Theoretische Informatik

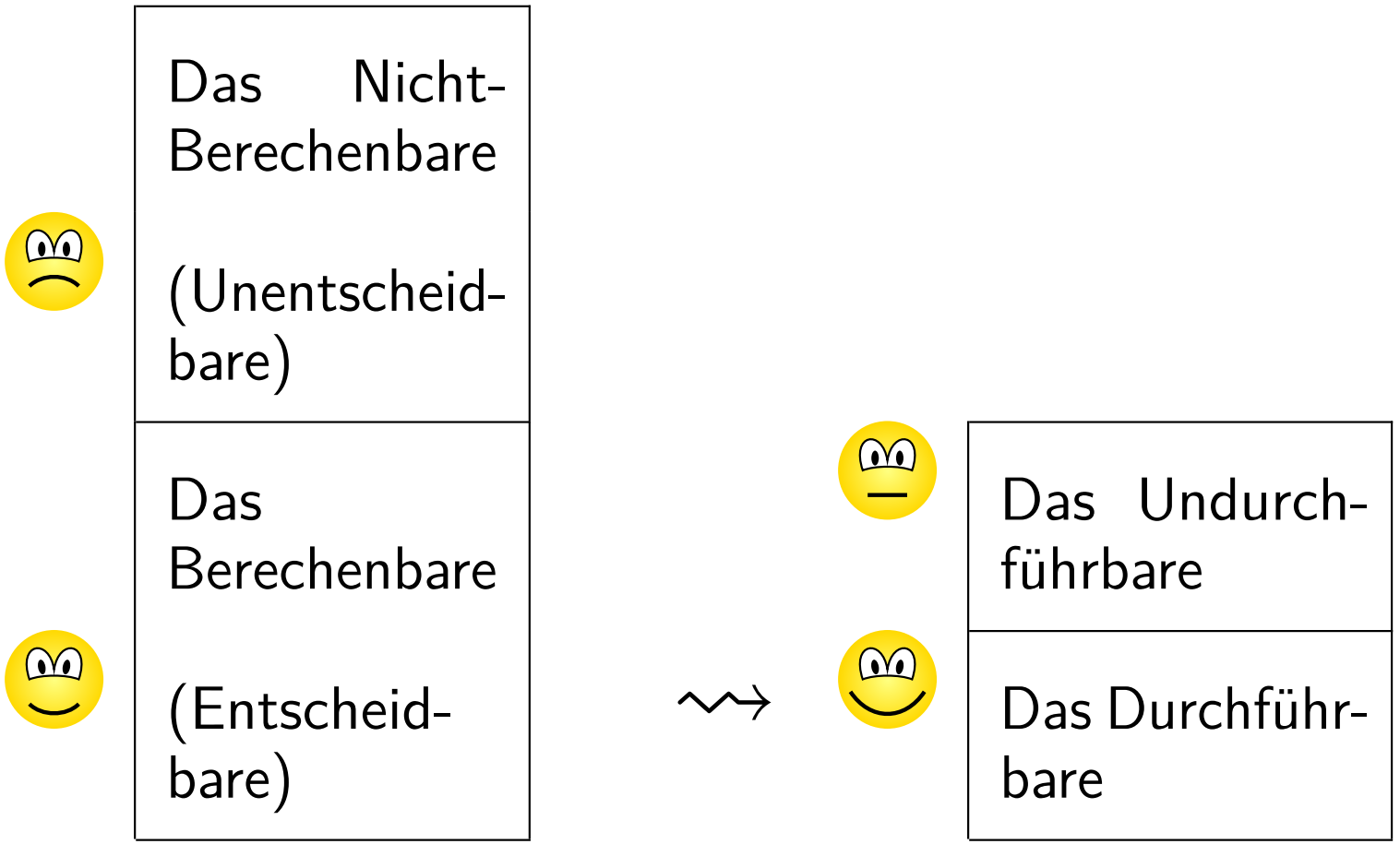
Teil 8
Komplexitätstheorie

Frühlingssemester 2019

L. Di Caro
D. Flumini
O. Stern

zh
aw

Berechenbarkeit und Komplexitätstheorie



(Nach: David Harel, *Das Affenpuzzle*)

Überblick

Komplexitätstheorie

Theorie der quantitativen Gesetze und Grenzen der algorithmischen Informationsverarbeitung.

Ziel

Klassifizierung von (entscheidbaren) Problemen nach ihrer Schwierigkeit (**Komplexität**).

Komplexitätsmasse

- **Zeitkomplexität**: Laufzeit des besten Programms, welches das Problem löst.
- **Platzkomplexität**: Speicherplatzbedarf des besten Programms.
- **Beschreibungskomplexität**: Länge des kürzesten Programms.

Hier im Folgenden: Zeitkomplexität

Zeitkomplexität im Formalismus der TM

Definition (Zeitkomplexität einer Eingabe)

Sei M eine TM mit Eingabealphabet Σ, die immer hält und sei $w \in \Sigma^*$. Der **Zeitbedarf von M auf der Eingabe w** ist

$$\mathsf{Time}_M(w) \quad = \quad \text{Anzahl von Konfigurationsübergängen in der Berechnung von } M \text{ auf } w$$

Verallgemeinerung: Abhängigkeit von der *Länge* der Eingabe

Definition

Für ein $n \in \mathbb{N}$ ist der **Zeitbedarf von M auf Eingaben der Länge n im schlechtesten Fall** definiert als

$$\mathsf{Time}_M(n) \quad = \quad \max\{\mathsf{Time}_M(w) \mid |w| = n\}\,.$$

Asymptotische Komplexitätsmessung

Idee:

- Die Zeitkomplexität darf nicht vom verwendeten System (Hardware und Software) abhängen.
- Deshalb sollen konstante Faktoren ignoriert werden.
- Vor allem die Laufzeit von grossen Eingaben ist ausschlaggebend.
- Funktionen, die sich für grosse Eingaben ähnlich verhalten, sollen zur gleichen Komplexitätsklasse gehören.

Asymptotische Komplexitätsmessung

$\mathcal{O}$-Notation (Landau Symbole)

Seien $f, g\colon \mathbb{N} \to \mathbb{N}$ zwei Funktionen. Dann gilt

$f \in \mathcal{O}(g)$: Es existieren ein $n_0 \in \mathbb{N}$ und ein $c \in \mathbb{N}$, so dass für alle $n \geq n_0$ gilt:

$$f(n) \leq c \cdot g(n)\,.$$

(f wächst asymptotisch nicht schneller als g)

