Zeitkomplexität II - Die Klasse NP

Vortrag im Rahmen des Proseminars Theoretische Informatik WS 10/11

von Max Wisniewski

Abstract

Die Klasse NP stellt eine obere Schranke der Laufzeit, der in ihr enthaltenen Algorithmen dar. Viele dieser Algorithmen haben einen großen Nutzen im Alltag eines Informatikers, doch da noch keine effiziente Lösung von allen Problemen dieser Klasse gefunden wurde, ist es von besonderem Interesse, diese Klasse weiter zu untersuchen.

1 Hamilton-Wege

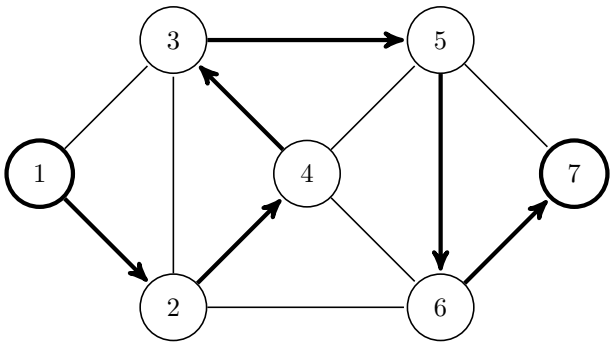
Ein Hamiltonweg ist ein Weg in einem Graphen, in dem jeder Knoten genau einmal passiert wird.

$$HAMPATH = \{\langle G, s, t\rangle \mid G \text{ ist ein ungerichteter Graph der einen Hamilton Weg von s nach t enthält}\}$$

Um dieses Problem zu lösen, kann man den Wegalgorithmus von s nacht t benutzen und so durch Brute-Force einen Weg erhalten, der exponentielle Laufzeit benötigt.
Angenommen, jemand gäbe uns einen vermeintlichen Hamilton-Weg, könnten wir auf folgende Weise prüfen, ob dies wirklich ein solcher Weg ist:

1. Teste, ob s Startpunkt und t Endpunkt ist.
2. Überprüfe, ob in der Folge alle Knoten genau einmal vorkommen.
3. Überprüfe, ob alle Wege ziwischen diesen Knoten existieren.

Beispiel:



$$G = (\{1,2,3,4,5,6,7\}, \{\{1,2\},\{1,3\},\{2,3\},\{2,4\},\{2,6\},\{3,4\},\{3,5\},\{4,5\},\{4,6\},\{5,6\},\{5,7\},\{6,7\}\})$$

Existiert ein Hamilton-Weg von 1 nach 7?
Proof: $1-2-4-3-5-6-7$ ist ein korrekter Weg.
Beweis:

1 und 7 sind die von uns gewählten Start- bzw. Endpunkte.
Die Pfade $1-2$, $2-4$, $4-3$, $3-5$, $5-6$, $6-7$ sind in unserem Graphen enthalten.
Es werden alle 7 Knoten besucht.
$\Rightarrow$ Der angegebene Weg ist ein Hamilton-Weg.

2 Definition

Def. 1: Ein Verifikator für eine Sprache A ist ein Algorithmus V, mit $A = \{w \mid \exists c \text{ String } : V \text{ akzeptiert } \langle w, c\rangle\}$.

Die Laufzeit eines Verifikators wird nur über die Länge von w gemessen. c dient hier als Zertifikat (engl.: certificate) oder Beweis (engl.: proof), dass w in A liegt.

Def. 2: NP ist die Klasse der Sprachen, für die ein Verifikator existiert, der in polynomieller Zeit ein Wort akzeptiert.

Def. 3: NP ist die Klasse der Sprachen, die durch eine nichtdeterministische Turingmaschine in polynomieller Zeit akzeptiert werden.

Beweis.: Def. 2 $\triangleq$ Def. 3
Sei $A \in NP$

" $\Rightarrow$ " Nach Def. 2 existiert ein (polynomieller Zeit) Verifikator V, der Laufzeit n^k benötigt um A zu akzeptieren.
z.z.: $\exists$ N NTM die A akzeptiert.
Konstruiere N (mit Eingabe w):
1. Erstelle nichtdeterministisch einen String c mit höchstens der Länge n^k
2. Lasse V auf $\langle w, c\rangle$ laufen.
3. N akzeptiert genau dann, wenn V akzeptiert hat.
N braucht höchstens n^k Schritte um den String zu erzeugen und V hat eine Laufzeit von n^k. Damit wird auch N höchstens $2n^k$ Schritte benötigen und akzeptiert das Wort damit in polynomieller Zeit.

" $\Leftarrow$ " Nach Def. 3 existiert eine NTM N, die Laufzeit n^k benötigt um A zu akzeptieren.
z.z.: $\exists V$ (polynomieller Zeit) Verifikator, der A akzeptiert.
Laufe V mit Eingabe $\langle w, c\rangle$.
1. Simuliere einen Durchlauf auf N, wobei an jeder nichtdeterministischen Verzweigung das nächste Symbol aus c als Wegweiser verwendet wird um den "richtigen" Weg zu wählen.
2. V akzeptiert die Eingabe genau dann, wenn dieser Weg aus N die Eingabe akzeptiert hat.
Da wir uns durch das Zertifikat für genau einen Weg entscheiden, ist die Berechnung deterministisch und weil N in seinem "richtigen" Weg eine polynomielle Laufzeit hat, wird auch V diese haben.

