10 NP-vollständige Probleme

Effiziente Algorithmen für **"gutartige" Probleme**

=

Algorithmen mit **polynomialer Laufzeit** (Klasse: **P**)

Laufzeit: O(P(n) P(n) = Polynom über der Eingabegröße n

Es gibt Probleme, für die keine Polynomzeit-Algorithmen bekannt sind.

Frage: Lohnt sich die Suche nach effizienten Lösungen?

Ziel: Identifikation von Problemen, die **nicht in P** liegen

Speziell: **NP-vollständige Probleme**

(NP = **N**icht-deterministisch in **P**olynomialer Zeit)

Äquivalenz aller NP-Probleme

Probleme, die in NP liegen, werden zu einer Klasse zusammengefasst, weil sie in folgendem Sinn alle **äquivalent** sind:

Für irgendein NP-vollständiges Problem gibt einen effizienten Algorithmus dann und nur dann, wenn es für alle NP-vollständigen Algorithmen effiziente Algorithmen gibt.

Hypothese: Es gibt keinen effizienten Algorithmus für irgend-ein NP-vollständiges Problem (bis heute nicht bewiesen)

Zunächst definieren wir die Klasse der NP-vollständigen Probleme. Dann gehen wir der Frage nach:

Wie beweist man, dass ein Problem NP-vollständig ist?

Polynomzeit-Reduktionen

Wir beschränken uns auf **Entscheidungsprobleme**, d.h. es geht nur um **ja/nein-Antworten**, z.B.: gibt es in einem Graph G einen Euler'schen Weg oder nicht?

Transformation: Entscheidungsproblem
$\Leftrightarrow$ Erkennung einer formalen Sprache

Sei **U** die Menge der möglichen Eingaben für ein Entscheidungsproblem. Sei $\mathbf{L} \subseteq \mathbf{U}$ die Menge aller Eingaben mit **ja**-Antwort. Die Sprache **L** repräsentiert das Problem. Sei **w** ein Eingabewort.

Entscheidungsproblem: $\mathbf{w} \in \mathbf{L}$ oder $\mathbf{w} \notin \mathbf{L}$?

Polynomzeit-Reduzibilität

Definition (Polynomzeit-Reduzibilität)

Seien L_1 und L_2 zwei Sprachen über den Input-Universen U_1 und U_2. L_1 ist ***polynomial reduzibel*** zu L_2, wenn jeder Input $u_1 \in U_1$ in einen Input $u_2 \in U_2$ in Polynomzeit konvertiert werden kann, so dass gilt:

$$\mathbf{u_1 \in L_1 \Leftrightarrow \ u_2 \in L_2}$$

Der Algorithmus hängt polynomial von der Größe des Inputs u_1 ab.

Vorgehensweise

Gesucht: Algorithmus für L_1

Bekannt: Algorithmus für L_2: **AL2**

Transformation $L_1 \rightarrow L_2$: **AC**

Input-Universum

Lösung: Gegeben ein beliebiger Algorithmus $u_1 \in U_1$; wir benutzen **AC**, um u_1 nach $u_2 \in U_2$ zu konvertieren; dann benutzen wir **AL2**, um zu entscheiden ob u_2 zu L_2 gehört, was uns sagt, ob u_1 zu L_1 gehört.

Zwei Theorem zur Polynomzeit-Reduzibilität

Theorem 1

Ist L_1 polynomial reduzibel zu L_2 und gibt es einen Polynomzeit-Algorithmus für L_2, dann gibt es einen Polynomzeit-Algorithmus für L_1.

Zwei Sprachen L_1 und L_2 heißen **polynomial äquivalent**, oder einfach äquivalent, wenn jede zur anderen polynomial reduiziert werden kann. Insbesondere sind alle nicht-trivialen "gutartigen" Probleme äquivalent, weil sie alle durch einen Polynomzeit-Algorithmus gelöst werden können.

Theorem 2 (Transitivität der Polynomzeit-Reduzibilität)

Ist L_1 polynomial reduzibel zu L_2, und ist L_2 polynomial reduzibel zu L_3, dann ist L_1 polynomial reduzibel zu L_3.

Nicht-Determinismus und Cook's Theorem

Die Theorie der NP-Vollständigkeit geht auf das Theorem von Cook (1971) zurück. Wir untersuchen hier nicht die Theorie selbst, sondern die Anwendung der Theorie auf praktische Probleme.

Um den Begriff des **Determinismus** zu verstehen benötigt man ein **Modell der Berechenbarkeit**, z.B. eine **Turing-Maschine** oder **Entscheidungsbäume**. Die Modelle sind äquivalent.