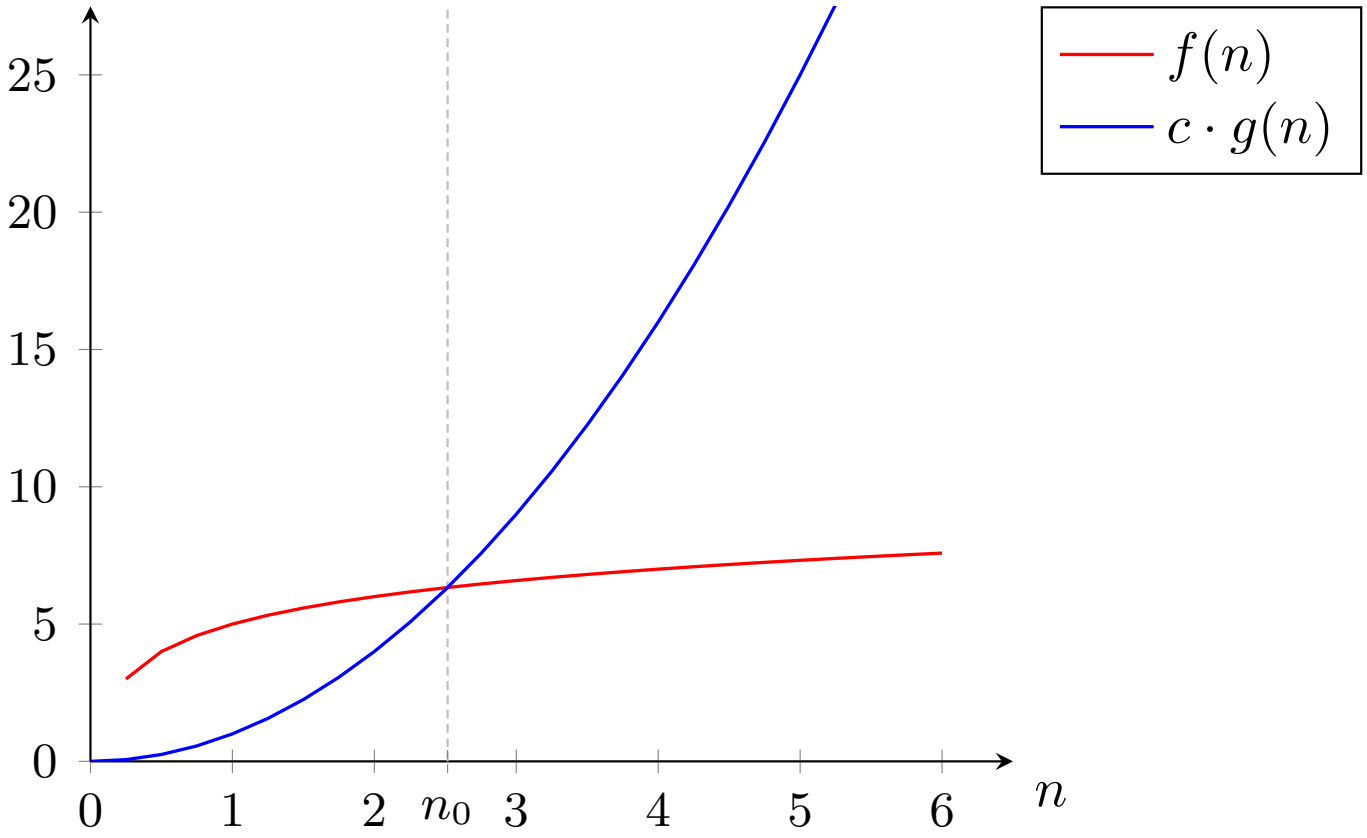
$f \in \Omega(g)$: Es existieren ein $n_0 \in \mathbb{N}$ und ein $d \in \mathbb{N}$, so dass für alle $n \geq n_0$ gilt:

$$f(n) \geq \frac{1}{d} \cdot g(n)\,.$$

(f wächst asymptotisch mindestens so schnell wie g)

$f \in \Theta(g)$: Es gilt $f(n) \in \mathcal{O}(g(n))$ und $f(n) \in \Omega(g(n))$.
(f und g sind asymptotisch gleich)

Asymptotische Komplexitätsmessung



$\mathcal{O}$-Notation

Beispiele

- $\underbrace{7n+4}_{f(n)} \in \mathcal{O}(\underbrace{n}_{g(n)})$, weil für $n \geq 6 = n_0$ gilt, dass

$$f(n) = 7n + 4 \leq 8 \cdot n = c \cdot g(n)\,.$$

- $\frac{1}{2}n^2 + 5n - 4 \in \mathcal{O}(n^2)$, weil für alle $n \geq 1$ gilt, dass

$$\frac{1}{2}n^2 + 5n - 4 \leq 6 \cdot n^2\,.$$

- $n\log(n) \in \mathcal{O}(n\log(n))$. Aber es gilt auch $n\log(n) \in \mathcal{O}(n^2)$, weil $\log(n) \leq n$ gilt für alle $n > 0$, also ist $n\log(n) \leq n^2$ für alle $n \geq 0$.

$\mathcal{O}$-Notation

Rechenregeln:

- Konstante Vorfaktoren kann man ignorieren: $c \cdot f(n) \in \mathcal{O}(f(n))$
- Für eine Konstante c gilt: $c \in \mathcal{O}(1)$
- Bei Polynomen ist nur die höchste Potenz entscheidend: $a_k n^k + a_{k-1} n^{k-1} + \ldots + a_1 n + a_0 \in \mathcal{O}(n^k)$
- Die $\mathcal{O}$-Notation ist transitiv: Aus $f(n) \in \mathcal{O}(g(n))$ und $g(n) \in \mathcal{O}(h(n))$ folgt $f(n) \in \mathcal{O}(h(n))$

Anmerkung:

$\mathcal{O}(g(n))$ ist die *Menge* aller Funktionen $f(n)$, für die es zwei Konstanten n_0 und c gibt, so dass für alle $n \geq n_0$ gilt, dass

$$f(n) \leq c \cdot g(n)\,.$$

Zeitkomplexität

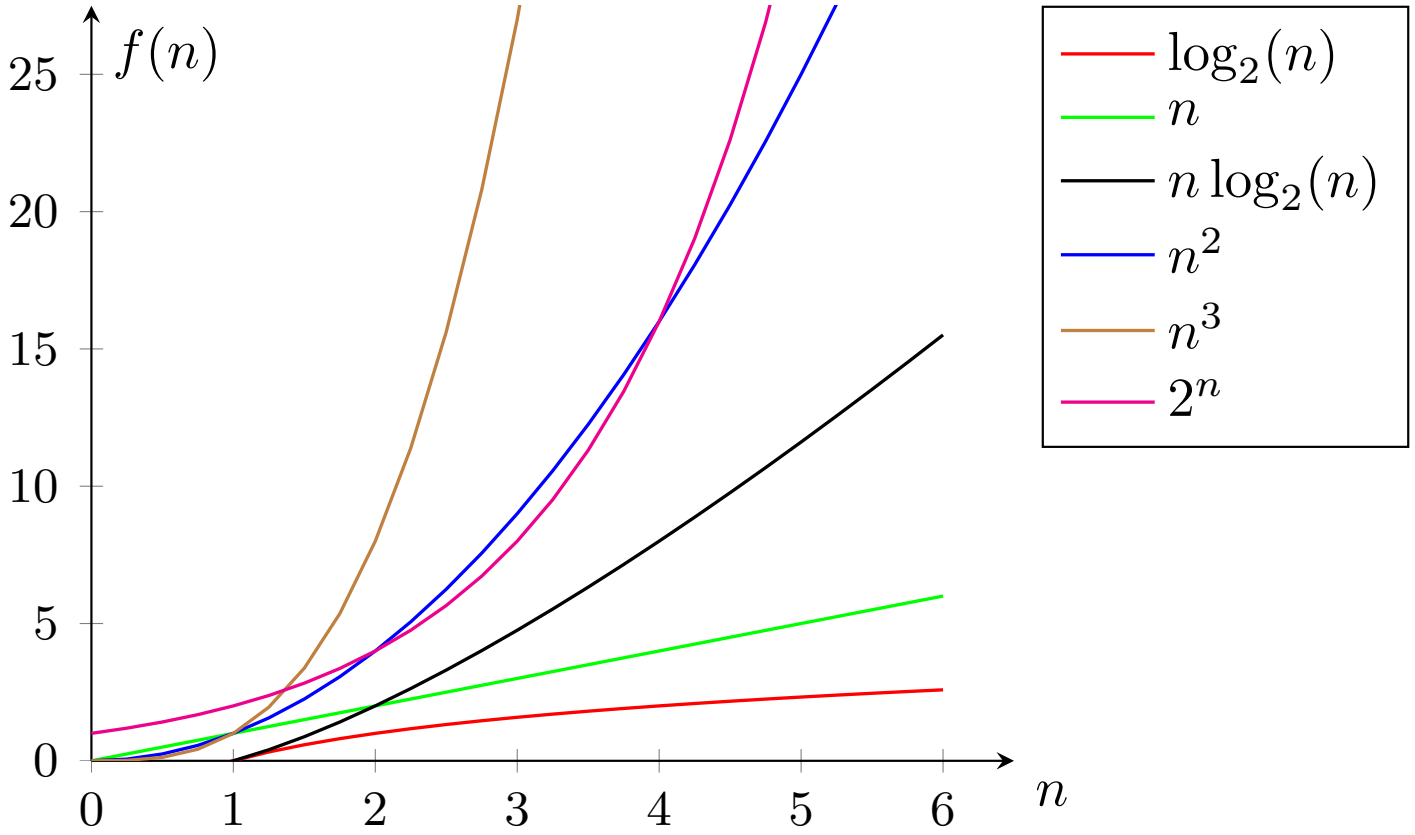
Beispiel (Zeitkomplexität zum Vergleichen von zwei Wörtern)

