P-NP-Problem

Stefanie Feuerriegel, Damaris Soldan

Hochschule Zittau/ Görlitz

08.01.2015

Inhaltsverzeichnis

Bereits bekannt:

- *Optimierungsprobleme*, wie TSP oder Rucksackproblem
- **Berechenbarkeitstheorie**: Betrachtung von Entscheidungsproblemen

→ Untersuchung, welche Funktionen prinzipiell berechenbar sind, unabhängig vom Aufwand

- **Komplexitätstheorie**: Untersuchung, welche Funktionen *effizient* berechenbar sind

→ Aufwandsbestimmung

- zeitlicher Aufwand eines Algorithmus hängt von dessen Input ab

→ Berechnungsaufwand steigt exponentiell mit Größe der Argumente

→ Algorithmus wird als nicht effizient betrachtet

- z.B. für TSP gibt es keine effiziente Lösung
- weitere Probleme dieser Art: SAT-Problem, Cliquenproblem

Vorgehensweise des **„systematischen Probieren“** der Algorithmen:

- Rate eine Lösung
- Prüfe, ob Lösung korrekt ist

SAT-Problem

- Erfüllbarkeit eines booleschen Termes F(x_1, x_2,..., x_m)
- eng.: *Satisfiability*
- Aufbau des Termes:
  - Variablen x_1, x_2,...
  - Operationszeichen $\wedge$ und $\vee$
  - Negation $\neg$
- Gesucht: Belegung der Variablen mit Wahrheitswerten 0 und 1, sodass aussagenlogische Formel wahr wird
- bei m Variablen gibt es 2^m mögliche Kandidaten für Belegung
- z.B.: $(\neg x \vee \neg y \vee \neg z) \wedge (x \vee \neg z) \wedge (y \vee z) \wedge (x \vee \neg y)$

SAT-Problem

Anwendung eines naiven Algorithmus, der systematisches Probieren enthält:

- generiere systematisch alle Belegungen
- teste jeweils, ob die Belegung die Formel erfüllt

SAT-Problem

Praktische Anwendungen:

- in praktischer Informatik
- in Industrie: Verifikation endlicher Systeme, Algorithmen und Protokolle

Cliquenproblem

- in ungerichteten Graph $G = (V, E)$ bezeichnet Teilmenge V' von Knoten eine Clique, falls je zwei verschiedene Elemente von V' durch eine Kante verbunden sind
- Gesucht: Größe einer maximalen Clique der Ordnung k, also einen vollständigen Teilgraphen $G' = (V', E')$ mit $V' \subseteq V$ und $|V'| = k$ zu G
- z.B.: alle Studenten einer Uni als Knotenmenge V'
- verbinde zwei Knoten, falls die entsprechenden Studenten sich kennen

$\rightarrow$ Clique aus einer Gruppe V' von Studenten, von denen jeder den anderen kennt

Cliquenproblem

Anwendung eines naiven Algorithmus, der systematisches Probieren enthält:

- generiere systematisch alle Teilmengen V' von Knoten von G
- prüfe, ob V' eine Clique ist

Ramsey- oder Party-Problem

- „Wie viele Gäste muss man einladen, damit sich (mindestens) m Personen gegenseitig kennen, oder (mindestens) n Personen, die sich gegenseitig nicht kennen?“
- Partygeber weiß selbst nicht, ob Personen sich gegenseitig kennen oder nicht
- z.B.: für $m = n = 3$ muss man mindestens 6 Gäste einladen

$\rightarrow$ Ramsey-Zahl $R(3, 3) = 6$

- es wird stets ein vollständiger Graph angegeben
- Eintragen aller Kanten in den Graphen mit den Farben *blau* für „sich kennen“ und *rot* für „sich nicht kennen“

$\rightarrow$ blaues und rotes Cliquenproblem

Zurückführung auf Entscheidungsprobleme

- Komplexität eines Optimierungsproblems (OP) ist zurückführbar auf Komplexität eines zugehörigen Entscheidungsproblems (EP)
- Existiert eine Lösung für ein Problem? $\rightarrow$ Ja/Nein
- Reduzierung von OP auf EP mit Hilfe einer Schranke T

$\rightarrow$ Gibt es eine Rundreise der Länge $\leq T$?

- Aber: Auch für reduziertes Problem ist kein prinzipiell effizienterer Algorithmus bekannt!
- Wenn es einen Polynomzeitalgorithmus zur Lösung des zugehörigen EP gibt, dann gibt es auch einen PZA zur Lösung des OP.

Probleme als Sprachen

- für jede Klasse von Problemen legt man eine Codierung durch ein Alphabet Σ fest
- jede Instanz eines Problems ist dann ein Wort $w \in \Sigma^*$
- Menge aller *lösbaren* Probleme ist dann eine Teilmenge, also eine Sprache $L \subseteq \Sigma^*$
- Frage, ob bestimmtes Problem lösbar ist $\rightarrow$ Liegt Wort w in der Sprache L aller lösbaren Probleme?