Uns interessiert die Frage:

Was ist ein nicht-deterministischer Algorithmus ?

Modell der freien Wahl beim Nicht-Determinismus

Nicht-Determinismus ist ein Modell, um Komplexitätseigenschaften von Algorithmen zu untersuchen.

Der Unterschied zwischen einem **deterministischen** und einem **nicht-deterministischen Algorithmus** liegt in der Art und Weise, wie dieser die Sprache erkennt.

Sei **nd-choice** eine Funktion, die zwischen einer festen Zahl von Wahlmöglichkeiten entscheidet.

Ein nicht-deterministischer Algorithmus **erkennt** eine Sprache L, wenn gilt:

Bei gegebener Eingabe **x** kann jede während der Ausführung des Algorithmus vorkommende *nd-choice* durch eine reale Wahl ersetzt werden, so dass gilt

der Algorithmus akzeptiert **x** $\Leftrightarrow$ **x** $\in$ **L**

Turingmaschinen und Nicht-Determinismus

Die Mächtigkeit nicht-deterministischer Algorithmen beruht darauf, dass man ihnen eine abstrakte Wahlmöglichkeit lässt. Nicht-Determinismus lässt sich z.B. mit Turingmaschinen simulieren.

Die Klasse der **nicht-deterministischen** Algorithmen mit **polynomialer** Laufzeit bezeichnet man mit

NP

Bis heute ungelöste Frage

Die Frage, die in der Informatik bis heute (2009) ungelöst ist lautet:

Sind nicht-deterministische Algorithmen mächtiger als deterministische Algorithmen?

Anders ausgedrückt:

Können mit nicht-deterministischen Algorithmen mehr Probleme gelöst werden als mit deterministischen Algorithmen, d.h.

$$\mathbf{P \subset NP \quad ?}$$

Lösungsmöglichkeit

Eine Möglichkeit, **P ⊂ NP ?** zu beweisen besteht darin, ein Problem zu finden, das in **NP** aber nicht in **P** liegt, d.h. ein Problem, das **nicht-deterministisch in Polynomzeit**, **deterministisch** aber **nicht in Polynomzeit** gelöst werden kann. Dies ist bis heute nicht gelungen!!

Umgekehrt könnte man auch versuchen zu beweisen, dass die beiden Komplexitätsklassen gleich sind, d.h.

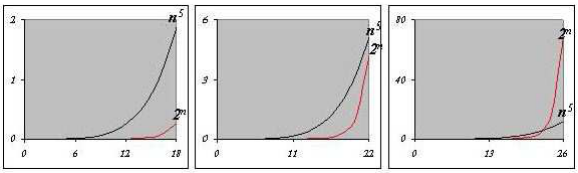
P = NP ?

Wenn dagegen P=NP gilt, kann jedes bisher der NP-Klasse zugeordnete Problem auch von einem deterministischen Algorithmus in polynomialer Zeit gelöst werden. Die **P=NP-Frage** ist wesentlich, wenn man entscheiden will, ob man überhaupt hoffen kann, "gute" Algorithmen für NP-Probleme zu finden. Falls P<>NP gilt (d.h. jemand kann das beweisen) hat man leider verloren :-(.

Erfüllbarkeits-Problem **SAT** (engl.: satisfiability)

Erfüllbarkeit: gibt es eine Gästeliste, die mindestens eine Präferenz jeder Person erfüllt?

Gästeliste
Debra
Elroy
Ben
Costas



Horiz. Achse: n = Anzahl der Präferenzen;
Vertik. Achse: Zeit für die Berechnung einer Gästeliste.

Erfüllbarkeitstheorem von Stephen Cook

Sei S ein boolscher Ausdruck in **konjunktiver Normalform (CNF)**. Das heißt, S ist das Produkt (log. $\wedge$) mehrerer Summen (log. $\vee$)

Beispiel: $S = (x \wedge y \wedge \neg z) \vee (\neg x \wedge y \wedge z) \vee (\neg x \wedge \neg y \wedge \neg z)$.

Jeder Boolsche Ausdruck kann in **CNF** transformiert werden. Ein Boolsche Ausdruck heißt **erfüllbar** (engl.: **satisfiable**), wenn es eine Instanziierung der boolsche Variablen mit 0 (= false) und 1 (= true) gibt, so dass der Wert des Aus-drucks gleich 1 ist. Das **SAT**-Problem besteht darin zu entscheiden, ob ein gegebener boolscher Ausdruck erfüllbar ist (ohne notwendigerweise eine Instanziierung der Variablen zu finden). Der Ausdruck S ist z.B. für x = 1, y = 1 und z = 0 erfüllbar.

Cook's Theorem: Das SAT-Problem ist NP-vollständig