Komplexität des Maximums einer polynomialzeitbeschränkten Funktion

Seminar Reelle Komplexität
Prof. Dr. Martin Ziegler, Dr. habil. Ulrike Brandt
Ausarbeitung von Holger Thies
Sommersemester 2011

TECHNISCHE
UNIVERSITÄT
DARMSTADT

Inhaltsverzeichnis

1 Einleitung

Sowohl in der numerischen Analysis, als auch in der klassichen Komplexitätstheorie, lassen sich viele Probleme darauf zurückführen, das Maximum einer Funktion zu berechnen. In der diskreten Komplexitätstheorie wurde von einer Vielzahl von Entscheidungsproblemen, die solchen Maximierungsproblemen zugrundeliegen, NP-Vollständigkeit nachgewiesen (beispielsweise Clique, Knapsack,...).
Der vorliegende Text ist die Ausarbeitung zu einem Seminarvortrag für das Seminar "Reelle Komplexität", das im Sommersemester 2011 unter der gemeinsamen Leitung von Prof. Dr. Martin Ziegler und Dr. habil. Ulrike Brandt an der TU Darmstadt angeboten wurde. Die Seminararbeit beschäftigt sich mit Komplexität der Berechnung des Maximums einer reellwertigen Funktion. Genauer beschäftigen wir uns mit der Frage, ob das Maximum einer polynomialzeitberechenbaren Funktion in polynomieller Zeit berechnet werden kann.
In dieser Seminararbeit wird gezeigt, dass die Aussage, dass das Maximum jeder polynomialzeitberechenbaren Funktion in polynomieller Zeit berechnet werden kann, äquivalent zur Aussage $P = NP$ ist. Damit wird das oben beschriebene kontinuierliche Problem auf die klassische (diskretwertige) Komplexitätstheorie zurückgeführt.
Bei der Untersuchung von Berechenbarkeits- und Komplexitätsfragen über den reellen Zahlen ist zunächst einmal das verwendete Berechnungsmodell wichtig. Im Gegensatz zur klassischen Berechenbarkeits- und Komplexitätstheorie, gibt es hier nämlich bisher kein allgemein akzeptiertes Modell wie die Turingmaschine. Diese Ausarbeitung baut fast ausschließlich auf dem Buch "Complexity Theory of Real Functions" von Ker-i Ko auf und verwendet das dort beschriebene Berechnungsmodell. Im ersten Abschnitt wird dieses Modell kurz vorgestellt, so dass für das Lesen lediglich Grundkenntnisse aus der diskreten Komplexitätstheorie notwendig sind.

2 Berechenbarkeit und Komplexität über den reellen Zahlen

Bevor wir zum eigentlich Thema des Seminarvortrags kommen, sollen hier zunächst die wichtigsten Definitionen zur Berechenbarkeit und Komplexität reeller Zahlen und Funktionen zusammengefasst werden.

2.1 Berechenbarkeit reeller Zahlen

Als Berechnungsmodell betrachten wir Turingmaschinen über dem Alphabet $\Sigma = \{0, 1\}$. Eine solche Turingmaschine kann natürlich nicht direkt mit reellen Zahlen rechnen, da nicht jede reelle Zahl mit endlich vielen Zeichen aus Σ dargestellt werden kann. Eine reelle Zahl kann aber beliebig genau durch eine endliche Darstellung approximiert werden. Wir nutzen als Basis für die Approximation die Menge der dyadisch rationalen Zahlen $\mathbb{D} = \{m \cdot 2^{-n} \mid m \in \mathbb{Z}, n \in \mathbb{N}\}$.
Weiter führen wir noch die Schreibweise $\mathbb{D}_n$ für die Menge der dyadisch rationalen Zahlen mit maximal n Bits hinter dem Komma ein, d.h. $\mathbb{D}_n = \{m \cdot 2^{-n} \mid m \in \mathbb{Z}\}$. Eine reelle Zahl x kann nun dargestellt werden, als Folge dyadisch rationaler Zahlen, die gegen x konvergiert:

