Proseminar Theoretische Informatik 12.01.2016

Die Komplexitätsklasse NP

Elmar Frerichs, Michael Tran Xuan *Wolfgang Mulzer*

1 Einführung

Bei vielen Problemen lässt sich zwar eine Lösung leicht überprüfen, wenn man sie gefunden hat, aber das Finden einer Lösung ist sehr schwierig. Dazu zählen viele Logikspiele wie Sudoku sowie das Beweisen von Sätzen selbst. Diese Äquivalenzklasse von Sprachen wird NP genannt.

Sie umfasst gleichzeitig genau die Sprachen, deren Laufzeit eines nichtdeterministischen Automaten (hier meist einer nichtdeterministischen Turingmaschine, NTM) ein Polynom ist. NP (Nichtdeterministisch Polynomiell) stellt damit das Gegenstück zur Klasse P (Polynomiell) dar, die für deterministische endliche Automaten definiert ist.

2 Definition

2.1 Verifizierer und Lösungswort

Definition 1. *Ein Problem ist in der Klasse NP, wenn es für dessen Sprache L einen effizient lösbaren Verifizierer V gibt, für den gilt:*

$$w \in L \Leftrightarrow \exists\, l : V \text{ akzeptiert } (w, l) \text{ mit } |l| = |w|^k \text{ für festes } k \in \mathbb{N} \tag{1}$$

Wir geben dem Verifizierer also eine mögliche Lösung l, um herauszufinden, ob l die richtige Lösung für w ist. Dabei darf l nicht bedeutend (nicht mehr als polynomiell) länger als sein zugehöriges w sein.

Hier wird keine Aussage darüber getroffen, wie man ein l für ein gegebenes w finden kann.

2.2 Algorithmen für Nichtdeterministische Turingmaschinen

Eine weitere Möglichkeit der Definition ist ein ähnlicher Weg, wie wir P definiert haben, mit $t(n)$ als Laufzeit des kürzesten akzeptierenden Ausführungsastes bei einer Wortlänge n:

Definition 2. *Sei $t : \mathbb{N} \to \mathbb{R}^+$ eine Funktion und NTIME($t(n)$) die Klasse aller Sprachen, die durch eine $\mathcal{O}(t(n))$-zeitbeschränkte nichtdeterministische Turingmaschine mit einer Eingabe der Länge n entscheidbar ist. Dann ist NP definiert über*

$$NP := \bigcup_{i \leq 1} NTIME(n^i) \tag{2}$$

Alle Sprachen, die durch eine nichtdeterministische Turingmaschinen in polynomieller Zeit entschieden werden können, sind Teil von NP. Dies schließt offensichtlich auch P ein, da deterministische Turingmaschinen ohne Mehraufwand durch eine nichtdeterministische Turingmaschinen dargestellt werden können.

2.3 Äquivalenz der Definitionen

2.3.1 NTM → Verifizierer

Satz 3. *Für jede Sprache L und deren nichtdeterministische Turingmaschine M gibt es einen Verifizierer V, der für eine Eingabe $w \in L$ bestimmen kann, ob w von M akzeptiert wird.*

Beweis. Gegeben ist eine nichtdeterministische Turingmaschine M mit $L(M) = L$. Ohne Beschränkung der Allgemeinheit nehmen wir an, dass jede Überführungsrelation von M nur 2 Elemente hat, die wir 0 und 1 nennen. l soll nun eine Anweisung für den Ausführungsbaum von M sein.

Konstruktion des Verifizierers: Der Verifizierer bekommt als Eingabe (w, l) und simuliert die M für den Pfad l. An jeder Abzweigung wird mit l entschieden, ob Zweig 0 oder 1 gewählt wird. Akzeptiert M, akzeptiert auch V. Die Länge von l ist dabei die Anzahl der Schritte für diesen Weg. Es gibt also für den kürzesten Weg ein l mit Länge $t(|w|)$, falls das Wort in der Sprache ist.

