Proseminar Komplexitätstheorie
P versus NP

Wintersemester 2006/07

Vortrag am 17.11.2006

Nichtdeterministische Turingmaschinen und NP

Yves Radunz

Inhaltsverzeichnis

1 Wiederholung

Im letzten Vortrag sind die deterministischen Turingmaschinen bereits ausführtlich behandelt worden. Daher möchte ich an dieser Stelle nur eine kurze Zusammenfassung dieses Themas anführen.

1.1 Allgemeines

Turingmaschinen sind exakte mathematische Modelle, um die Funktionsweise eines Algorithmus (bzw. den Algorithmus selbst) besser beschreiben zu können. Dabei kann man jedem Algorithmus eine Turingmaschine zuordnen, welche den Algorithmus simuliert.
Außerdem kann man mit Hilfe von Turingmaschinen bessere Aussagen über die Berechenbarkeit eines Problems (z.B. einer speziellen Funktion) machen.
Damit sind Turingmaschinen lediglich formale Darstellungen eines Algorithmus. (Schon auf Grund des beliebig langen Rechenbandes ist es praktisch unmöglich, eine echte Turingmaschine zu bauen.)

1.2 Definitionen

Definition (deterministische Turingmaschine)
Eine Turingmaschine ist ein Tupel $(\Gamma, \Sigma, b, Q, \delta, q_0, E)$. Hierbei ist:

1. Γ das Bandalphabet, d.h. alle Zeichen bzw. Symbole, die auf das (unendliche bzw. ausreichend große) Band geschrieben werden können.

2. $\Sigma \subseteq \Gamma$ das Eingabealphabet. Diese Menge ist das Alphabet, welches der betrachteten Sprache zu Grunde liegt.

3. $b \in \Gamma \setminus \Sigma$ das Leerzeichen. Dieses Symbol steht in jeder Zelle des Bandes, bevor etwas anderes (z.B. das Eingabewort) darauf geschrieben wird.

4. Q die endliche Zustandsmenge.

5. $\delta : Q \times \Gamma \to Q \times \Gamma \times \{\pm 1\}$ die Übergangsfunktion. Sie gibt an, wie sich der Automat, bzw. die (deterministische) Turingmaschine verhält, wenn er ein bestimmtes Zeichen in dem aktuellen Zustand liest. Hierbei wird der Folgezustand festgelegt, sowie das Zeichen, dass auf das Band geschrieben wird und die Bewegung der Maschine. $+1$ entspricht ein Feld weiter und -1 ein Feld zurück. Bei einigen Interpretationen wird auch 0 zugelassen, was dem Stehenbleiben der TM entspricht.

6. $q_0 \in Q$ der Startzustand. In diesem Zustand befindet sich die Maschine, bevor sie anfängt, das Eingabewort zu verarbeiten.

7. $E \subseteq Q$ die Menge der Endzustände.

   In unserer Darstellung einer Turingmaschine sollte $E = \{q_N, q_Y\}$ gelten, wobei der erste Zustand darstellt, dass die Turingmaschine das Wort nicht akzeptiert. Im zweiten Zustand hingegen akzeptiert sie es.

   Weil die Turingmaschinen eine andere Darstellung von Algorithmen sind, müssen sie nach endlich vielen Berechnungsschritten einen Endzustand erreichen. (Anderenfalls lassen wir die Turingmaschine nicht zu, obwohl es durchaus Überführungsfunktionen gibt, die derartige Maschinen beschreiben.)

Definition (erkannte Sprache, Zeikomplexität)
Mit $L_M = \{x \in \Sigma^* | M \text{ akzeptiert } x\}$ bezeichnen wir die Sprache, die von der Turingmaschine M definiert bzw. erkannt wird.
Die Zeit $m_T(x)$ bezeichne hier die Anzahl der Berechnungsschritte, die eine Turingmaschine zur Verarbeitung eines Wortes benötigt.
Dann ist $T_M(n) = \max\{m | \exists x \in \Sigma^*, |x| = n : m_T(x) = m\}$ die maximale Zeit, welche von der Maschine M benötigt wird, um ein Wort der Länge n zu verarbeiten.
M heißt zeitpolynomiale Turingmaschine, wenn es ein Polynom p mit $\forall n \in \mathbb{N}_+ : T_M(n) \leq p(n)$ gibt.
Nun definieren wir die Klasse P als die Klasse aller Probleme Π, für welche eine Kodierung e und somit eine Sprache $L = L[\Pi, e]$ existiert, welche die Eigenschaft $L_M = L$ erfüllt. M bezeichnet hierbei eine zeitpolynomiale Turingmaschine.