$\Rightarrow$ Lösbarkeit eines Problems wird auf *Entscheidbarkeit einer Sprache* zurückgeführt

- Turingmaschine definiert Komplexität eines Problems zur Aufwandsbestimmung mittels *Bit-Komplexität*

$\rightarrow$ Takte der Turingmaschine bei Anwendung auf Probleminstanz

Probleme als Sprachen

- bisher sind keine effizienten Lösungen für SAT-Problem, Cliquenproblem oder TSP-Problem bekannt
- eine Sprache L wird als **polynomiell** klassifiziert, falls es ein Polynom $p(n)$ gibt, so dass L Komplexität $\mathcal{O}(p(n))$ hat

$\Rightarrow$ P = **Klasse aller Sprachen** $L \in \Sigma^*$ **mit polynomialer Komplexität**

SAT-Probleme als Sprache

- Menge aller SAT-Probleme stellt eine kontextfreie Sprache über Σ dar
- Menge aller lösbaren SAT-Probleme ist Teilsprache *SAT*
- Entscheidungsalgorithmus für $SAT \subseteq \Sigma^*$

$\rightarrow$ Algorithmus zur Lösung des SAT-Problems

Die Klasse P

Definition

P ist die Klasse aller Probleme, die mit deterministischen Algorithmen in polynomialer Zeit gelöst werden können.

Die Klasse P

- Klasse der leichten oder effizient lösbaren Probleme
- untere und obere Schranke: polynomial
- es gibt (berechnbare) Probleme, die nicht in **P** liegen

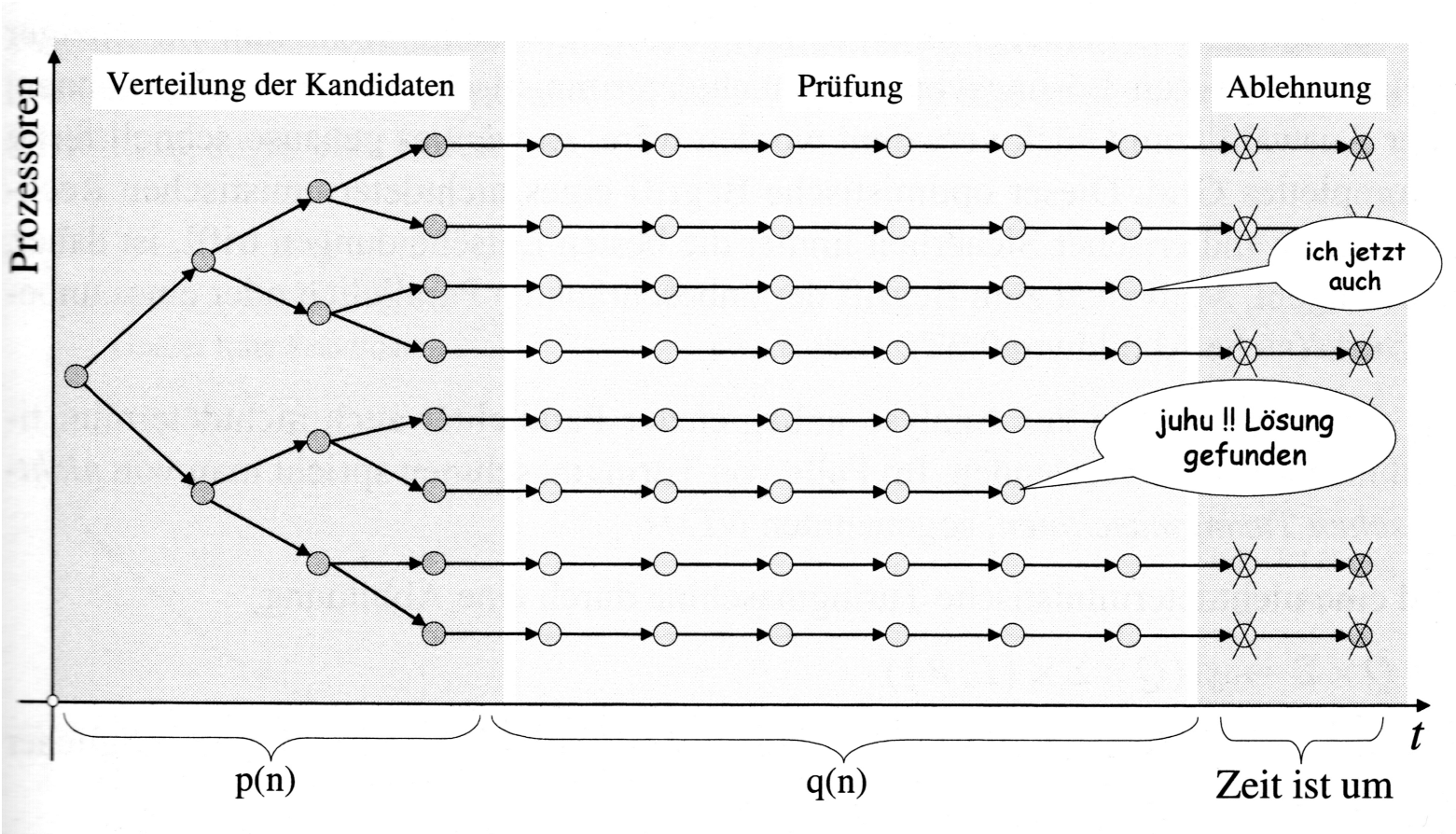
$\rightarrow$ werden mit (mehrfach) exponentiellem Mindestaufwand gelöst
$\Rightarrow$ Problemklasse **EXPTIME**