Korrektheit: Gemäß Konstruktion gilt:

$$\begin{aligned} w \in L &\Leftrightarrow M \text{ akzeptiert } w &(3)\\ &\Leftrightarrow \exists\, l \in \{0,1\}^{t(|w|)} : V \text{ akzeptiert } (w, l) &(4)\end{aligned}$$

Der Verifizierer kann die Rechnung mit polynomiellem Zeitverlust ausführen. Damit sind die geforderten Eigenschaften erfüllt. □

2.3.2 Verifizierer → NTM

Satz 4. *Für jede Sprache L mit einem Verifizierer V gibt es eine nichtdeterministische Turingmaschine M, sodass gilt: $L = L(M)$.*

Beweis. Gegeben sein ein Verifizierer V mit (deterministisch) polynomieller Laufzeit, für den gilt:

$$w \in L \Leftrightarrow \exists\, l \in \{0,1\}^*, |l| < |w|^k : V \text{ akzeptiert } (w, l) \quad (5)$$

Konstruktion von M mit zwei Schritten:

1. M erstellt nichtdeterministisch die Lösung l für w
2. M simuliert V für (l, w) und akzeptiert, falls V akzeptiert.

Korrektheit:

- M erkennt die Sprache L, weil gilt

$$\begin{aligned} w \in L &\Leftrightarrow \exists\, l \in \{0,1\}^*, |l| < |w|^k : V \text{ akzeptiert } (w, l) &(6)\\ &\Leftrightarrow \text{Es gibt einen akzeptierenden Rechenweg in } M \text{ für } w &(7)\\ &\Leftrightarrow M \text{ akzeptiert } w &(8)\end{aligned}$$

- Die Laufzeit von M ist nichtdeterministisch polynomiell beschränkt, da

– die Laufzeit in Schritt 1 der Länge des Lösungsworts entspricht, und
– die Laufzeit in Schritt 2 der Laufzeit des Verifizierers entspricht.

Beide sind polynomiell beschränkt.

□

3 Beispiele für NP-Probleme

3.1 Der Handlungsreisende (TSP)

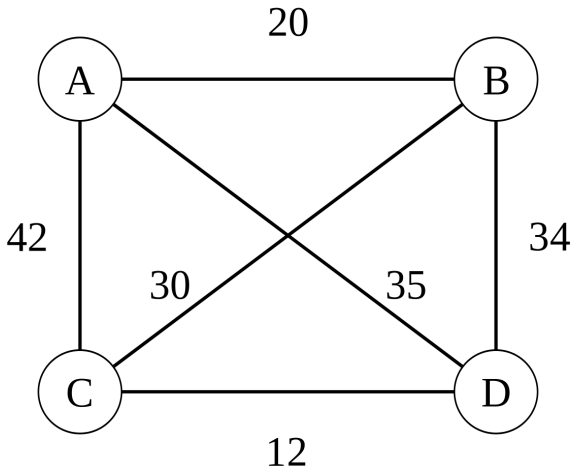

Abbildung 1: Graph des Handlungsreisenden

Das Problem des Handlungsreisenden (*Traveling Salesman Problem*, TSP) ist eines der bekanntesten Probleme der Komplexitätsklasse NP.

Beschreibung: Ein Händler will durch die größten Städte seines Landes reisen, um dort jeweils seine Waren anzubieten. Dabei will er – als Vorzeigeökonom – selbstverständlich die kürzeste mögliche Reisestrecke nehmen. Am Ende seiner Reise will er wieder in seiner Heimatstadt ankommen, es wird also eine Rundreise.

Kurz zur Größenordnung: Angenommen der Händler wohnt in einer der größten 15 Städte Deutschlands, dann gibt es 43.589.145.600 Routen. Intuitiv kann man sich also schon vorstellen, wieso es nicht ganz so einfach – sprich eventuell unmöglich – ist einen Algorithmus mit Polynomfunktion Laufzeit zu finden.

Praktische Anwendungen für dieses Problem sind Pannendienste, Postdienste, Lieferdienste, eine Reise durch eure Lieblingsstädte in Europa...

Mathematische Abstraktion: Wir stellen das Problem als vollständigen Graphen dar. Die Städte sind die Knoten, die wir mit gewichteten Kanten verbinden. Dann heißt das, dass wir

innerhalb dieses Graphen den kürzesten Weg finden wollen, der jeden Knoten *genau* einmal abläuft. Dies ist auch als Hamiltonkreisproblem bekannt.

Eine beispielhafte Darstellung mit vier Städten und den Strecken zueinander ist in Abbildung 1 gezeigt.