2 Nichtdeterministische Algorithmen

2.1 Problembeispiele

TSP reloaded: Noch einmal der Handelsreisende
Bevor wir weitere formale Definitionen vornehmen, befassen wir uns mit dem Problem des Handelsreisenden, bzw. dem Traveling Salesman Problem (kurz TSP). Das Problem an sich ist bereits bekannt: Wir suchen den kürzesten Weg, der die gegebenen Städte verbindet.
Auch mit dem dazugehörigen Entscheidungsproblem haben wir uns bereits befasst: Gibt es einen Weg, der nicht länger als b ist?
Nun nehmen wir an, jemand behauptet, das Entscheidungsproblem positiv gelöst zu haben, d.h. er meint, einen derartigen Weg gefunden zu haben. Für uns ist es an dieser Stelle leicht, zu erkennen, ob dieser Weg wirklich die Anforderungen erfüllt. Dazu müssen wir lediglich überprüfen, ob der Weg alle Städte verbindet und die Gesamtlänge b nicht überschreitet.
Weiterhin können wir dies sogar algorithmisch überprüfen. Der Algorithmus, der das Ergebnis auf Korrektheit testet, hat dann offensichtlich nur polynomielle Komplexität.

Isomorphe Teilgraphen
Ein weiteres Beispiel eines Problems mit dieser Eigenschaft ist das Folgende:
Für zwei Graphen $G_i = (K_i, V_i)$ mit $i = 1, 2$ ist zu entscheiden, ob eine Teilmenge $G' \subseteq G_1$ des ersten Graphen isomorph zum zweiten Graphen ist.
Auch hier ist es leicht zu überprüfen, ob ein gegebener Teilgraph G' von G_1 wirklich isomorph zu G_2 ist.

2.2 Nichtdeterministische Algorithmen

Definition (Vermutung)
In den zuvor beschriebenen Beispielen haben wir bereits gezeigt, dass es ausreicht, eine bestimmte Struktur (z.B. einen Weg im TSP) anzugegeben, um nachzuweisen, dass ein bestimmtes Entscheidungsproblem mit „ja“ beantwortet werden kann.
Wenn jemand eine derartige Beispielstruktur gefunden hat, kann man leicht überprüfen, ob durch diese Struktur wirklich eine bestimmte Instanz eines Problems mit „ja“ beantwortet werden kann. Diese Beispielstruktur möchte ich im Folgenden als Vermutung bezeichnen, da jemand (oder etwas) diese Struktur als vermutete Lösung erzeugt und dann auf Korrektheit testen lässt.
Natürlich gibt es auch Probleme, wo man keine derartige Struktur angeben kann. Zum Beispiel existiert für die folgende Frage keine derartige Möglichkeit:
Gibt es für eine gegebene Menge von Städten keinen Weg, der alle Städte verbindet und nicht länger als b ist?

Definition (Nichtdeterministische Algorithmen)
Wie wir an Hand der Beispiele gesehen haben, kann man für viele Probleme einen polynomiellen Algorithmus beschreiben, der überprüft, ob eine Vermutung (z.B. ein Weg im TSP) die gewünschte Eigenschaft hat. Nichtdeterministische Algorithmen kann man daher durch zwei Schritte beschreiben:

1. Raten

   Die erste Stufe erhält die Instanz des Problems und rät eine Beispielstruktur. Diese wird dann mit der Instanz des Problems an die zweite Stufe übergeben.

2. Überprüfen

   Wenn wir eine derartige Struktur geraten haben, können wir mit einem polynomiellen Algorithmus bestimmen, ob uns diese Vermutung das Entscheidungsproblem mit „ja“ beantworten lässt.