Zeitkomplexität einer 2-Band TM zum Vergleich von zwei Wörtern x und y der Länge jeweils n.

Arbeitsweise der Maschine	Zeit
x auf das 2. Band kopieren:	$\mathcal{O}(n)$
Lesekopf auf dem 2. Band zurück zum Anfang:	$\mathcal{O}(n)$
Vergleich Zeichen für Zeichen:	$\mathcal{O}(n)$
Total:	$\mathcal{O}(n)$

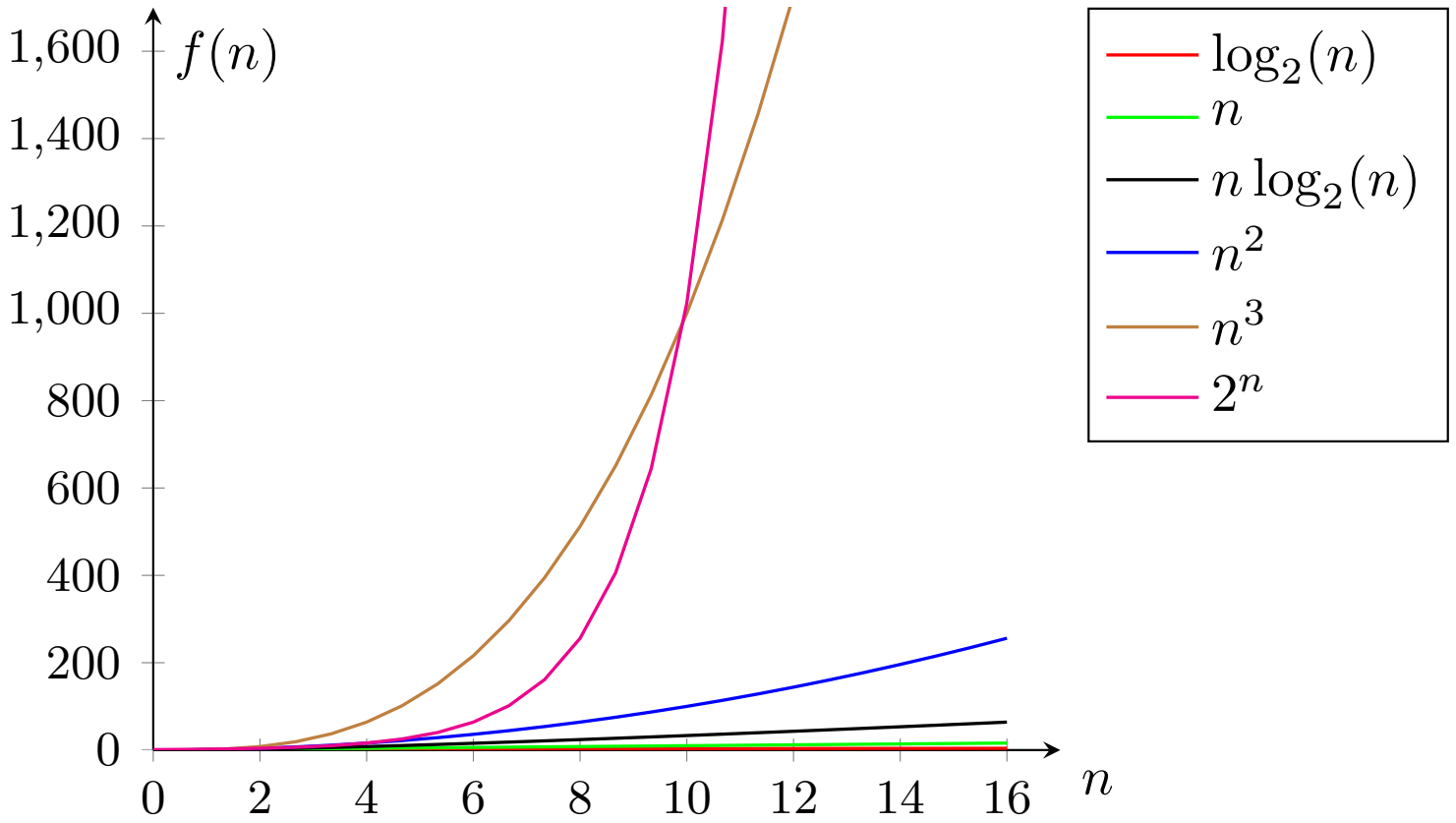
Zeitkomplexität

Verschiedene Zeitkomplexitäten im Vergleich:



Zeitkomplexität

Verschiedene Zeitkomplexitäten im Vergleich:



Zeitkomplexität von Problemen

Im Allgemeinen nicht exakt bestimmbar.

Deshalb:

Nur obere und untere Schranken an die Zeitkomplexität von Problemen.

Definition (Schranken für die Zeitkomplexität eines Problems)

Seien $f, g\colon \mathbb{N} \to \mathbb{R}^+$ zwei Funktionen und U ein (entscheidbares) Problem.

- $\mathcal{O}(f(n))$ ist eine **obere Schranke** für die Zeitkomplexität von U, falls **eine TM existiert**, die U löst und eine Zeitkomplexität in $\mathcal{O}(f(n))$ hat.
- $\Omega(g(n))$ ist **untere Schranke** für die Zeitkomplexität von U, falls **für alle TM** M, die U lösen, gilt, dass $\text{Time}_M(n) \in \Omega(g(n))$.

Zeitkomplexität von Problemen

Anmerkungen:

- **Obere Schranken**

  Meist recht einfach zu bestimmen, weil die Angabe **eines** Algorithmus und die Abschätzung seiner Zeitkomplexität ausreicht.

- **Untere Schranken**

  Sehr schwierig, weil gezeigt werden muss, dass **alle** Algorithmen keine bessere Schranke liefern.
  Deshalb sind meist nur triviale untere Schranken von $\Omega(n)$ bekannt (Eingabe muss komplett gelesen werden).

Klassifizierung von Problemen

Zur Klassifizierung von Problemen nach Schwierigkeit verwenden wir eine „relative Schwierigkeit“:

„Wenn ein Problem A schnell lösbar wäre, dann auch tausend andere Probleme.“

Was heisst schnell lösbar?

Definition (In Polynomzeit lösbar)

Ein Problem U heisst **in Polynomzeit lösbar**, wenn es eine obere Schranke $\mathcal{O}(n^c)$ gibt für eine Konstante $c \geq 1$.