Das Problem kann man auf [1] mit diversen Karten und einer beliebigen Anzahl zwischen 3 und 50 Städten per Hand nachspielen.

3.2 Zahlenaufteilungsproblem (PARTITION)

Gegeben sei eine Menge Y mit

$$Y = \{y_1, ..., y_n\} \text{ mit } y_i \in \mathbb{N} \tag{9}$$

Gesucht sind zwei Mengen Y_1, Y_2 mit $Y_1 \cup Y_2 = Y$, sodass gilt:

$$\sum_{y \in Y_1} y = \sum_{x \in Y_2} x \tag{10}$$

Beispiel: Gegeben sein ein $Y = \{1, 2, 3, 4, 5, 10, 11, 15, 29\}$. Dann gibt es $Y_1 = \{1, 2, 3, 4, 5, 10, 15\}$ und $Y_2 = \{11, 29\}$, deren Summe jeweils 40 ergibt.

Satz 5. *PARTITION liegt in NP.*

Beweis. Wir haben zwei Partionen Y_1 und Y_2 gegeben und bauen einen Verifizierer V für eine Eingabe (Y, Y_1, Y_2).

1. Teste, ob die Summe aller Zahlen aus Y_1 und die Summe alle Zahlen aus Y_2 gleich sind
2. Teste, ob $Y_1 \cup Y_2 = Y$ gilt
3. Akzeptiere, falls beide Tests *wahr* ausgeben

□

3.3 Teilsummenproblem (SUBSET-SUM)

Gegeben sei eine Menge von Zahlen $Z = \{z_1, ..., z_n\}$ sowie eine Zielzahl S mit $z_i, S \in \mathbb{N} \; \forall \; i \in \mathbb{N}$.

Gesucht ist ein $Z' \subseteq Z$ mit

$$\sum_{z \in Z'} z = S \tag{11}$$

Beispiel: Wir haben $Z = \{4, 11, 16, 21, 27\}, S = 25$ gegeben. In diesem Fall gibt es das Subset $Z' = \{4, 21\}$, denn $4 + 21 = 25$.

Satz 6. *SUBSET-SUM liegt in NP.*

Beweis. Wir haben ein Subset c und bauen einen Verifizierer V für eine Eingabe $((Z, S), c)$.

1. Teste, ob c ein Subset von Zahlen ist, deren Summe S ergibt

[1] `https://www-m9.ma.tum.de/games/tsp-game/index_de.htm`

2. Teste, ob Z alle Zahlen aus c enthält

3. Akzeptiere, falls beide Tests *wahr* ausgeben

□

Alternativer Beweis. Beweis über eine NTM ohne ein gegebenes Subset
Sei N eine nichtdeterministische Turingmaschine mit Eingabe (Z, S).

1. Wähle nicht-deterministisch ein Subset c aus den Zahlen in Z

2. Teste, ob c ein Subset von Zahlen ist, deren Summe S ergibt

3. Akzeptiere, falls 2. *wahr* ausgibt

□

Wir halten fest: Um die Zugehörigkeit eines Problems zu NP zu beweisen, haben wir durch die beiden äquivalenten Definitionen zwei Möglichkeiten der Beweisführung: Wir können einen Verifizierer oder eine nichtdeterministische Turingmaschine bauen.

4 Einordnung

4.1 Abgrenzung zwischen P, NP und PSPACE

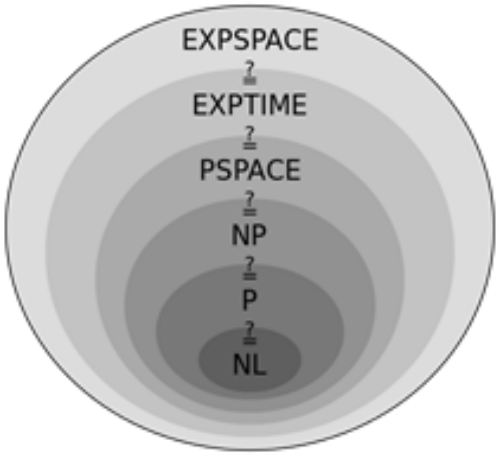

Abbildung 2: NP in Relation zu ähnlichen Klassen

4.1.1 Abgrenzung von P und NP

Satz 7. *Es gilt $P \subseteq NP$: Jedes Problem der Klasse P ist auch ein Problem der Klasse NP.*

Beweis. Gemäß dem Fall, dass eine deterministische Turingmaschine ein Problem in polynomieller Zeit lösen kann (Problem in P), kann auch eine nicht-deterministische Turingmaschine das gleiche Problem in polynomieller Zeit lösen (Definition 2 von NP), da es sich bei einer DTM um eine spezielle NTM handelt.

