist das keine übermäßige Einschränkung? KTV

Man drückt jede Probleminstanz als **Wort** über einem
Alphabet Σ aus, d. h. als Zeichenfolge über Σ.
$\Sigma=\{0,1\}$ wird bevorzugt.

Beispiel

Einen bildlich gegebenen Graphen codiert man
als Wort aus Knotennamen, Kommata, und Mengenklammern –
oder als sog. Adjazenzmatrix aus 0/1-Einträgen.

Der **Lösung des Problems** (d.h. aller seiner Instanzen) entspricht die
Sprache aller Wörter, die eine Instanz mit der Antwort „Ja“ codieren.

Entscheidungsprobleme (1)

Hier also:
Problem = Wortproblem

Beispiel

Problem	=	Quadrieren von Zahlen:	gegeben n,	gesucht n^2
Wortproblem	=	Ist x Quadrat von n?:	gegeben (n,x),	gesucht Ja/Nein

Unterschied

Auch mit einem Lösungsalgorithmus des obigen Wortproblems müsste man immer noch die Antworten auf (n,0), (n,1), …, (n,n^2) berechnen,
um nach dem ersten „Ja“ dann endlich n^2 wirklich zu kennen.

Wieso tut das aufwandsmäßig nicht weh? → KTV.

Komplexität

Verhalten (**Komplexität**) eines Lösungsalgorithmus

Wachstumsverhalten des Aufwands bei wachsender Instanzengröße

Schwierigkeit (**Komplexität**) eines Problems

Wachstumsverhalten des Aufwands bei wachsender Instanzengröße

bei Verwendung der effizientesten **bekannten** Lösungsalgorithmen.

Wegen zunehmender Rechnerleistungen

(Speicherplatz, Geschwindigkeit) fragt man nach

asymptotischem Ressourcenverbrauch bei

grenzenlos wachsender Instanzengröße.

Steigerungstypen

Quotient Ressourcenaufwand/Instanzengröße

Steigt der Aufwand (in Relation zur Größe) asymptotisch z.B.

- **linear** $O(n)$
- **polynomiell**, $O(n^k)$ für ein k
- **exponentiell** $O(k^n)$ für ein k
- oder gar **überexponentiell**?

O-Notation, vgl. auch o-Notation: → KTV

Die Funktion f in Klammern bei O(f) ist die **Schrankenfunktion**.

Der Aufwand kann sich verringern durch neu gefundene effizientere Algorithmen (**Stand der Kunst!**) – es sei denn es kann **erwiesenermaßen nicht schneller** gehen.

Falltypen

Bei Instanzen gleicher Größe (oft unendlich vielen)
kann es noch weitere Faktoren geben, die den Aufwand beeinflussen.
Der Aufwand kann dann (trotz gleicher Größe) sehr unterschiedlich ausfallen.

Man fragt für einen gegebenen Algorithmus oft nach dem

- **besten** Fall: Wie arbeitet der Algorithmus (in Bezug auf die in Frage stehende Ressource) im günstigsten Fall?
- **schlechtesten** Fall
- **durchschnittlichen** Fall
  (Achtung – hierbei benötigte Info:
  Welche Instanzen kommen mit welcher Häufigkeit vor? (**Verteilung**)

Berechnungsmodelle

Die Komplexität-Eingruppierung ist i.a. unabhängig von dem betrachteten **Berechnungsmodell**:

- Registermaschine (z.B. RAM)
- Turing-Maschine (mit x Bändern)
- Kellerautomat (mit x>1 Kellern)
- …

Erweiterte Church-Turing-These:
Alle universellen Maschinenmodelle sind in Bezug auf die Rechenzeit bis auf polynomielle Faktoren gleich mächtig.

Polynomieller Aufwand (z.B. beim Vergleich von Berechnungsmodellen) wird meist als **geringfügig** bzw. **praktisch lösbar** behandelt.

Aspekte von Komplexitätsklassen

- **Berechnungsmodell** (Turingmaschine, Registermaschine usw.).
- **Berechnungsmodus** (deterministisch, nichtdeterministisch, probabilistisch usw.).
- **Berechnungsressource** (Zeit, Platz usw.)
- **Kostenmaß** (uniform, logarithmisch)
- **Schrankenfunktion** f.
  Beispielsweise ist DTIME(f) die Klasse aller Probleme, die auf einer deterministischen Turingmaschine in der Zeit O(f) entschieden werden können.
  (f = Schrankenfunktion)

Details: → KTV

Einige Komplexitätsklassen

… für Sprachen deren Wortproblem mit
(i.a. deterministischer) Turingmaschine gelöst wird.

- **DSPACE(f(n))**: auf O(f(n)) Platz lösbar.
- **DTIME(f(n))**: in O(f(n)) Schritten lösbar.
- **P**: in Polynomialzeit lösbar: DTIME(n) $\cup$ DTIME(n^2) $\cup$ DTIME(n^3) $\cup$ … .
- **PSPACE(f(n))**: auf polynomiellem Platz lösbar: DSPACE(n) $\cup$ DSPACE(n^2) $\cup$ … .
- **EXPTIME**: in Exponentialzeit lösbar → KTV
- **NP**: mit **nichtdeterministischer** Turingmaschine in Polynomialzeit lösbar,
  grob gesagt Probleme, deren Lösungen von Instanzen
  in polynomial vielen Schritten **überprüft** werden können.

usw. → KTV

Schwere und Vollständigkeit

Ist K eine Komplexitätklasse, so nennt man ein Problem P ...

- **K-schwer**, wenn ihr Lösungsalgorithmus für jedes Problem Q in K mit höchstens polynomialem Zusatzaufwand zur Lösung von Q verwendet werden kann, und

- **K-vollständig**, wenn P K-schwer ist und in K liegt.

NP-Vollständigkeit

Besonders viele vollständige Probleme wurden für NP identifiziert:
Viele (!) **NP-vollständige** Beispiele finden sich auf
https://en.wikipedia.org/wiki/List_of_NP-complete_problems

Bereiche mit besonders vielen NP-vollständigen Problemen:

- Graphentheorie
- Optimierung
- formale Sprachen
- Spiele und Rätsel
- Logik

Die folgende Frage ist bislang ungeklärt:

P=NP?