$\rightarrow$ $\mathbf{P} \subset \mathbf{EXPTIME}$

Effiziente parallele Lösungen

- beliebig große Parallelrechner oder Cluster von unbegrenzt vielen Rechnern, von denen jeder einen kleinen Teil des Problems bearbeitet $\rightarrow$ *Grid-Computing*
- Lösung in zwei Phasen: Aufgabenverteilung und Prüfung
- Lösungskandidaten in $\mathcal{O}(p(n))$ Schritten erzeugt
- jeder Rechner überprüft seinen Kandidat, ob es wirklich eine Lösung ist
- Überprüfung erfolgt in **polynomialer** Zeit $q(n)$
- Zeit abgelaufen $\rightarrow$ Berechnung wird eingestellt $\rightarrow$ Problem unerfüllbar

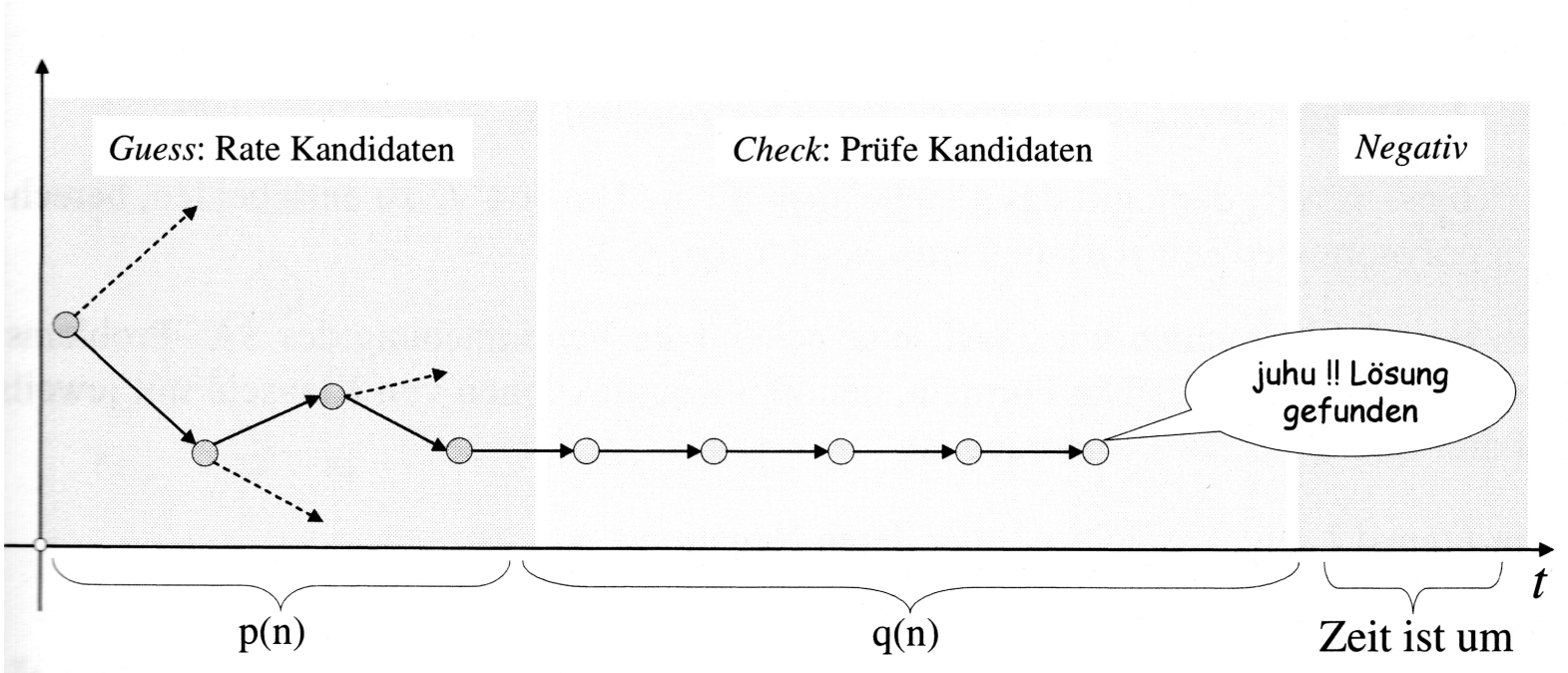
Lösung in zwei Phasen



Unter Betrachtung des Nichtdeterminismus

- zu jedem Zeitpunkt stehen ein oder mehrere mögliche nächste Schritte zur Verfügung, die beliebig ausgewählt werden können
- ein einziger Computer erzeugt einen beliebigen Lösungsvorschlag nichtdeterministisch und trifft immer beste Entscheidung
- Erfolg nach polynomial vielen Schritten
- Lösung in zwei Phasen: „Guess“ und „Check“

Guess- und Check-Phasen



Guess- und Check-Phasen

- Guess-Phase geschieht in **nichtdeterministisch polynomialer** Zeit $p(n)$
- Check-Phase in **deterministisch polynomialer** Zeit $q(n)$ beendet
- Lösung spätestens zum Zeitpunkt $p(n) + q(n)$

$\Rightarrow$ *P: Probleme, die effizient gelöst werden können*

$\Rightarrow$ *NP: Probleme, deren Lösungskandidaten effizient überprüft werden können*

Die Klasse NP

Definition

Die Klasse der NP-Probleme ist die Menge aller Probleme, die nichtdeterministisch in Polynomzeit gelöst werden können.