Definition 2.1. Sei $x \in \mathbb{R} \cap [0, 1]$. Die Funktion $\Phi : \mathbb{N} \to \mathbb{D}$ konvergiert binär gegen x, wenn $\Phi(n) \in \mathbb{D}_n$ und

$$|\Phi(n) - x| \leq 2^{-n} \text{ für alle } n \in \mathbb{N}.$$

Wir bezeichen die Menge aller Funktionen, die binär gegen x konvergieren mit $CF(x)$ (Cauchy-Funktion für x). Für ein $d \in \mathbb{D}$ schreiben wir b_d für die Standardcauchyfunktion für d, d.h. $b_d(n)$ liefert die (Standard-)Binärdarstellung von d bis zu den ersten n Nachkommastellen.

Wir wollen eine reelle Zahl als berechenbar bezeichnen, wenn es eine Turingmaschine gibt, die eine beliebig genaue dyadisch-rationale Approximation der Zahl ausgibt:

Definition 2.2. Eine reelle Zahl $x \in \mathbb{R}$ ist berechenbar, wenn es eine berechenbare Funktion $\Phi \in CF(x)$ gibt.

Wir haben somit die Berechenbarkeit reeller Zahlen auf den klassischen Berechenbarkeitsbegriff zurückgeführt.

2.2 Berechenbarkeit reeller Funktionen

Auch für die Berechnung von Funktionen werden wir dyadisch rationale Approximationen verwenden. Intuitiv soll eine Funktion $f : R \to \mathbb{R}$ berechenbar sein, wenn es eine Turingmaschine gibt, die beliebig genaue Approximationen $d \in \mathbb{D}$ an $f(x)$ liefert. Da die Eingabe x eine reelle Zahl ist, kann die Turingmaschine auch diese jedoch möglicherweise nicht in endlicher Zeit verarbeiten. Die Maschine kann aber beliebig exakte Approximationen an die Eingabe einlesen. Eigentlich soll die Maschine also eine unendliche Folge als Eingabe bekommen und diese beliebig weit lesen können, d.h. sie soll Anfragen der Form "Gib mir ein $d \in \mathbb{D}$ mit $|d - x| \leq 2^{-n}$" stellen können und darauf eine Antwort erhalten. Um dieses Modell zu formalisieren, benötigen wir zunächst die folgende Definition:

Definition 2.3. Eine Orakel-Turingmaschine mit Orakel Φ (Φ ist dabei eine Funktion $\Sigma^* \to \Sigma^*$) ist eine Turing-Maschine mit einem zusätzlichem Band, dem so genannten Anfrageband, und zwei zusätzlichen Zuständen, dem Frage- und Antwortzustand. Die Maschine verhält sich genau wie eine gewöhnlich Turingmaschine, solange sie sich nicht im Anfragezustand befindet. Wenn die Maschine in den Anfragezustand geht, wird der String w auf dem Anfrageband gelöscht und durch das Ergebnis der Orakelfunktion $\Phi(w)$ ersetzt. Danach geht die Maschine in den Antwortzustand über. Die Antwort des Orakels nimmt lediglich einen Rechenschritt in Anspruch. Wir bezeichnen mit M^Φ die mit dem Orakel Φ ausgestattete Turingmaschine M und mit $M^\Phi(n)$ das Ergebnis der Berechnung bei Eingabe n.
Orakel-Turingmaschinen mit mehr als einem Orakel lassen sich analog definieren (es gibt dann mehrere Anfragebänder).

Anmerkung 2.1. Auch wenn die Anfrage an das Orakel lediglich einen Rechenschritt in Anspruch nimmt, benötigt die Maschine trotzdem $|\Phi(w)|$ Schritte, um das Ergebnis der Anfrage auszulesen.