Klassifizierung von Problemen

Definition (Die Klasse $\mathbf{P}$)

Die **Klasse aller in Polynomzeit entscheidbaren Sprachen** wird $\mathbf{P}$ genannt.

Beispiele

- Zuweisung eines Wertes $\in \mathcal{O}(1)$
- Sequentielle Suche $\in \mathcal{O}(n)$
- Binäre Suche $\in \mathcal{O}(\log_2(n))$
- Sortieren mit Mergesort $\in \mathcal{O}(n \log_2(n))$

Wieso bedeutet polynomiell effizient lösbar?

1. Wachstum von Polynomen im Vergleich zu anderen Funktionen:

$f(n)$	10	50	100	300
$20\log_2(n)$	≈ 83	≈ 113	≈ 133	≈ 165
$10n$	100	500	1 000	3 000
$2n^2$	200	5 000	20 000	180 000
n^3	1 000	125 000	1 000 000	27 000 000
2^n	1 024	16 Ziffern	31 Ziffern	91 Ziffern
$n!$	$\approx 3.6 \cdot 10^6$	65 Ziffern	158 Ziffern	615 Ziffern

2. Für die meisten praktisch auftretenden Probleme, die in Polynomzeit lösbar sind, kann auch ein Programm mit Laufzeit $\mathcal{O}(n^c)$ gefunden werden für ein kleines c.
3. Die Klasse der Polynome ist sehr robust, unabhängig vom verwendeten Rechnermodell.
4. Polynome sind unter Addition, Multiplikation und Komposition abgeschlossen.

Nichtdeterministisch in Polynomzeit lösbar

Definition (Die Klasse $\mathbf{NP}$)

Die **Klasse aller von einer NTM in Polynomzeit entscheidbaren Sprachen** nennen wir $\mathbf{NP}$.

Achtung

$\mathbf{NP}$ heisst NICHT „nicht polynomiell“ sondern „nichtdeterministisch polynomiell“.

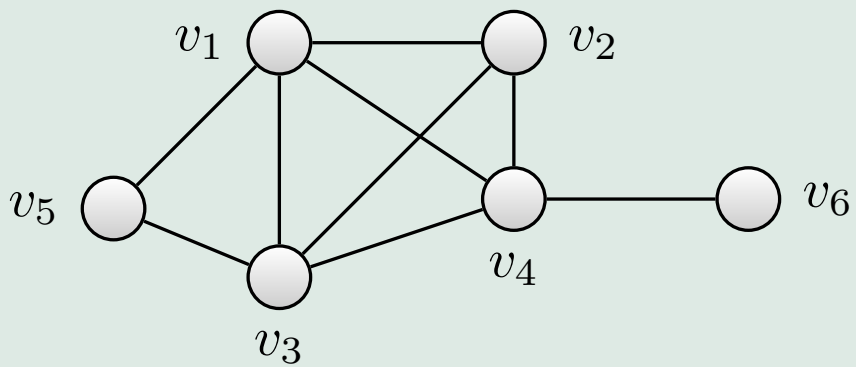
Anschauliche Definition von NP: Polynomzeit-Verifizierer

Definition (Graph)

Ein **Graph** $G = (V, E)$ besteht aus einer endlichen Menge von **Knoten** $V = \{v_1, v_2, \ldots, v_n\}$ und einer Menge von **Kanten**

$$E \subseteq \{\{v_i, v_j\} \mid v_i, v_j \in V \text{ für } i \neq j\} \ .$$

Beispiel

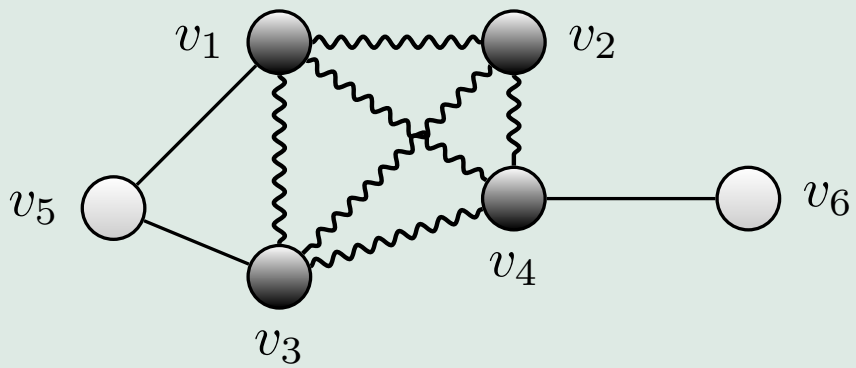


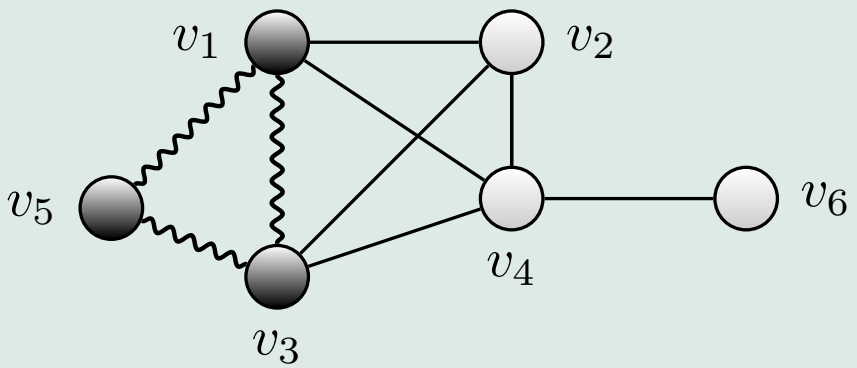
$G = (V, E)$ mit
$V = \{v_1, v_2, v_3, v_4, v_5, v_6\}$ und
$E = \{\{v_1, v_2\}, \{v_1, v_3\}, \ldots\}$

Anschauliche Definition von NP: Polynomzeit-Verifizierer

Beispiel

Die folgenden Teilmengen von Knoten erfüllt die Eigenschaft, dass alle Knoten aus der Teilmenge paarweise verbunden sind:





Definition

Eine Teilmenge $S \subseteq V$ von Knoten, so dass alle Knoten aus S paarweise durch Kanten verbunden sind, wird eine **Clique** genannt.

Polynomzeit-Verifizierer

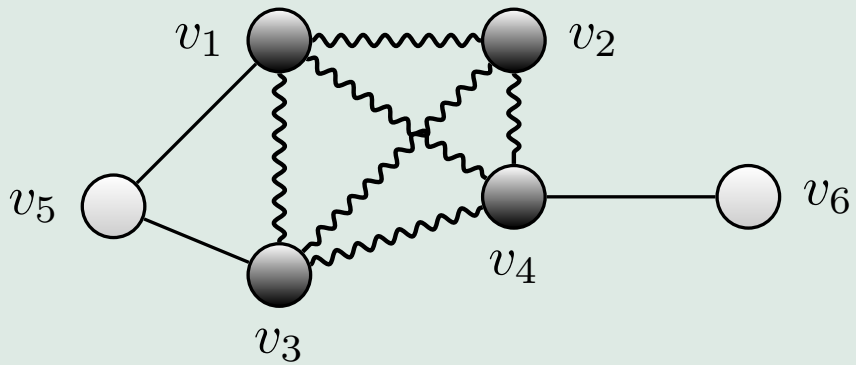
Definition (CLIQUE-Problem)

Gegeben: Ungerichteter Graph $G = (V, E)$, Zahl $k \leq |V|$

Frage: Gibt es in G eine Clique S der Grösse k?