Die Klasse NP

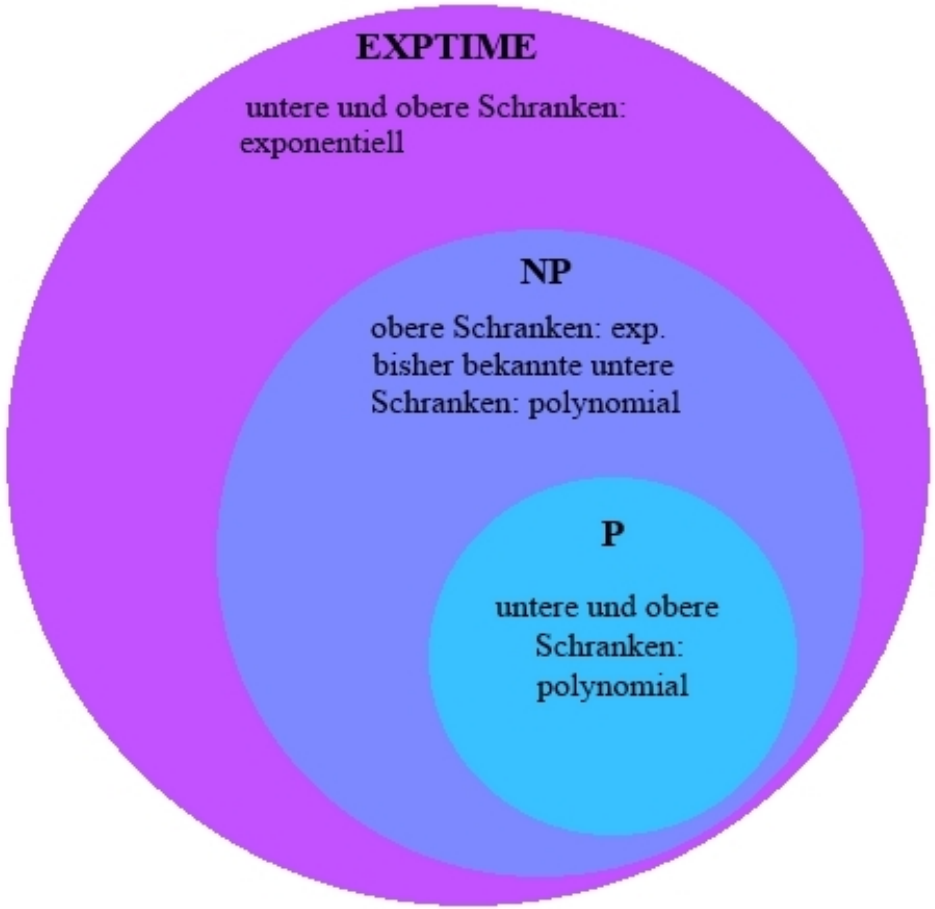
- es gilt $\mathbf{P} \subseteq \mathbf{NP}$
- Lösungen mit exponentiellem Aufwand

$\rightarrow$ bisher bekannte untere Schranken: linear, quadratisch und obere Schranken: (mehrfach) exponentiell

- Algorithmen sind ineffizient, wenn deren Komplexität nicht durch Polynome beschränkt werden können

$\rightarrow$ Suche nach effizienten Lösungsverfahren für NP-Probleme

Problemklassen P, NP und EXPTIME



Problemklassen P, NP und EXPTIME

- bisher nur $\mathbf{P} \subset \mathbf{EXPTIME}$ bewiesen
- $\mathbf{NP} \subset \mathbf{EXPTIME}$ oder $\mathbf{NP} = \mathbf{EXPTIME}$ noch nicht belegt
- nicht bekannt, ob jedes deterministisch in exponentieller Zeit lösbare Problem in **NP** liegt $\rightarrow$ $\mathbf{NP} \subseteq \mathbf{EXPTIME}$?

$\rightarrow$ es gibt berechenbare Probleme, die vermutlich nicht in **NP** liegen

Praktisch relevante Probleme

- Kombinatorik, Graphentheorie
- Schaltkreisentwurf
- Wirtschaftswissenschaften
- Logistik, Stadtplanung
- Bankgeschäft
- Telekommunikation

Zusammenfassung

- Probleme in der Klasse **P** zählen zu den *effizient* lösbaren Problemen
- Probleme in der Klasse **NP** zählen zu den schwierigen oder harten Problemen
- es gilt $\mathbf{P} \subseteq \mathbf{NP}$
- Nachweis, dass Problem zu **NP**, aber nicht zu **P** gehört, ist bis heute nicht gelungen
- $\mathbf{P} = \mathbf{NP}$ oder $\mathbf{P} \subset \mathbf{NP}$
- Vermutung: $\mathbf{P} \neq \mathbf{NP}$
- **P-NP-Problem:** Gilt $\mathbf{P} = \mathbf{NP}$?