Die Berechenbarkeit einer Funktion $f : [0,1] \to \mathbb{R}$ lässt sich nun mithilfe von Orakel-Turingmaschinen definieren. Zur Vereinfachung betrachten wir nur Funktionen mit Definitionsbereich $[0,1]$.

Definition 2.4. Eine Funktion $f : [0,1] \to \mathbb{R}$ heißt berechenbar, wenn es eine Orakelturingmaschine M gibt mit

$$M^\Phi(n) \in \mathbb{D}_n \text{ und } |M^\Phi(n) - f(x)| \leq 2^{-n} \text{ für alle } n \in \mathbb{N} \text{ und alle } \Phi \in CF(x).$$

D.h. $M^\Phi \in CF(f(x))$ für alle $\Phi \in CF(x)$.
Mithilfe von Turingmaschinen mit mehreren Orakeln lässt sich diese Definition auf Funktionen $f : [0,1]^n \to \mathbb{R}$ verallgemeinern.

Anmerkung 2.2. Jede berechenbare reelle Funktion ist stetig.

Die Berechenbarkeit von Funktionen lässt sich noch auf eine andere Weise, ohne die Verwendung von Orakel-Turingmaschinen, charakterisieren. Wir benötigen dazu zunächst den Begriff der Modulfunktion:

Definition 2.5. Sei $f : [a,b] \to \mathbb{R}$ stetig. Eine Funktion $m : \mathbb{N} \to \mathbb{N}$ heißt Modulfunktion von f auf $[a,b]$, wenn für alle $x, y \in [a,b]$ gilt

$$|x - y| \leq 2^{-m(n)} \Rightarrow |f(x) - f(y)| \leq 2^{-n}.$$

Dies lässt uns den folgenden Satz formulieren:

Satz 2.1. Sei $f : [0,1] \to \mathbb{R}$ eine reellwertige Funktion.
Die folgenden beiden Aussagen sind äquivalent:

1. f ist berechenbar
2. f hat eine berechenbare Modulfunktion $m : \mathbb{N} \to \mathbb{N}$ auf $[0,1]$ und es gibt eine berechenbare Funktion $\Phi : (\mathbb{D} \cap [0,1]) \times \mathbb{N} \to \mathbb{D}$, so dass für alle $d \in \mathbb{D} \cap [0,1]$ und alle $n \in \mathbb{N}$ $|\Phi(d,n) - f(d)| \leq 2^{-n}$

2.3 Komplexität reeller Zahlen und Funktionen

Die Komplexität einer reellen Zahl lässt sich mithilfe der obigen Definitionen sehr einfach auf die klassische Komplexitätstheorie zurückführen:

Definition 2.6. Die (Zeit-)Komplexität einer reellen Zahl ist eine Funktion $T : \mathbb{N} \to \mathbb{N}$. Eine berechenbare reelle Zahl x hat Zeit-Komplexität $\leq T$, wenn es eine Turingmaschine M und ein $\Phi \in CF(x)$ gibt, so dass M bei Eingabe n in Unärdarstellung $\Phi(n)$ berechnet und die Zeitkomplexität von M durch T beschränkt ist.

Die Komplexität einer reellen Zahl x ist also die Anzahl der Schritte, die eine Turingmaschine benötigt, um x bis zu einer bestimmten vorgegebenen Genauigkeit zu approximieren.
Die Zeitkomplexität reellwertiger Funktionen lässt sich analog über die Komplexität von Orakelturingmaschinen definieren. Diese ist genau so definiert, wie die Komplexität gewöhnlicher Turingmaschinen. Es werden lediglich auch die Schritte auf dem Anfrageband mitgezählt.

Definition 2.7. Die Zeit-Komplexität einer rellwertigen Funktion ist eine Funktion $T : \mathbb{N} \to \mathbb{N}$. Eine berechenbare Funktion $f : [0,1] \to \mathbb{R}$ hat Zeit-Komplexität $\leq T$, wenn es eine Orakelturingmaschine M gibt, die f berechnet und für alle $\Phi \in CF$ und $n \in \mathbb{N}$ gilt $M^\Phi(n)$ wird in $\leq T(n)$ Schritten berechnet.