Beispiel (CLIQUE-Problem)

$k = 4$:



$S = \{v_1, v_2, v_3, v_4\}$ ist eine Clique

Zur Bestimmung einer Clique in einem Graphen sind nur exponentielle Verfahren bekannt.

Wenn aber ein *Lösungskandidat* bekannt ist, kann in polynomieller Zeit überprüft werden, ob dies tatsächlich eine Lösung ist.

Polynomzeit-Verifizierer

Definition (p-Verifizierer)

Sei $L \subseteq \Sigma^*$ eine Sprache und $p\colon \mathbb{N} \to \mathbb{N}$ eine Funktion. Eine TM M ist ein **p-Verifizierer** für L, falls M wie folgt auf allen Eingaben $w\#x$ für $w \in \Sigma^*$ und $x \in \{0,1\}^*$ arbeitet:

- $\text{Time}_M(w\#x) \leq p(|w|)$ für alle Eingaben $w\#x$.
- Für jedes $w \in L$ existiert ein $x \in \{0,1\}^*$ mit $|x| \leq p(|w|)$, so dass M die Eingabe $w\#x$ akzeptiert.
  x heisst **Zeuge** für $w \in L$.
- Für alle $w \notin L$ existiert kein Zeuge.

Definition (Polynomzeit-Verifizierer)

Falls $p \in \mathcal{O}(n^k)$ für ein $k \in \mathbb{N}$, dann ist M ein **Polynomzeit-Verifizierer**.

Polynomzeit-Verifizierer

Informell:

Beispiel (Polynomzeit-Verifizierer für das CLIQUE-Problem)

Eingabe:

- Ein ungerichteter Graph mit n Knoten und eine Zahl k.
- Zeuge: Menge von Knoten, die in der Clique sind.

Der Verifizierer überprüft, ob es sich tatsächlich um eine Clique handelt, d. h. ob in dieser Menge zwischen je zwei Knoten eine Kante vorhanden ist.

$\Longrightarrow$ Rechenzeit in $\mathcal{O}(n^2)$.

Polynomzeit-Verifizierer (Backup)

Formal:

Beispiel (Polynomzeit-Verifizierer für das CLIQUE-Problem)

Eingabe:

- Kodierung eines ungerichteten Graphen mit n Knoten, Zahl k
- Zeuge: Ein Bit x_i für jeden Knoten v_i des Graphen:

$$x_i = \begin{cases} 1, & \text{falls } v_i \in S \\ 0, & \text{falls } v_i \notin S \end{cases}$$

für alle $1 \leq i \leq n$ und eine Clique S.

Der Verifizierer überprüft für alle Knotenpaare (v_i, v_j) mit $x_i = 1$ und $x_j = 1$, ob die Kante in G vorhanden ist.

$\Longrightarrow$ Rechenzeit in $\mathcal{O}(n^2)$.

Polynomzeit-Verifizierer

Definition (Koffer-Pack-Problem)

Gegeben: Ein Koffer und n verschiedene Gegenstände mit unterschiedlichen Formen und Grössen.

Frage: Passen diese n Gegenstände in den Koffer?

Beispiel (Polynomzeit-Verifizierer für das Rucksack-Problem)

Eingabe

- Ein Koffer und n verschiedene, unterschiedlich geformte Gegenstände.
- Zeuge: Plan, wie diese Gegenstände in den Koffer gepackt werden.

Der Verifizierer legt die Gegenstände so in den Koffer, wie der Plan vorgibt, und überprüft damit, ob sie alle Platz haben.

$\Longrightarrow$ Rechenzeit in $\mathcal{O}(n)$.

Polynomzeit-Verifizierer

Theorem

$\mathbf{NP}$ *ist die Menge aller Sprachen, für die ein Polynomzeit-Verifizierer existiert.*

Zusammenfassung:

$\mathbf{P} \triangleq$ Lösung *finden* in Polynomzeit

$\mathbf{NP} \triangleq$ Lösung *verifizieren* in Polynomzeit

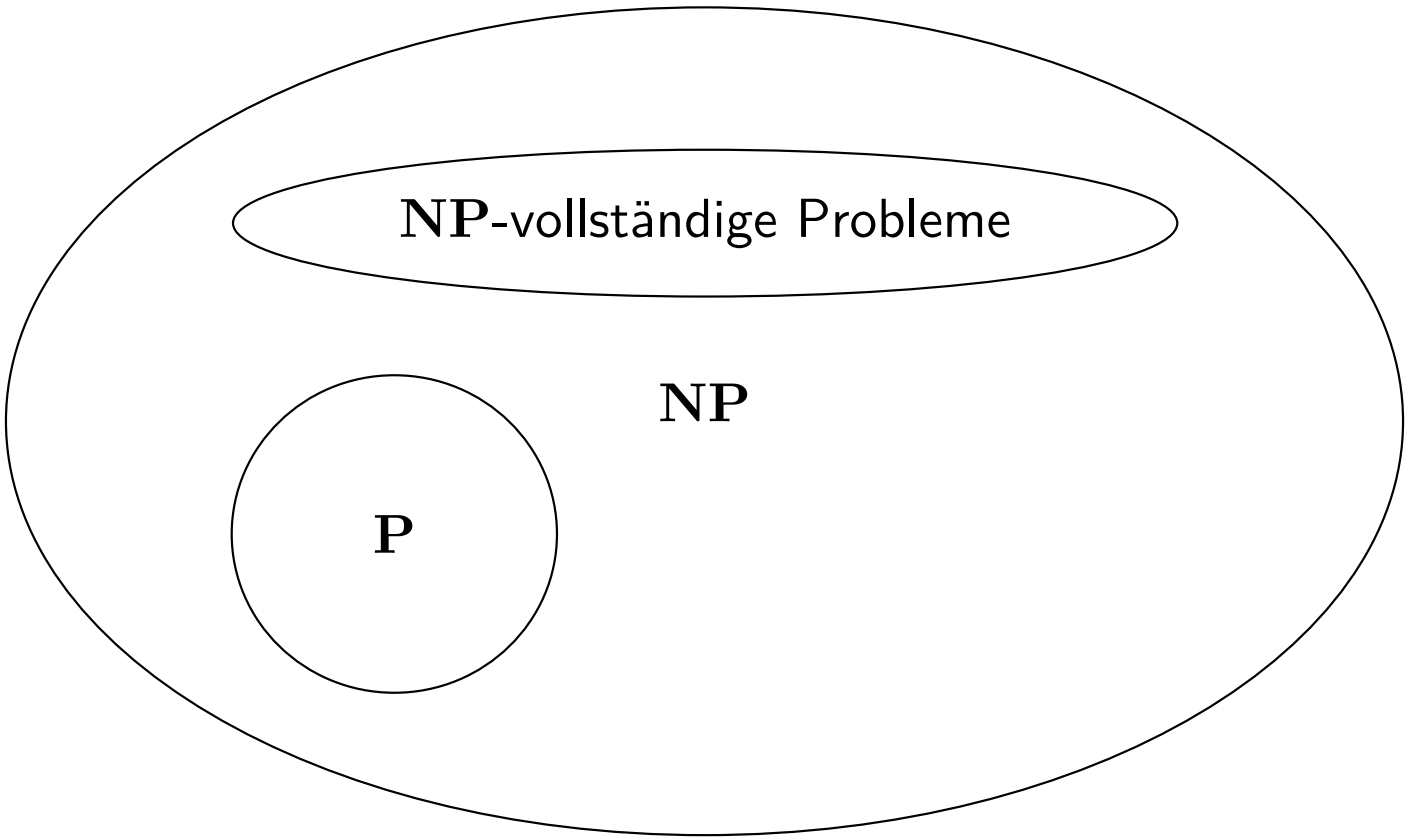
Offene Frage: Gilt $\mathbf{P} = \mathbf{NP}$?