Man beachte, dass die Länge der Vermutung durch ein Polynom in der Länge der Eingabe begrenzt sein muss. Anderenfalls bräuchte der 2. Schritt zu lange, um die Vermutung einzulesen und zu testen. Es kann jedoch durchaus Probleme geben, für welche sich diese Vermutungen nicht derartig beschränken lassen.

2.3 Vorläufige Beschreibung von NP

Beschreibung
Wir können also NP als die Klasse aller Probleme beschreiben, für die bei einer gegebenen Instanz und einer Vermutung ein polynomieller Algorithmus existiert, der entscheidet, ob mit Hilfe dieser Struktur das Problem mit „ja“ beantwortet werden kann.

Vergleich mit P
Zunächst beachte man, dass es einen grundlegenden Unterschied zwischen einer Berechnung mit einem deterministischen Algorithmus und der Berechnung mit einem nicht deterministischen Algorithmus gibt: Die Berechnung mit einem nichtdeterministischen Algorithmus ist keine Berechnung im konventionellen Sinne, sondern eher eine Beschreibung, wie eine vermeintliche Lösung auf Korrektheit zu prüfen ist.
Außerdem gibt es eine andere wichtige Eigenschaft, welche diese beiden Arten von Algorithmen voneinander unterscheidet:
Die Frage ob die Aussage X für eine bestimmte Instanz I wahr ist, ist bei deterministischen Algorithmen genau so schwer mit „ja“ zu beantworten, wie die Frage, ob X für I falsch ist. Dies erkennt man, indem man bei der zugehörigen deterministischen Turingmaschine die Endzustände q_Y und q_N vertauscht.

Bei nicht deterministischen Algorithmen jedoch gilt dies nicht immer:
Betrachten wir zum Beispiel das TSP. Wie uns bereits bekannt ist, kann man mit einem polynomiellen Algorithmus testen, ob ein Weg wirklich kürzer als eine gegebene Schranke b ist.
Nun betrachten wir das komplementäre Problem: Existiert kein Weg, der nicht länger als b ist?
In diesem Fall bleibt uns bei einer bejahenden Antwort nichts anderes übrig, als alle möglichen Wege zu testen. Daher ist dieses Problem auch mit nicht deterministischen Algorithmen nicht in polynomieller Zeit zu lösen.

3 Nichtdeterministische Turingmaschinen

Nun möchten wir uns mit der Darstellung von nichtdeterministischen Algorithmen mit Hilfe der Turing-Maschinen widmen. Bisher haben wir meist nur von Turingmaschinen gesprochen, ohne explizit zu erwähnen, dass diese deterministisch arbeiten. Ab sofort müssen wir jedoch zwischen den deterministischen und den undeterministischen Turingmaschinen unterscheiden.
Wie eine deterministische Turingmaschine (DTM) arbeitet eine nichtdeterministische Turingmaschine (NDTM) auf einem unendlich (oder zumindest ausreichend großen) Band, dass in einzelne Zellen zerlegt ist. Die Instanz eines Problems wird dabei als ein Wort über dem Eingabealphabet Σ auf das Band geschrieben, bevor die Turingmaschine mit der Bearbeitung beginnt.

3.1 Zweiteilige NDTM

Es gibt verschiedene Möglichkeiten, eine NDTM darzustellen. Zunächst werde ich „unsere“ NDTM beschreiben.

Aufbau
Analog zu unserem Verständnis von nichtdeterministischen Algorithmen können wir auch die NDTM in zwei Teile zerlegen:

1. Vermutungseinheit

   Dieser Teil der NDTM ist mit einem Schreibkopf ausgestattet, welcher vor der eigentlichen Verarbeitung des Eingabewortes links von diesem eine beliebige Vermutung beliebiger Länge aufschreibt. Diese Vermutung muss dabei jedoch sinnvoll kodiert sein, damit ihre Länge, wie schon erwähnt, durch ein Polynom in der Länge des Eingabewortes beschränkt ist.

   Irgendwann hält diese Einheit an und hört auf zu schreiben. Dann beginnt die zweite Einheit der Turingmaschine zu arbeiten.

2. DTM

   Der zweite Teil der NDTM ist eine deterministische Turingmaschine, welche genau so arbeitet, wie es im Vortrag über DTMs beschrieben wurde. Wichtig ist hier jedoch, dass ihr nicht nur das Eingabewort (also die spezielle Instanz des Problems), sondern auch die Vermutung zu Verfügung steht.