Reelle Zahlen und Funktionen lassen sich damit analog zur diskreten Komplexitätstheorie in Komplexitätsklassen einteilen. Im weiteren benötigen wir lediglich die Klasse $P_{C[0,1]}$ der polynomialzeitberechenbaren Funktionen $f : [0,1] \to \mathbb{R}$. Wir führen zusätzlich die Notation $P_{C^\infty[0,1]}$ für die Klasse der polynomialzeitberechenbaren und unendlich oft stetig differenzierbaren Funktionen ein.

2.4 Berechenbarkeit von Funktionalen

In den vorherigen Abschnitten haben wir Berechenbarkeit und Komplexität reeller Zahlen und Funktionen betrachtet. In dieser Seminararbeit soll es aber um das Problem gehen, das Maximum einer berechenbaren Funktion zu finden. D.h. wir wollen das Funktional $max : C[0,1] \to \mathbb{R}$ untersuchen, das eine gegebene Funktion auf den Wert ihres Maximums in $[0,1]$ abbildet. Bevor wir uns also mit der Komplexität des Maximums einer Funktion beschäftigen können, benötigen wir zunächst ein formales Berechnungsmodell für solche numerischen Funktionale (d.h. für Abbildungen $C[0,1] \to \mathbb{R}$). Ein Hauptgebiet der numerischen Analysis ist es, effiziente Algorithmen zur Berechnung von bestimmten Funktionalen (Nullstellenfindung, Integration, Maximierung,...) zu finden. Bei Komplexitätsbetrachtungen wird dabei oft implizit ein Modell mit folgenden Eigenschaften angenommen:

1. Der Algorithmus kann nach dem Wert für $f(x)$ für alle x fragen.
2. Der Algorithmus bekommt das exakte Ergebnis $y = f(x)$ durch ein Orakel.
3. Jede solche Anfrage benötigt genau einen Zeitschritt.

Mit diesem Modell lassen sich einfach Komplexitätsschranken finden. Es ist aber kein realistisches Modell, da mit exakten reellen Zahlen gerechnet wird, was mit einer Turingmaschine nicht möglich ist.
Ein realistisches Modell sollte, ähnlich wie das zuvor gesehene Modell zur Berechnung von Funktionen, mit rationalen Approximationen arbeiten. Um ein solches Modell beschreiben zu können, benötigen wir zunächst die folgende Definition:

Definition 2.8. Zwei Funktionen $m : \mathbb{N} \to \mathbb{N}$ und $\Phi : (\mathbb{D} \cap [0,1]) \times \mathbb{N} \to \mathbb{D}$ **repräsentieren** f, wenn

1. Die Funktion m ist eine Modul-Funktion für f auf $[0,1]$.
2. Für alle $d \in \mathbb{D} \cap [0,1]$ und alle $n \in \mathbb{N}$ gilt $|\Phi(d,n) - f(d)| \leq 2^{-n}$.

Beispiel 2.1. Die Funktionen $m : \mathbb{N} \to \mathbb{N}, m(k) = k+1$ und $\Phi(\mathbb{D} \cup [0,1]) \times \mathbb{N} \to \mathbb{D}, \Phi(d,n) = d^2$ repräsentieren die Funktion $f : [0,1] \to \mathbb{R}, f(x) = x^2$

Beweis. Sei $x, y \in [0,1]$ mit $|x-y| \leq 2^{-m(n)} = 2^{-(n+1)}$. Dann ist

$$|f(x) - f(y)| = |x^2 - y^2| = |x+y||x-y| \leq 2 \cdot |x-y| \leq 2^n.$$

Also ist m eine Modulfunktion für f. Weiter ist

$$|\Phi(d,n) - f(d)| = |d^2 - d^2| = 0 \leq 2^{-n}.$$

Also ist auch die zweite Bedingung erfüllt und somit gilt die Behauptung. □

Anmerkung 2.3. Die beiden Funktionen aus 2.8, sind genau die Funktionen aus Satz 2.1. Damit folgt direkt, dass f berechenbar ist, wenn es Funktionen m und Φ gibt, die f repräsentieren.