$\rightarrow$ derzeit schwierigstes Problem der theoretischen Informatik

$\rightarrow$ zählt zu den *Millenium-Problemen*

Polynomiale Reduktion

- neues Problem x wird auf ein anderes, bereits gelöstes Problem f(x) zurückgeführt
- Entscheidungsproblem als Frage *„Ist Wort w in Sprache L?“*

Definition

Seien A und B Sprachen (Probleme) mit $A \subseteq \Sigma_1^*$ und $B \subseteq \Sigma_2^*$. Dann heißt A polynomial reduzierbar auf B, geschrieben: $\mathbf{A} \leq_p \mathbf{B}$, wenn es eine totale und mit polynomialer Komplexität berechenbare Funktion $f : \Sigma_1^* \to \Sigma_2^*$ gibt, sodass für alle $x \in \Sigma_1^*$ gilt: $x \in A \Leftrightarrow f(x) \in B$.

Polynomiale Reduktion

- $\leq_p$ zeigt Relation der Schwierigkeitsgrade von Problemen:
  - „A ist im Wesentlichen nicht schwieriger als B“ oder
  - „B ist mindestens so schwierig wie A“
- wenn A durch Einbettung von B und vorgeschalteter polynomialer Transformation ausgedrückt werden kann, dann gilt $\mathbf{A} \leq_p \mathbf{B}$
- wenn man bereits Lösung für B besitzt, dann bedarf es nur geringfügig einer größeren Zeitkomplexität für Lösung von A

Aussagen

Satz

Wenn $A \leq_p B$ und $B \in \mathbf{P}$, dann gilt $A \in \mathbf{P}$.

Satz

Wenn $A \leq_p B$ und $B \in \mathbf{NP}$, dann gilt $A \in \mathbf{NP}$.

Beweis

- Voraussetzung: DTM zur Entscheidung, ob $y \in \Sigma_2{}^*$ zu B gehört, deren Rechenzeit durch Polynom p_B nach oben beschränkt ist
- für polynomiale Reduktion f in A $\leq_p$ B ist TM durch Polynom p_f nach oben beschränkt
- zeitliche Aufwände für Transformation $p_f(x)$ und $p_B(f(x))$ für Entscheidung, ob $f(x) \in$ B, addieren

$\rightarrow$ Entscheidung, ob $x \in \Sigma_1{}^*$ zu A gehört

- gesamte Rechenzeit durch $p_f(x) + p_B(f(x))$ nach oben beschränkt $\rightarrow$ Funktion ist Polynom

$\Rightarrow$ **A** $\in$ **P** bewiesen

NP-schwere Probleme

- Extremfall: alle anderen Probleme aus **NP** sind polynomial auf A reduzierbar

$\rightarrow$ A ist ganz besonders schwierig

- durch transitive Eigenschaft der Ordnungsrelation $\leq_p$ auf Menge der Entscheidungsprobleme kann Nachweis vereinfacht werden

$\rightarrow$ aus A $\leq_p$ B und B $\leq_p$ C folgt A $\leq_p$ C

NP-schwere Probleme

Definition

Ein Problem (eine Sprache) heißt **NP-schwer**, wenn für alle Probleme (Sprachen) $L \in \text{NP}$ gilt: $L \leq_p \text{A}$.

Man beachte:

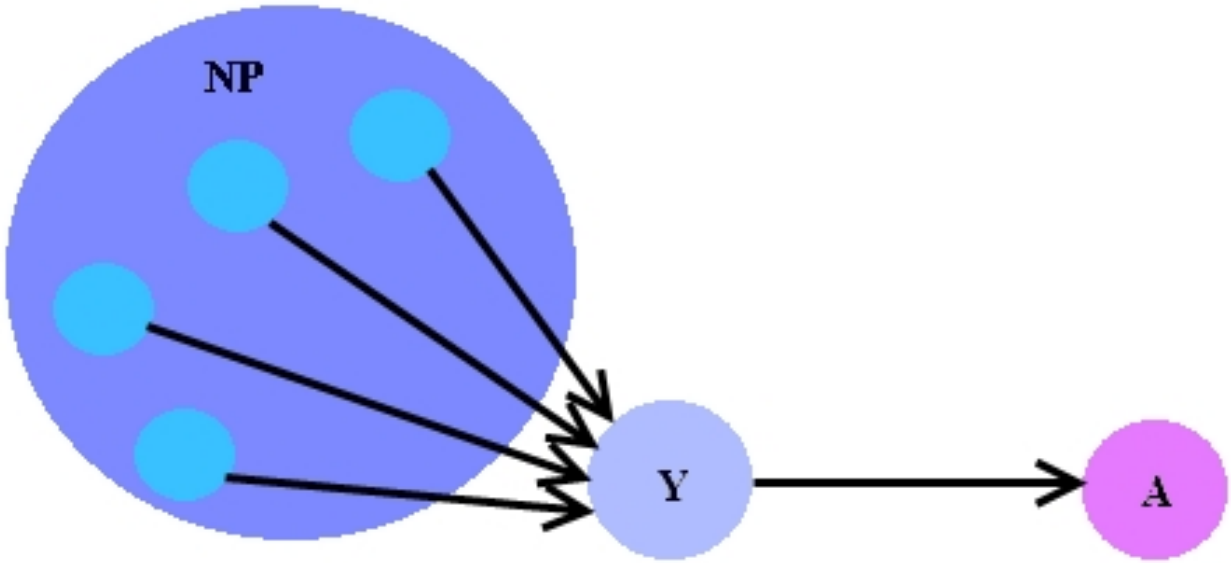
Ein NP-schweres Problem muss nicht notwendigerweise in **NP** liegen!