Arbeitsweise
Bevor der zweite Teil der NDTM mit der Arbeit beginnt, wird eine beliebige Vermutung $s \in \Gamma^*$ auf das Band geschrieben. Weil die Vermutungseinheit das Eingabewort nicht verarbeitet, kann sie auch keine bestimmte Vermutung zu der speziellen Instanz auf das Band schreiben. Daher schreibt sie lediglich Wörter aus Γ^* auf das Band. Die Turingmaschine testet dann zunächst, ob die Vermutung gültig ist. Wenn nicht, kann sie sofort in den Zustand q_N übergehen.
Die DTM arbeitet anschließend parallel alle möglichen Vermutungen ab. Sie akzeptiert dann ein Eingabewort, wenn eine Vermutung existiert, sodass sie den Zustand q_Y erreicht. Wenn es keine derartige Vermutung gibt, geht sie immer in den Zustand q_N über und akzeptiert das Wort daher nicht.

3.2 Einteilige NDTM

Eine andere Interpretation ist eine Turingmaschine, welche bis auf die Überführungsfunktion wie eine DTM aufgebaut ist.
Die Überführungsfunktion ist jedoch eine Abbildung $\delta : Q \times \Gamma \to \mathcal{P}(Q) \times \Gamma \times \{\pm 1\}$. Das bedeutet, dass die Turingmaschine abhängig von ihrem aktuellen Zustand und dem eingelesenen Zeichen möglicherweise in mehrere Folgezustände übergehen kann.
Sie akzeptiert letztendlich ein Wort, wenn es irgendeine Möglichkeit gibt, sich bei jedem Übergang so zu entscheiden, dass sie letztendlich den Endzustand q_Y erreicht.

3.3 Definitionen

Analog zu den deterministischen Turingmaschinen werden wir jetzt einige Definitionen einführen. Diese Definitionen beziehen sich ausschließlich auf „unsere“ Turingmaschine. Für andere Darstellungen müssen sie ggf. entsprechend abgewandelt werden.

Definition (Sprache einer NDTM)
Es sei $M = (\Gamma, \Sigma, b, Q, \delta, q_0, E)$ eine nichtdeterministische Turingmaschine.
Dann ist $L_M = \{x \in \Sigma^* | M \text{ akzeptiert } x\}$ die Sprache, die von M erkannt wird.

Definition (Verarbeitungszeit)
Die Zeit, die eine (nichtdeterministische) Turingmaschine benötigt, um ein Wort zu erkennen, definieren wir als die minimale Länger aller Berechnungsprozesse, welche die Turingmaschine in einen akzeptierenden Zustand überführen.
Daher setzen wir
$m_M(x) = \min\{n \in \mathbb{N}_+ | \exists \text{ Vermutung } S : \text{Die Turingmaschine erreicht nach } n \text{ Schritten den Zustand } q_Y .\}$.

Definition (Zeitkomplexität)
Da wir auch bei nichtdeterministischen Turingmaschinen von einer Zeitkomplexität sprechen möchten, definieren wir diese wieder über die Anzahl der Schritte, welche von der Turingmaschine benötigt werden, um ein Wort der Länge n zu erkennen. Wenn die Turingmaschine kein Wort der Länge n erkennen kann, dann setzen wir den Zeitaufwand für dieses n auf 1.
Wir definieren daher $T_M(n) = \max\left(\{1\} \cup \{m \in \mathbb{N}_+ | \exists x \in L_M : m_M(x) = m\}\right)$.

3.4 Formale Definition von NP

Nachdem wir nun eine Möglichkeit gefunden haben, nichtdeterministische Algorithmen formal darzustellen, können wir NP als die Klasse aller Probleme beschreiben, die durch eine nichtdeterministische Turingmaschine mit polynomieller Zeitkomplexität gelöst werden können. Das bedeutet, dass bei diesen NDTMs die Zeitkomplexität durch ein Polynom beschränkt ist.
Ein Entscheidungsproblem Π (dargestellt durch eine Codierung e) gehört nun zu der Klasse NP, wenn die Sprache $L[\Pi, e]$ aus NP ist.