Mithilfe der letzten Definition können wir nun definieren, wann ein numerisches Funktional berechenbar ist:

Definition 2.9. Ein numerisches Funktional F auf $D \subseteq C[0,1]$ heißt berechenbar, wenn es eine Zwei-Orakel-Turingmaschine M gibt, so dass für jede Funktion $f \in D$, alle Orakel-Funktionen m und Φ, die f repräsentieren, und jede Eingabe $n \in \mathbb{N}$ gilt

$$|M^{m,\Phi}(n) - F(f)| \leq 2^{-n}.$$

Um ein numerisches Funktional zu berechnen, nutzen wir also eine Orakel-Turingmaschine, die die Eingabefunktion $f \in D$ als Orakel bekommt. Das Orakel liefert bei Anfragen für dyadische Zahlen $d \in D$ Approximationen für den Wert von $f(d)$ mit Fehler $\leq 2^{-n}$. Die Maschine verfügt außerdem über ein Orakel, dass den Konvergenzmodul der Funktion f berechnet.

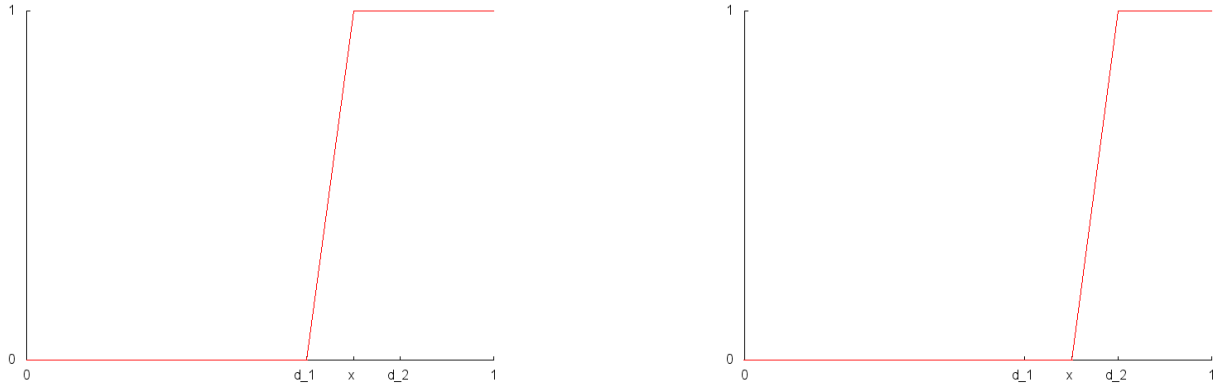

Abbildung 1: Zwei für die Turingmaschine nicht unterscheidbare Funktionen

2.5 Notwendigkeit des Modulorakels

Durch das Modulorakel kann die Maschine bestimmen, wie gut durch die Berechnung von $f(d)$ für eine dyadisch rationale Approximation d an $x \in \mathbb{R}$ der Wert $f(x)$ für $x \in \mathbb{R}$ angenähert wird. Dieses Orakel ist auch wirklich notwendig, um ein sinnvolles Berechnungsmodell zu erhalten. Um dies einzusehen betrachten wir das einfache Funktional

$$F : C[0,1] \to \mathbb{R},\ F(f) = f(\frac{\sqrt{2}}{2}).$$

Das Funktional liefert also den Wert der Eingabefunktion an der (berechenbaren) Stelle $\frac{\sqrt{2}}{2}$. Intuitiv ist klar, dass ein solch einfaches Funktional berechenbar sein sollte, denn es muss ja nur die berechenbare Funktion f an einem berechenbaren Punkt ausgewertet werden. Nach dem oben angegebenen Modell ist das Funktional auch berechenbar. Eine Maschine, die F berechnet, kann folgendermaßen vorgehen:

1. Berechne $k = m(n+1)$
2. Finde $d \in \mathbb{D}$, so dass $|d - \frac{\sqrt{2}}{2}| \leq 2^{-k}$
3. Befrage das Orakel Φ nach $e = \Phi(d, n+1)$
4. Es gilt dann $|F(f) - e| \leq 2^{-n}$

Ohne das Modulorakel kann das Funktional jedoch nicht berechnet werden. Die Maschine kann in endlicher Zeit nur endlich viele Anfragen an das Orakel Φ stellen. Angenommen ein Orakel liefert $\Phi(d,k) = 0$ für alle Anfragen mit $d < \frac{\sqrt{2}}{2}$ und $\Phi(d,k) = 1$ für alle Anfragen mit $d > \frac{\sqrt{2}}{2}$. Es lassen sich dann zwei Funktionen konstruieren, die für die Turingmaschine nicht unterscheidbar sind, deren Wert an der Stelle $\frac{\sqrt{2}}{2}$ aber verschieden ist. Sei dazu d_1 der größte Punkte kleiner $\frac{\sqrt{2}}{2}$ und d_2 der kleinste Punkt größer $\frac{\sqrt{2}}{2}$, nachdem die Maschine das Orakel befragt hat. Wir betrachten nun zwei stückweise linearen Funktionen f_1, f_2 (siehe Abb 1). f_1 ist definiert durch $f_1(0) = 0$, $f_1(d_1) = 0$, $f_1(\frac{\sqrt{2}}{2}) = 1$, $f_1(1) = 1$ und f_2 durch $f_2(0) = 0$, $f_1(\frac{\sqrt{2}}{2}) = 0$, $f_2(d_2) = 1$, $f_1(1) = 1$. Die Maschine müsste also sowohl für f_1 als auch für f_2 den selben Wert liefern. Es gilt jedoch $F(f_1) = 1$ und $F(f_0) = 0$.
Ohne das Modulorakel ist also nichtmal ein so einfaches Funktional, wie das oben angegebene berechenbar und wir kommen zu dem Schluß, dass das Orakel für ein sinnvolles Berechenbarkeitsmodell notwendig ist.

2.6 Komplexität von Funktionalen

Genau wie bei Funktionen können wir nun auch die Komplexiät von Funktionalen definieren. Wir beschränken uns hier auf die Definition eines polynomialzeitberechenbaren Funktionals:

Definition 2.10. Sei $D \subseteq C[0,1]$. Ein Funktional $F : D \to \mathbb{R}$ heißt polynomialzeitberechenbar, wenn es eine Orakelturingmaschine M gibt, so dass für alle m, Φ die ein $f \in \mathbb{D}$ repräsentieren $M^{m,\Phi}(n)$ polynomielle Laufzeit in n hat.

Die obige Definition erweist sich allerdings als zu stark für die meisten praktischen Betrachtungen. Statt der Frage, ob ein Funktional polynomialzeitberechenbar ist, wollen wir uns deshalb im Folgenden mit der Frage befassen, ob das Funktional polynomialzeitberechenbare Funktionen auf polynomialzeitberechenbare reelle Zahlen abbildet.

3 Berechenbarkeit und Komplexität der Maximumsberechnung

Im folgenden Abschnitt geht es um die Berechenbarkeit und Komplexität der Maximumsberechnung, d.h. der Berechnung des Wertes $\max f(x)$ für $x \in [0,1]$. Wir werden sehen, dass dieser Wert immer berechenbar ist, die Berechnung aber schon für polynomialzeitberechenbare Funktionen schwierig sein kann.

3.1 Berechenbarkeit des Maximums

Wir zeigen zuerst, dass das Maximum einer berechenbaren Funktion berechenbar ist, wir zeigen also folgenden Satz:

Satz 3.1. Das Funktional $max : C[0,1] \to \mathbb{R}$, das das Maximum einer berechenbaren Funktion $f : [0,1] \to \mathbb{R}$ in $[0,1]$ berechnet ist berechenbar.