Vermutung: $\mathbf{P} \neq \mathbf{NP}$

NP-Vollständigkeit

Konzept zum Beweis relativer Schwierigkeit von Problemen.

Vermutung:



Polynomielle Reduktion

Definition (Polynomielle Reduktion)

Seien $L_1 \subseteq \Sigma_1^*$ und $L_2 \subseteq \Sigma_2^*$ zwei Sprachen.
L_1 ist **polynomiell** auf L_2 **reduzierbar**, $L_1 \preceq_p L_2$, falls eine TM M existiert mit $\text{Time}_M(n) \in \mathcal{O}(n^k)$ für ein $k \in \mathbb{N}$, die für eine Eingabe $x \in \Sigma_1^*$ ein Wort $M(x) \in \Sigma_2^*$ berechnet mit

$$x \in L_1 \iff M(x) \in L_2 \,.$$

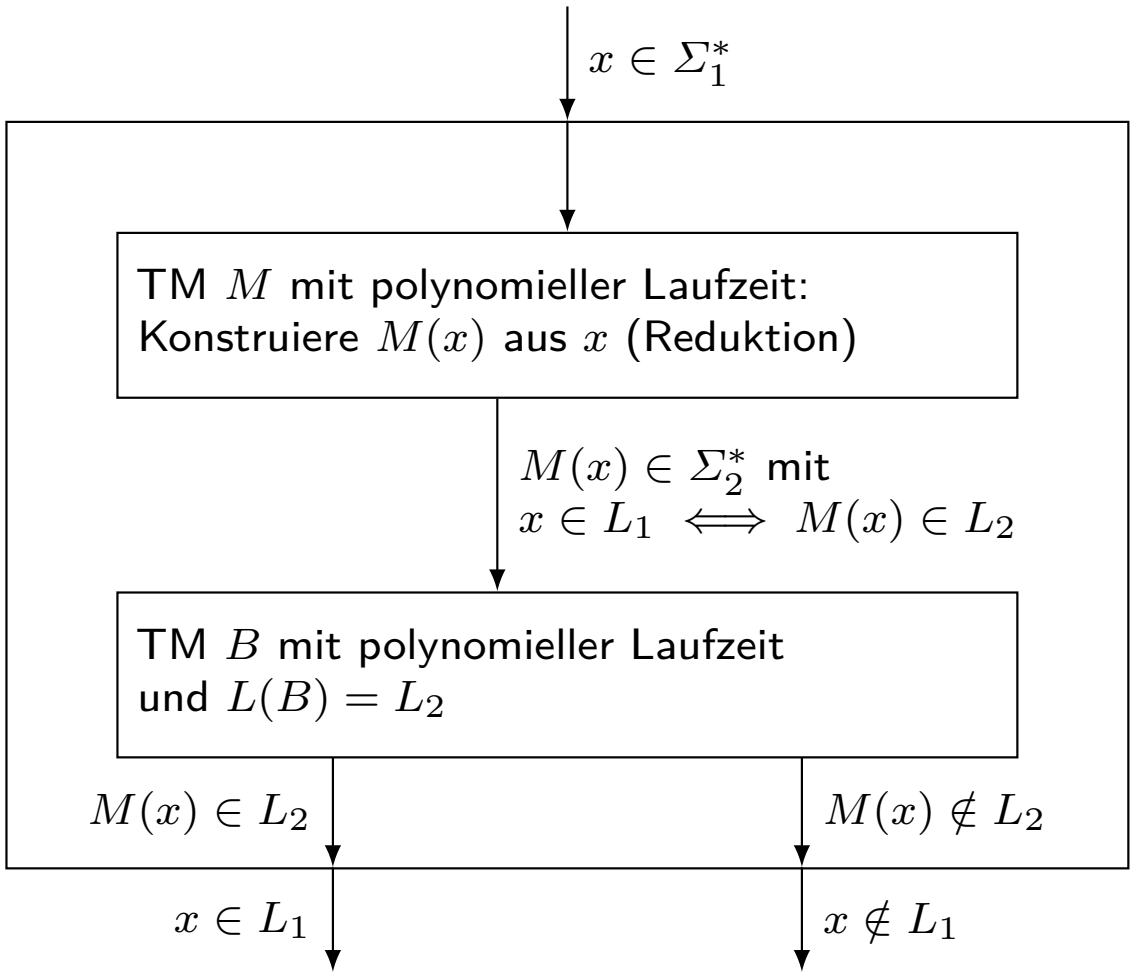
Anmerkung:

$L_1 \preceq_p L_2$ bedeutet, dass L_2 mindestens so schwer ist in Bezug auf die Lösbarkeit in polynomieller Zeit wie L_1.

(vgl. Definition der Reduzierbarkeit im Teil 7 zur Berechenbarkeit)

Polynomielle Reduktion

Wenn $L_1 \preceq_p L_2$ gilt, dann kann man eine polynomielle TM B für L_2 dafür verwenden, eine polynomielle TM A für L_1 zu bauen:



TM A mit polynomieller Laufzeit und $L(A) = L_1$

$\mathbf{NP}$-schwer und $\mathbf{NP}$-vollständig

Definition ($\mathbf{NP}$-schwer)

Eine Sprache L heisst **$\mathbf{NP}$-schwer**, falls für alle Sprachen $L' \in \mathbf{NP}$ gilt, dass $L' \preceq_p L$.

D.h. L ist bezüglich der Lösbarkeit in polynomieller Zeit mindestens so schwer wie jedes einzelne Problem in $\mathbf{NP}$.

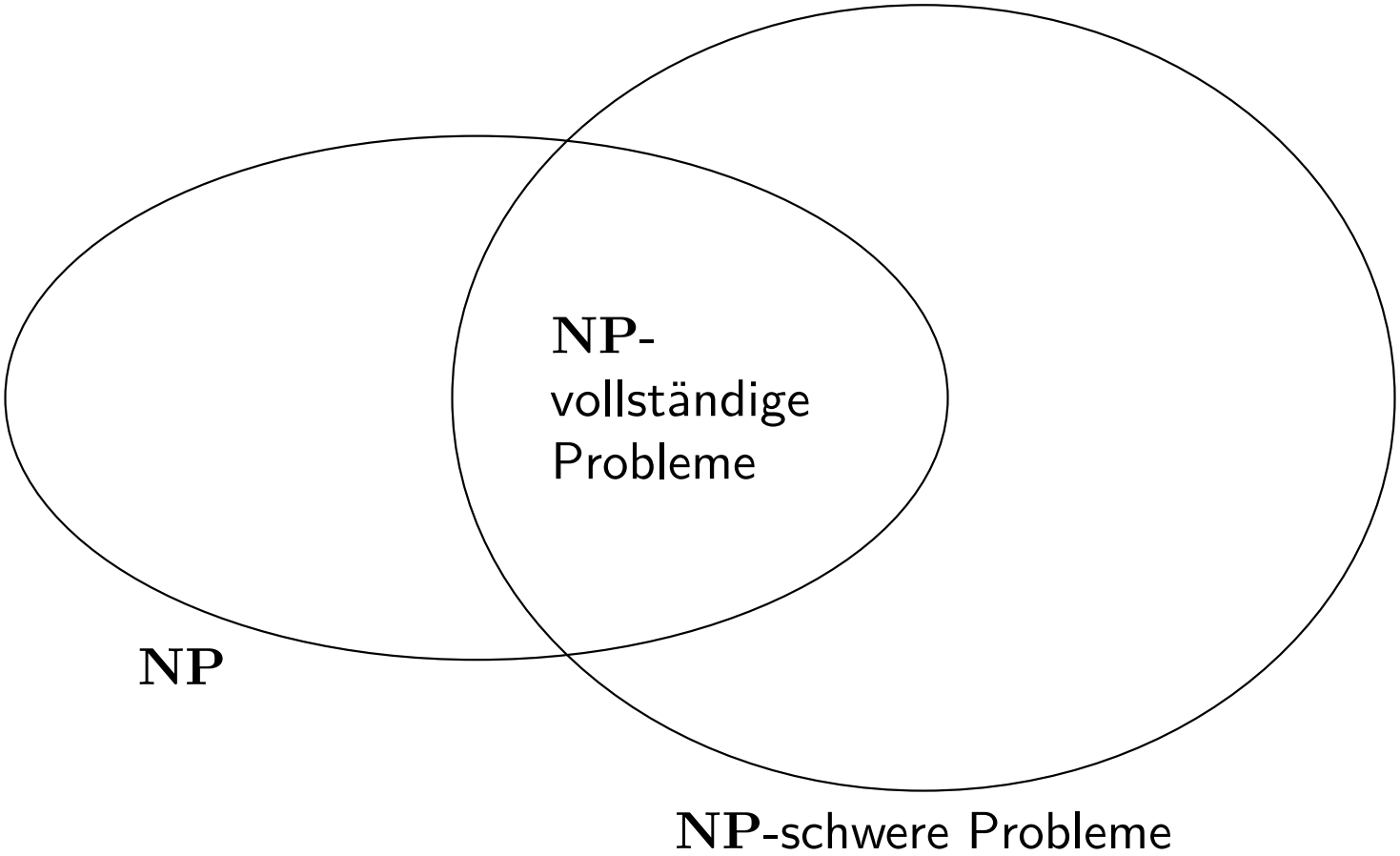
Definition ($\mathbf{NP}$-vollständig)