Nachweis: A ist NP-schwer

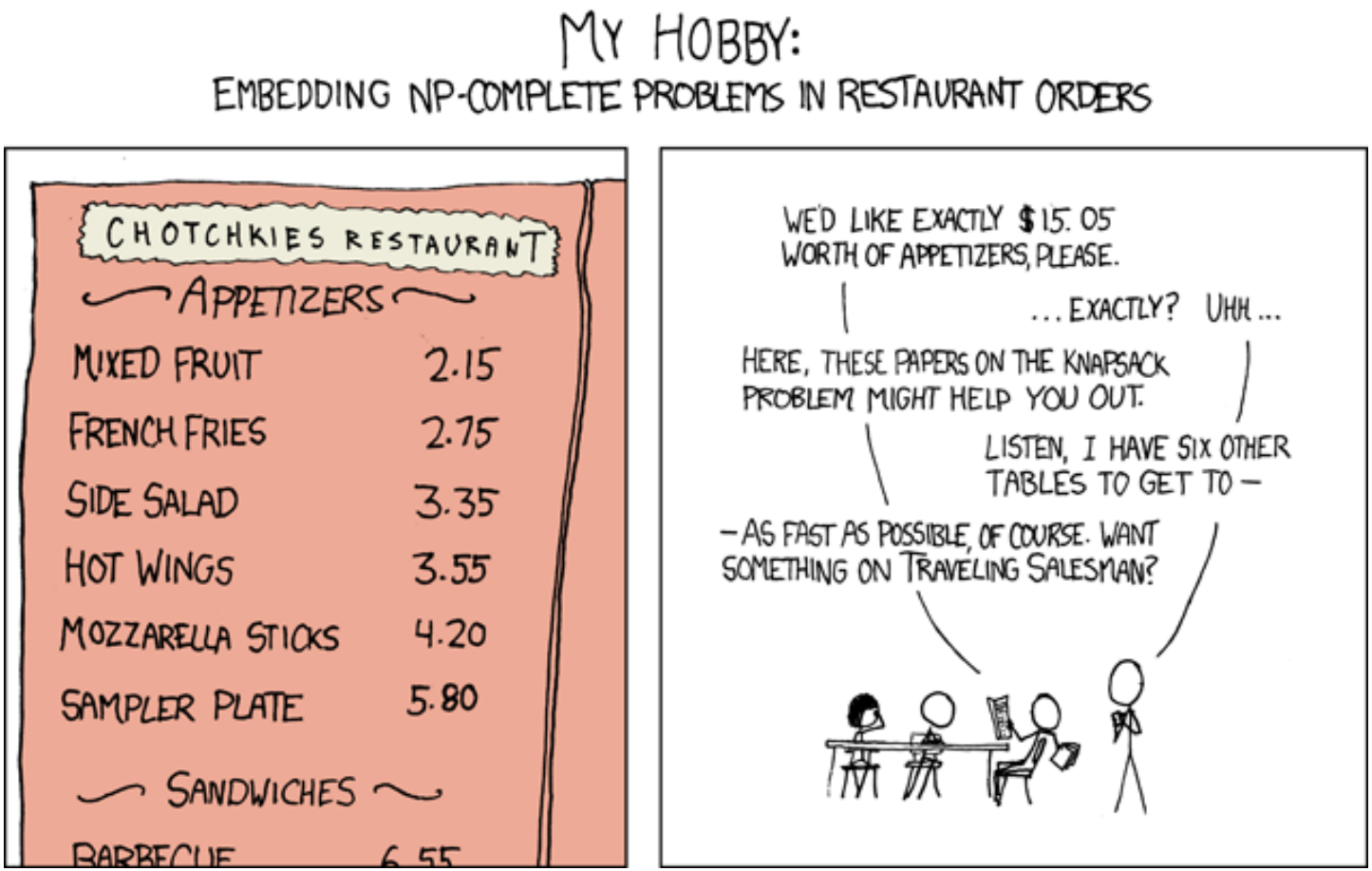
- Annahme: *alle* **NP**-Probleme sind auf A polynomial reduzierbar
- Nachweis mittels Transitivität von $\leq_p$
- *ein* (bekanntes) **NP-schweres** Problem Y ist gegeben
- für alle $L \in$ **NP** gilt: $L \leq_p Y$
- es wird gezeigt, dass Y polynomial reduzierbar auf A ist $\rightarrow$ $Y \leq_p A$