Für die Abgrenzung beider Klassen nutzen wir Definition 1 von NP:

1. P ist die Klasse von Problemen, wo eine Lösung in polynomieller Zeit *gefunden* werden kann.
2. NP ist die Klasse von Problemen, wo eine Lösung in polynomieller Zeit *bestätigt* werden kann.

□

Intuitiv lässt sich viel mehr polynomiell bestätigen, als polynomiell Lösungen gefunden werden können. Bestätigen ist nach dem Alltagsverständnis einfacher als Lösen. Ein mathematischer Beweis ist bisher nicht jedoch nicht gelungen. Es gibt kein einziges Problem in NP für das es einen formalen Beweis gibt, dass es nicht in P liegt (P-NP-Problem).

4.1.2 Abgrenzung von NP und PSPACE

PSPACE ist eine Form von Komplexitätsklassen, die wir bisher nicht kennen gelernt haben. Bisher kennen wir P und NP, die beide die Zeit einschränken, in der ein Problem gelöst werden kann. PSPACE gehört (ebenso wie EXPSPACE und NL, welche wir in einem gesonderten Vortrag kennen lernen werden) zu den Komplexitätsklassen, die sich auf den Platzverbrauch eines Algorithmus zur Lösung bezieht.

Definition 8. *Sei M eine Turingmaschine, die eine Sprache L akzeptiert. Dann ist das Problem der Sprache L in PSPACE, falls der Platzbedarf von M polynomiell ist.*

Satz 9. *Es gilt NP $\subseteq$ PSPACE: Alle Probleme in NP können mit polynomiellem Platzbedarf gelöst werden.*

Beweisidee: Für ein einziges beliebiges NP-vollständiges Problem muss gezeigt werden, dass es in PSPACE liegt, also dass der Algorithmus zur Lösung polynomiellen Platzbedarf hat.

Es gibt jedoch auch Probleme, die in PSPACE liegen, aber nicht in NP. Von den sogenannten PSPACE-vollständigen Problemen wird angenommen, dass sie nicht in polynomieller Zeit lösbar sind.

4.2 Bedeutung des P-NP-Problems

Es ist eines der größten bisher ungelösten mathematischen Probleme, ob die Klasse NP äquivalent zu P ist. Bisher wurden zwar für diverse Probleme bewiesen, dass sie in NP liegen, und es sind sogar einige bekannt, mit denen man alle anderen NP-Probleme simulieren könnte (NP-Vollständigkeit). Es wurde bisher aber weder bewiesen noch widerlegt, dass die Klassen P und NP disjunkt sind, auch wenn ein Großteil der wissenschaftlichen Gemeinschaft davon ausgeht, dass P $\subsetneq$ NP gilt.

Die Beantwortung dieser Frage hat auch Auswirkungen außerhalb der Mathematik und der theoretischen Informatik. Kryptographische Verfahren benötigen die Zertifikate (Lösungen) aus NP-Problemen, die leicht zu überprüfen, aber schwer zu fälschen sind. Über die häufig genutzte Primfaktorzerlegung ist allerdings schon bekannt, dass sie mit speziell dafür gebauten Quantencomputern in polynomieller Zeit lösbar sein könnte. Würde NP allerdings in P liegen, müssten hier völlig neue Verfahren entwickelt werden.

Quellen

- Anil Maheshwari und Michiel Smid: Introduction to Theory of Computation, 2014
- Michael Sipser: Introduction to the Theory Computation, 1997
- Uwe Schöning: Theoretische Informatik - kurz gefasst, 2008
- Carsten Lutz, letzter Zugriff 10.01.2016
  `http://www.informatik.uni-bremen.de/tdki/lehre/ss09/kt/Kapitel3.pdf`
- Berthold Vöcking, letzter Zugriff 10.01.2016
  `http://algo.rwth-aachen.de/Lehre/WS0910/VBuK/Folien/komplexitaet1.pdf`