Eine Sprache L heisst **$\mathbf{NP}$-vollständig**, falls

1. $L \in \mathbf{NP}$, und
2. L ist $\mathbf{NP}$-schwer.

Die Menge der $\mathbf{NP}$-vollständigen Sprachen betrachten wir als die gesuchte Teilklasse von schwersten Entscheidungsproblemen in $\mathbf{NP}$.

NP-schwer und NP-vollständig



$\mathbf{NP}$-schwer und $\mathbf{NP}$-vollständig

Beobachtung:

Wenn es gelingt für auch nur ein $\mathbf{NP}$-vollständiges L nachzuweisen, dass $L \in \mathbf{P}$ ist, dann gilt

$$\mathbf{P} = \mathbf{NP}\,.$$

Nachweis der $\mathbf{NP}$-Vollständigkeit

Vorgehen zum Nachweis der $\mathbf{NP}$-Schwere:

1. Finde ein erstes $\mathbf{NP}$-schweres Problem.
2. Finde eine Polynomzeitreduktion von einem bekannten $\mathbf{NP}$-schweren Problem auf das neue Problem, dessen $\mathbf{NP}$-Schwere gezeigt werden soll.

$\mathbf{NP}$-Vollständigkeit:

1. Zeige, dass das Problem $\mathbf{NP}$-schwer ist.
2. Zeige, dass das Problem in $\mathbf{NP}$ liegt.

Ein erstes $\mathbf{NP}$-vollständiges Problem

Definition (Aussagenlogische Formeln)

Aussagenlogische Formeln sind induktiv definiert durch:

1. Die Konstanten 0 und 1 und die Variablen $x_1, x_2, \ldots, x_n$ sind Aussagenlogische Formeln.
2. **Negation, NICHT**: Wenn φ eine Aussagenlogische Formel ist, dann auch $\neg\varphi$.
3. **Konjunktion, UND**: Wenn φ und ψ Aussagenlogische Formeln sind, dann auch $(\varphi \wedge \psi)$.
4. **Disjunktion, ODER**: Wenn φ und ψ Aussagenlogische Formeln sind, dann auch $(\varphi \vee \psi)$

Ein erstes $\mathbf{NP}$-vollständiges Problem

Konvention zur Klammerersparnis:

- $\neg$ bindet stärker als $\wedge$ oder $\vee$
- $\wedge$ bindet stärker als $\vee$
- statt $\neg x_i$ schreiben wir $\overline{x_i}$

Ein erstes $\mathbf{NP}$-vollständiges Problem

Definition (Konjunktive Normalform)

Eine Aussagenlogische Formel ist in **konjunktiver Normalform** (KNF), wenn sie eine Konjunktion von Disjunktionen von einfachen oder negierten Variablen ist.

Beispiel (Formel in KNF)

$$(x_1 \vee \underbrace{x_2}_{\text{Literal}} \vee \underbrace{\overline{x_4}}_{\text{Literal}}) \wedge \underbrace{(x_5 \vee x_2)}_{\text{Klausel}} \wedge (\overline{x_1} \vee \overline{x_3})$$

Ein erstes $\mathbf{NP}$-vollständiges Problem

Definition ((Erfüllende) Belegung)

Eine **Belegung** ist eine Funktion, die jeder Variablen einen Wert aus $\{0, 1\}$ zuordnet.
Eine Belegung **erfüllt** eine Formel φ, wenn die Auswertung der Formel nach den üblichen Regeln der Aussagenlogik den Wert 1 ergibt.

Definition (Erfüllbare Formel)

Eine Formel ist **erfüllbar**, wenn sie eine erfüllende Belegung besitzt.

Definition ($\mathbf{SAT}$)

$\mathbf{SAT}$ (von engl. satisfiability) ist das Problem zu entscheiden, ob eine gegebene Formel in KNF erfüllbar ist.

Satz von Cook

Satz (Satz von Cook)

SAT *ist* **NP**-*vollständig.*

Beweisidee[1]: Das Folgende muss gezeigt werden:

SAT $\in$ NP: Polynomzeit-Verifizierer für **SAT**: Zeuge $l_1, l_2, \ldots, l_n$ mit $l_i = 1$, falls $x_i = 1$ für eine erfüllende Belegung, und $l_i = 0$, falls $x_i = 0$ für eine erfüllende Belegung.
Der Verifizierer überprüft in polynomieller Zeit, ob der Zeuge eine erfüllende Belegung ist.

SAT ist NP-schwer: D.h. es gilt $L \preceq_p \mathbf{SAT}$ für alle $L \in \mathbf{NP}$: Kodiere die Berechnung einer polynomialzeitbeschränkten NTM durch eine KNF-Formel.

□

[1] Siehe Hopcroft et. al S. 486ff.

Nachweis der NP-Vollständigkeit weiterer Probleme

Satz

*Wenn P_1 **NP**-schwer und P_2 in **NP** enthalten ist und eine polynomielle Reduktion $P_1 \preceq_p P_2$ existiert, dann ist P_2 **NP**-vollständig.*

Beweis: Zu zeigen ist: $L \preceq_p P_2$ für alle $L \in \mathbf{NP}$
Es gilt $L \preceq_p P_1$ für alle $L \in \mathbf{NP}$, also existiert eine TM A mit polynomieller Laufzeit $p_A(n)$, so dass

$$x \in L \iff A(x) \in P_1 \,.$$

Es gilt auch $P_1 \preceq_p P_2$ für alle $L \in \mathbf{NP}$, also existiert eine TM B mit polynomieller Laufzeit $p_B(n)$, so dass

$$A(x) \in P_1 \iff B(A(x)) \in P_2 \,.$$

Die Simulation dieser beiden TM durch eine TM C ergibt eine polynomielle Laufzeit von $p_A(n) + p_B(p_A(n))$. □

Nachweis der $\mathbf{NP}$-Vollständigkeit weiterer Probleme

Beispiel (CLIQUE ist $\mathbf{NP}$-vollständig)

CLIQUE $\in \mathbf{NP}$: schon gezeigt

$\mathbf{SAT} \preceq_p$ CLIQUE: Sei φ eine Formel in KNF.