$\rightarrow$ aus $L \leq_p Y$ und $Y \leq_p$ A folgt $L \leq_p$ A

Nachweis: A ist NP-schwer



NP-Vollständigkeit



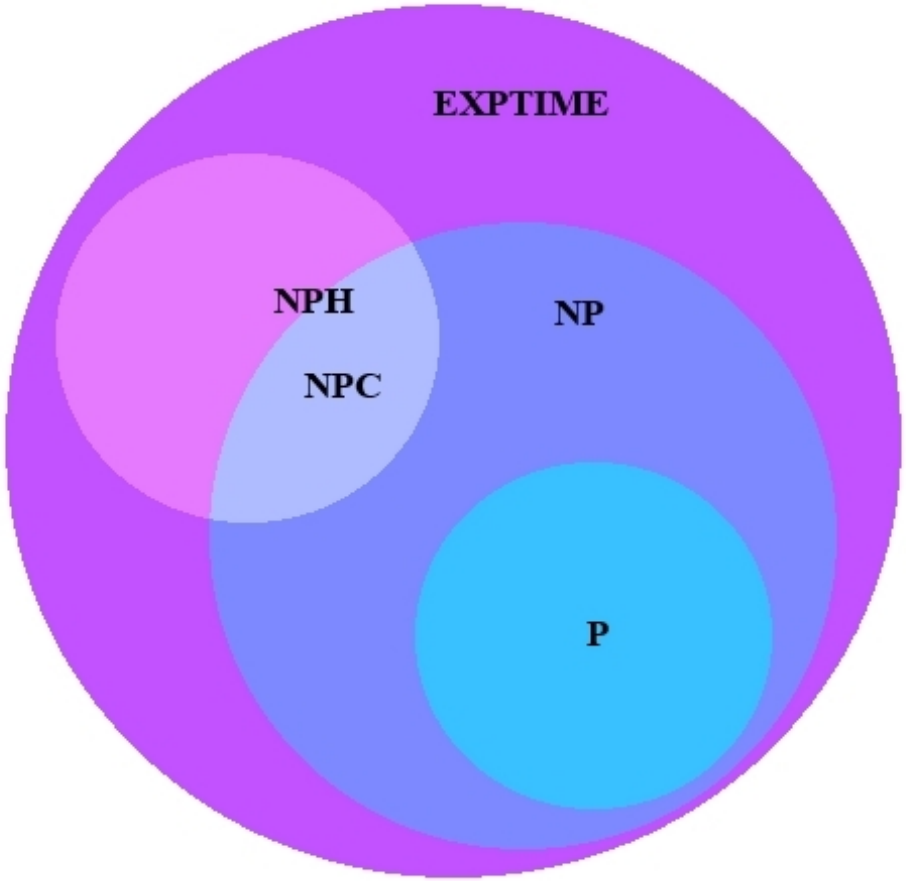
http://xkcd.com/287/

NP-Vollständigkeit

Definition

A heißt **NP-vollständig**, wenn A **NP-schwer** ist und A $\in$ **NP** gilt.

Relationen zwischen Problemklassen, falls P $\neq$ NP



Relationen zwischen Problemklassen

- **NPH** = Klasse der **NP-schweren** Probleme
- **NPC** = Klasse der **NP-vollständigen** Probleme
- Menge **NP**$\setminus$**(NPC** $\cup$ **P)** ist nicht leer
- man weiß nicht von allen **NP**-Problemen, ob sie **NP**-vollständig sind oder nicht
- es ist nicht bekannt, ob **NP = EXPTIME** und ob **P=NP**
- da **P** $\neq$ **EXPTIME**, gilt: **P** $\subset$ **NP** oder **NP** $\subset$ **EXPTIME** oder beides
- Vermutung: **P** $\subset$ **NP** $\subset$ **EXPTIME**

Relationen zwischen Problemklassen

- **Komplexitätstheorie:** Frage nach Existenz von Polynomzeitalgorithmen für Probleme, die nicht effizient gelöst werden können
- bis heute kein **NPC**-Problem in Polynomzeit gelöst

→ Vermutung: es gibt *keine* Polynomzeit-Lösungen für alle **NP**-Probleme

- Zuspitzung der Fragestellung: $\mathbf{P} = \mathbf{NP}$?

P = NP ?

1. nur für *ein einziges* **NP-vollständiges** Problem müsste es für dessen Lösung einen Algorithmus in **P** geben $\rightarrow$ dann gilt dies für alle **NP**-Probleme

Satz

Wenn A ein **NP-vollständiges** Problem ist, dann gilt: $\mathbf{A} \in \mathbf{P} \Leftrightarrow \mathbf{P} = \mathbf{NP}$.

2. nur für *ein einziges* **NP-vollständiges** Problem liegt die untere Schranke für den Berechnungsaufwand in $\mathcal{O}(k^n) \rightarrow$ dann gilt dies für alle **NP**-Probleme $\Rightarrow \mathbf{P} \subset \mathbf{NP}$

Nachweis der NP-Vollständigkeit eines Problems

- neues, noch einzustufendes **NP**-Problem B
- polynomiale Reduktion eines bereits bekannten **NP-vollständigen** Problems A auf B
  1. Zeige, das B $\in$ **NP** gilt.
  2. Wähle ein **NP-vollständiges** Problem A, das B ähnelt.
  3. Definiere eine polynomielle Transformation f, die Eingaben für A in Eingaben für B überführt.
  4. Zeige, dass $x \in A \Leftrightarrow f(x) \in B$ gilt.

Nachweis: TSP $\in$ NPC mittels HAMILTON-Kreis

- Voraussetzung: **NP-vollständiges** Problem des ungerichteten HAMILTON-Kreises (UDHC) steht zur Verfügung
- ungerichteter HAMILTON-Kreis = Rundweg durch einen ungerichteten Graphen, der jeden der n Knoten genau einmal durchläuft und zum Startknoten zurückkehrt
- jedem HAMILTON-Kreis wird mittels f bijektiv eine Rundreise zugeordnet

Man beachte:

Auch bei sich stark ähnelnden Problembeschreibungen müssen die Schwierigkeitsgrade zugehöriger Lösungsalgorithmen nicht übereinstimmen!