Wir können zu jedem Verifikator eine nichtdeterministische Turingmaschine konstruieren und umgekehrt, die ein Wort in der selben polynomiellen Laufzeit akzeptieren. Die beiden Definitionen sind somit äquivalent. ■

3 Laufzeit und Einordnung

NTIME(t(n)) $:= \{L \mid$ L ist eine Sprache, die in $O(t(n))$ Schritten von einer NTM entschieden wird$\}$

NP $= \bigcup_{i \in \mathbb{N}} \text{NTIME}(n^i)$

P $\subseteq$ NP: Dies gilt offensichtlich, da jede deterministische Turingmaschine gleichzeitig eine nichtdeterministische ist.
Ihr bleibt nur die Wahl zwischen einem Schritt.

NP $\subseteq$ PSPACE: PSPACE ist die Klasse der Turingmaschinen, die mit polynomiellen Speicherverbrauch auskommen.

4 Abschlusseigenschaften

Die Beweise lassen sich leicht auf die Beweise von P zurückführen, wenn man Verifikatoren betrachtet. Wir beweisen aber nochmal konkret die Eigenschaften für NTM.
Seien $A, B \in NP$

4.1 $A \cup B \in NP$

Beweis: Seien T_a, T_b Turingmaschinen mit $L(T_a) = A$ und $L(T_b) = B$.

Konstruiere $T_\cup$ folgendermaßen: $T_\cup$ ist eine 2-Band-Turingmaschine.
1. Schritt: Kopiere die Eingabe auf das 2. Band
2. Schritt: Lasse T_a einen Schritt auf Band 1 ausführen, akzeptiert T_a lasse $T_\cup$ akzeptieren.
3. Schritt: Lasse T_b einen Schritt auf Band 2 ausführen und verfahre analog zu Schritt 2
4. Schritt: Fahre mit Schritt 2 fort.

z.z.: $T_\cup$ akzeptiert w in polynomieller Zeit

Sei $w \in (A \cup B)$ Eingabe, mit $|w| = n$.
Nach Voraussetzung sind $A, B \in NP$. Sei $t_a(x)$ Polynom, das die Laufzeit von T_a beschreibt und $t_b(x)$ Polynom, dass die Laufzeit von T_b beschreibt.
1. Schritt: $T_\cup$ kopiert w auf Band 2 und zurückfahren$\Rightarrow$ Laufzeit $2n$
Schleife: In jedem Durchlauf werden 3 Schritte ausgeführt.

Fall 1 $w \in A$ (Annahme $w \notin B$)

Die Schleife wird $t_a(x)$ -mal ausgeführt. $\Rightarrow T_\cup$ braucht $3 \cdot t_a(w) + 2n$ Schritte

Fall 2 $w \in B$ analog zu Fall 1 $\Rightarrow T_\cup$ braucht $3 \cdot t_b(w) + 2n$ Schritte

Sei $t_{min} = \min\{t_a, t_b\}$. $\Rightarrow T_\cup$ braucht $3 \cdot t_{min}(w) + 2n$ Schritte.
Da t_{min} Polynom ist und bei der Addition und derMuliplikation von einem Polynomen und einem Faktor wieder ein Polynom ist, ist auch die Laufzeit von $T_\cup$ ein Polynom.
$\Rightarrow L(T_\cup) = A \cup B \in NP$

4.2 $A \cap B \in NP$

Beweis: Konstruktion

Seien T_a, T_b Turingmaschinen mit $L(T_a) = A$ und $L(T_b) = B$.
Konstruiere $T_\cap$ als 2-Band-Turingmaschine.
1. Schritt: Kopiere die Eingabe auf Band 2 und fahre zurück.
2. Schritt: Lasse T_a auf Band 1 laufen. Wird die Eingabe akzeptiert, fahre fort.
3. Schritt: Simuliere T_b auf dem 2. Band.
4. Schritt: Hat T_b akzeptiert, so gehe in einen gesamt akzeptierenden Zustand über.

Laufzeit Sei $t_a(x)$ Laufzeit von T_a und $t_b(x)$ Laufzeit von T_b. Da beide akzeptieren müssen damit $T_\cap$ akzeptiert $\Rightarrow$Laufzeit von $T_\cap$ ist $t_a(x) + t_b(x) + 2 \cdot |x|$.

4.3 $A \cdot B \in NP$

Beweis: Konstruktion

Konstruiere T' als 2-Band-Turingmaschiene.
Sei Eingabe w, $|w| = m$
und T_a, T_b Turingmaschinen mit $L(T_a) = A$ und $L(T_b) = B$
1. Schritt: Teile w nichtdeterministische in w_l, w_r mit $w_l \cdot w_r = w$, kopiere w_r auf Band 2 und fahre zurück.
2. Schritt: Lasse T_a auf Band 1 laufen, akzeptiert T_a fahre mit Schrit 3 fort.
3. Schritt: Lasse T_b auf Band 2 laufen.
4. Schritt: Hat T_b die Eingabe akzeptiert, so akzeptiere das Wort.