Konstruiere eine Eingabe (G, k) für CLIQUE, so dass gilt:

$$\varphi \in \mathbf{SAT} \iff (G, k) \in \text{CLIQUE}\,.$$

Setze $k = m$ (Anzahl der Klauseln).

Für jedes Auftreten eines Literals in einer Klausel i an Stelle j in φ wird ein Knoten $[i, j]$ gesetzt. Kanten werden zwischen Knoten aus unterschiedlichen Klauseln gesetzt, falls die Literale nicht Negationen voneinander sind.

Nachweis der $\mathbf{NP}$-Vollständigkeit weiterer Probleme

Beispiel (CLIQUE ist $\mathbf{NP}$-vollständig)

Zu zeigen:

a) (G, k) lässt sich aus φ in polynomieller Zeit konstruieren und

b) φ erfüllbar $\iff$ G enthält eine Clique der Grösse k

Idee zu b): Keine Kante zwischen x und $\overline{x}$

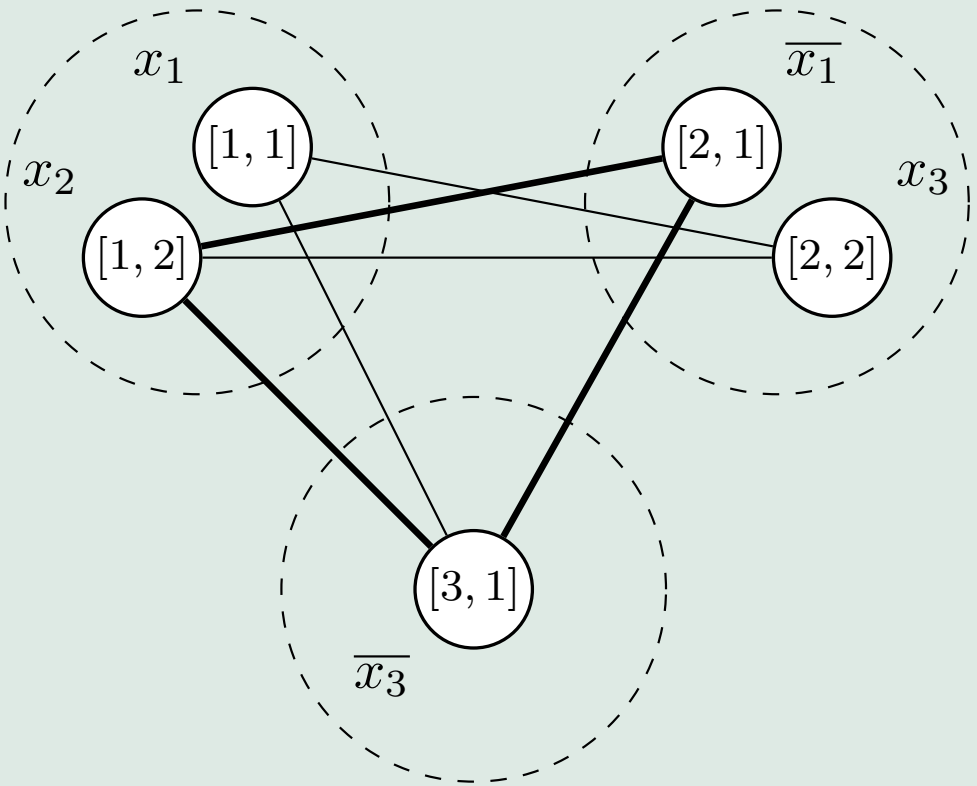
$\Longrightarrow$: Jede Clique in G entspricht einer Menge von gleichzeitig auf 1 setzbaren Literalen

$\Longrightarrow$: Clique der Grösse k enthält einen Knoten pro Klausel, also ist jede Klausel erfüllt.

$\Longleftarrow$: erfüllende Belegung definiert Clique in G der Grösse $k = m$

Beispiel (CLIQUE ist NP-vollständig)

Beispiel für die Konstruktion: $\varphi = (x_1 \vee x_2) \wedge (\overline{x_1} \vee x_3) \wedge (\overline{x_3})$



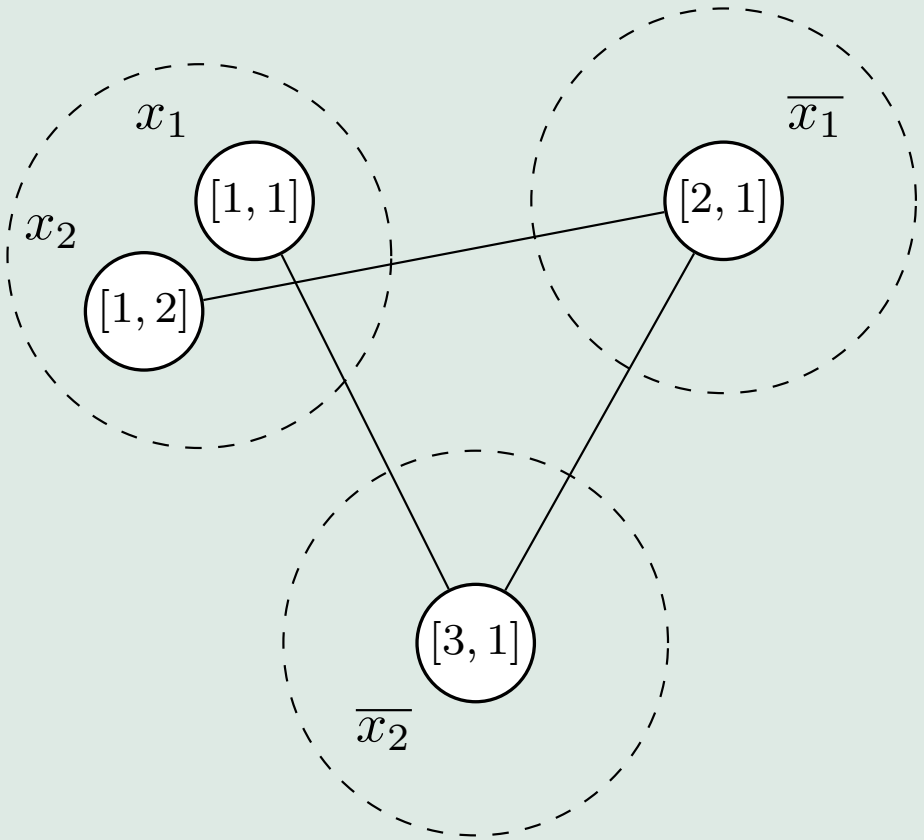
Die Formel ist erfüllbar.

$\Leftrightarrow$

Der Graph enthält eine Clique der Grösse 3.

Beispiel (CLIQUE ist NP-vollständig)

Beispiel für die Konstruktion: $\varphi' = (x_1 \vee x_2) \wedge (\overline{x_1}) \wedge (\overline{x_2})$



Die Formel ist nicht erfüllbar.

$\Leftrightarrow$

Der Graph enthält keine Clique der Grösse 3.

Beispiel (CLIQUE ist NP-vollständig)

$\varphi'' = (x_1 \vee x_2) \wedge (x_1 \vee \overline{x_2} \vee \overline{x_3}) \wedge (\overline{x_1} \vee x_3) \wedge (\overline{x_2})$