Beispiel: Euler-Kreis

- ungerichteter Euler-Kreis verwandt mit UDHC
- *Euler-Kreis* = Rundweg durch einen ungerichteten Graphen, der jede Kante genau einmal durchläuft und zum Startknoten zurückkehrt
- bekanntes Beispiel: *Königsberger Brücken*
- Graph besteht aus vier Punkten und sieben Kanten
- Punkte haben ungerade Anzahl von verbundenen Kanten
- Euler-Kreis kommt nur dann zustande, wenn es zu jeder hinführenden Kante auch eine wegführende gibt

$\rightarrow$ Anzahl der indizierenden Kanten für jeden Knoten muss gerade sein

$\Rightarrow$ es handelt sich **nicht** um einen Euler-Kreis

Schlussfolgerung

- Überprüfung des Grades eines jeden Knoten kann in Polynomzeit erfolgen
- Problem des Euler-Kreises liegt in **P** und nicht, wie vielleicht vermutet, in **NP** wie UDHC

$\Rightarrow$ Schwierigkeitsgrade stimmen nicht überein

Satz von Steven A. Cook

- es muss (wenigstens) *ein* **NP-vollständiges** Problem geben, dass nicht durch polynomiale Reduktion eines anderen, bereits vorhandenen NP-vollständigen Problems nachgewiesen wurde
- Frage nach dem „ersten“ NP-vollständigen Problems aufgeworfen
- **SAT** = erstes NP-vollständiges Problem (1971)

$\rightarrow$ **SAT** = Menge *aller* erfüllbaren aussagenlogischen Formeln

Satz

Das Erfüllbarkeitsproblem der Aussagenlogik (SAT) ist **NP-vollständig**.

3SAT-Problem

- **3SAT** = Vereinfachung des SAT-Problems
- Konjuktionen von Klauseln mit jeweils maximal drei Literalen
- Literal l = Variable x_i oder deren Negation $\neg x_i$
- Klausel K = Disjunktion von Literalen $l_1 \vee l_2 \vee \ldots \vee l_3$
- jedes **3SAT**-Problem ist auch ein **SAT**-Problem

$\rightarrow$ 3SAT $\leq$ SAT

- Umkehrung SAT $\leq$ 3SAT ist möglich: beliebiges SAT-Problem als 3SAT-Problem kodiert

Anwendungen

- *Kryptographie*: beschäftigt sich mit Methoden der Geheimhaltung von Daten
- *Kryptologie*: Wissenschaft von den (offenen) Geheimschriften

→ bildet mit Hilfe der Kryptoanalyse Vorschriften zur Entzifferung von Code

- Praxis: Ver- und Entschlüsselungsverfahren, die nicht oder nur sehr schwer lösbar sind

RSA-Verfahren

- erste Public-Key-Kryptosystem
- 1977 von **R**ivest, **S**hamir und **A**dleman erfunden
- öffentlicher Schlüssel C zur Verschlüsselung einer Nachricht besteht aus zwei Teilen m und e
- privater Schlüssel D zur Dechiffrierung enthält m und geheimen Teil d

Schlüsselerzeugung

- Bestimmung von m durch Multiplikation zweier möglichst großer Primzahlen p und q, d.h. $m = p * q$
- Primzahlen sollten mehr als 200 Stellen haben
- Ermittlung von $\varphi(m)$, welches die Anzahl der Zahlen von 1 bis $m - 1$ angibt, die zu m *teilerfremd* sind
- Es gilt für $p \neq q$: $\varphi(p * q) = (p - 1) * (q - 1)$
- Wahl einer zufälligen Zahl e, sodass $ggT(\varphi(m), e) = 1$
- Berechnung des geheimen Teiles $d = e^{-1} \mod \varphi(m)$

Beispiel

- $p = 17$ und $q = 19$
- $m = p * q = 17 * 19 = 323$
- $\varphi(m) = (p-1) * (q-1) = 16 * 18 = 288$
- zufällig: $e = 5$
- $d = e^{-1} \mod \varphi(m) = 5^{-1} \mod 288 = 173$
- öffentlich: (323,5) geheim: (323,173)

Hinweis: geheimer Teil d kann mit Hilfe des Euklidischen Verfahrens berechnet werden

Codierung und Decodierung

- Vorschriften:
  - **Verschlüsselung:** $x \longmapsto x^e \mod m$
  - **Entschlüsselung:** $y \longmapsto y^d \mod m$
- im Beispiel wird der Wert $x = 13$ für den zu verschlüsselnden Klartext gewählt
  - $y = 13^5 \mod 323 = 166$
  - $x = 166^{173} \mod 323 = 13$

Schlussfolgerung

- d nur berechenbar, wenn m bekannt ist

$\rightarrow$ folglich muss man Primzahlen p und q kennen, denn $m = p * q$

- aufgrund ihrer Größe sind die Primzahlen nur mit exponentiellem Aufwand bestimmbar
- Vorteil des RSA-Verfahrens: *Faktorisierung* von m

$\rightarrow$ dafür gibt es bisher keinen effizienten Algorithmus

$\Rightarrow$ macht Lösung einer Verschlüsselung praktisch unlösbar

Quellen & Literatur

- Wagenknecht, Christian: „Algorithmen und Komplexität“
- Gumm, H. und Sommer, M.: „Einführung in die Informatik“
- Cormen, Th. H. und Leiserson, Ch. E. und Rivest, R. und Stein, C.: “Algorithmen - Eine Einführung“

Vielen Dank für Ihre Aufmerksamkeit!

Gibt es noch Fragen?