Laufzeit Im 1. Schritt kopieren wir maximal die komplette Eingabe auf Band 2. Sei t_a polynomielle Laufzeit von T_a und t_b polynomielle Laufzeit von T_b. Damit ist die Laufzeit von T': $t_a(w) + t_b(w) + 2 \cdot |w|$.

4.4 $A^* \in NP$

Beweis: Konstruktion

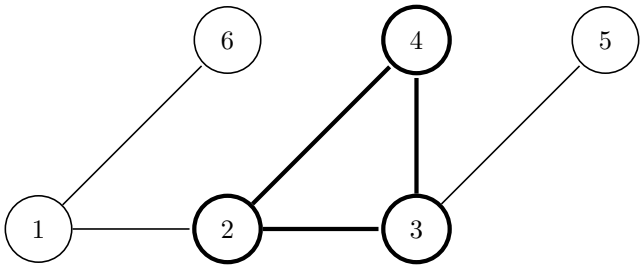
Konstruiere T' als 2-Band-Turingmaschine.
Sei Eingabe w, $|w| = m$ (Anzahl der verbleibenden Zeichen in Durchlauf 0)
und T_a Turingmaschinen mit $L(T_a) = A$.
1. Schritt : Ist Band 1 leer, akzeptiere das Wort. Kopiere von n verbleibenden Zeichen l $(l \leq n)$ von Band 1 auf Band 2.
2. Schritt : Simuliere T_a auf dem 2. Band.
3. Schritt : Hat T_a akzeptiert, fahre mit Schritt 1 fort.
4. Schritt: Akzeptiere.

Laufzeit: Es wird die gesamte Eingabe auf Band 2 geschrieben. Dies benötigt $2|w|$ Schritte(mit zurückfahren). Das 2. Band muss immer wieder gelöscht werden, da aber $NP \subseteq PSPACE$ ist, kann das Löschen auch nur wieder eine polynomielle Anzahl Schritten benötigen. Im 2. Schritt läuft T_a auf den Teileingaben.
Da das Kopieren und Löschen polynomielle Zeit benötigt und durch Addition mit den Laufzeiten von T_a wiederum ein Polynom herrauskommt, ist $T' \in NP$.

5 Veranschaulichung

Cliquen-Problem Gibt es in einem Graphen k paarweise untereinander verbundene Knoten?

Beispiel: Gibt es in diesem Graphen $G = (\{1,2,3,4,5,6\}, \{\{1,2\},\{1,6\},\{2,3\},\{2,4\},\{3,4\},\{4,5\}\})$ eine 3er Clique?
Proof: 2,3,4 ist eine Clique.



Wir sehen ersteinmal, dass 2,3,4 wie verlangt drei Knoten sind. Aus dem Bild wird uns ersichtlich, dass alle miteinander verbunden sind. Ein TM wird das in polynomieller Zeit bestätigen.

Kompositum Lässt sich eine natürliche Zahl n als Produkt zweier natürlicher Zahlen darstellen? Dies ist das Komplement zum Primzahlproblem.

Um dies zu Überprüfen, verwendet man am Leichtesten einen Verifikator.
Sei $KOMP$ die Sprache, die alle Komposita enthält.
Bsp:
Eingabe: 12, Proof : 3,4 $\Rightarrow 3 \cdot 4 = 12 \Rightarrow 12 \in KOMP$
Eingabe: 45, Proof : 4,9 $\Rightarrow 4 \cdot 9 = 45 \Rightarrow 45 \in KOMP$

6 Außerhalb von NP

Nicht alle Probleme liegen in NP, sind also nicht nichtdeterministisch in polynomieller Zeit lösbar.
Das einfachste Beispiel hierfür ist die Aufgabe, 2^n 1en auf das Band der Turingmaschine zu schreiben bei Eingabe n.

Beweis Sei $EINS$ die die oben beschriebene Sprache und n die Eingabe. Selbst unter der Annahme, dass wir in jedem Schritt eine 1 schreiben (mehr ist nicht möglich, da sich der Kopf nur an einer Stelle gleichzeitig befinden kann), müssen wir immer noch 2^n Schritte ausführen um 2^n 1en zu schreiben.
$\Rightarrow \forall T$ Turingmaschine : $L(T) = EINS \Rightarrow$ Laufzeit ist exponentiell
$\Rightarrow EINS \notin NP$

Quellen

1. Sipser, Michael. Introduction to the Theory of Computation. 2. Auflage. Boston : Course Technology, 2006
2. Wegener, Ingo. Kompendium Theoretische Informatik - eine Ideensammlung. 1. Auflage. Stuttgart : B.G. Teubner Stuttgart , 1996
3. NP (complexity) - Wikipedia, the free encyclopedia. Wikimedia Foundation, Inc. [Online] November 2010. http://en.wikipedia.org/wiki/NP_(complexity)