Beweis. Wir geben einen Algorithmus zur Berechnung des Maximums an, der lediglich die beiden Funktion m und Φ benutzt, die f repräsentieren. Damit ist das Maximum nach Definition 2.9 berechenbar. Zunächst wird das Intervall $[0,1]$ in $2^{m(n+1)}$ gleich große Subintervalle

$$[d_i, d_{i+1}] := [i \cdot 2^{-m(n+1)}, (i+1) \cdot 2^{-(m(n+1))}],\ i \in \{0, \ldots, 2^{(m(n+1))} - 1\}$$

unterteilt. Für $x \in [d_i, d_{i+1}]$ gilt dann $|f(x) - f(d_i)| \leq 2^{-(n+1)}$. Das Maximum kann nun berechnet werden, indem die $2^{m(n+1)}$ Approximationen für $f(d_i)$, $i \in \{0, \ldots, 2^{(m(n+1))} - 1\}$ mit Fehler kleiner $2^{-(n+1)}$ berechnet werden (d.h. $\Phi(d, n+1)$) und von diesen der Größte ausgewählt wird. □

Korollar 3.1. Da die Funktionen m und Φ selbst berechenbar sind, ist insbesondere für jede berechenbare Funktion $f : [0,1] \to \mathbb{R}$ der Wert $\max f$ eine berechenbare reelle Zahl.

Anmerkung 3.1. Wir haben gezeigt, dass der Wert des Maximums einer berechenbaren Funktion immer berechenbar ist. Betrachtet man stattdessen das Problem, zu einer gegebenen Funktion den Punkt, an dem das Maximum angenommen wird, zu berechnen, so ist dies im Allgemeinen nicht berechenbar. Es lässt sich sogar zeigen (siehe [Ko91]), dass es eine in polynomieller Zeit berechenbare Funktion $f : [0,1] \to \mathbb{R}$ gibt, die überabzählbar viele Maximumspunkte in $[0,1]$ hat, von denen aber kein einziger berechenbar ist.

3.2 Komplexität der Maximumsberechnung

Der oben angegebene einfache Algorithmus zur Berechnung des Maximums hat offensichtlich auch schon für polynomialzeitberechenbare Funktionen exponentielle Laufzeit. Wir wollen als nächstes untersuchen, ob es einen besseren Algorithmus gibt, der das Maximum effizient berechnen kann. Wir beschränken uns dabei auf die Frage, ob das Maximum einer polynomialzeitberechenbaren Funktion in einem Intervall $[0,x] \subseteq [0,1]$ eine in polynomieller Zeit berechenbare reelle Zahl ist. Im Folgenden werden wir jedoch zeigen, dass dies im Allgemeinen nicht der Fall ist, wenn $P \neq NP$. Genauer zeigen wir den folgenden Satz:

Satz 3.2. Die folgenden Aussagen sind äquivalent:

(a) $P = NP$

(b) Die Funktion $g(x) = max\{f(x,y) \mid 0 \leq y \leq 1\}$ ist in $P_{C[0,1]}$ für alle $f \in P_{C[0,1]^2}$

(c) Die Funktion $h(x) = max\{f(y) \mid 0 \leq y \leq x\}$ ist in $P_{C[0,1]}$ für alle $f \in P_{C[0,1]}$

(d) Die Funktion $k(x) = max\{f(y) \mid 0 \leq y \leq x\}$ ist in $P_{C[0,1]}$ für alle $f \in P_{C^\infty[0,1]}$

Wir zeigen also, dass schon das Problem, das Maximum einer polynomialzeitberechenbaren und unendlich oft differenzierbaren Funktion zu finden, NP-schwer ist.
Wir beweisen zunächst $(a) => (b)$:
Wir können o.B.d.A. annehmen, dass $im(f) \subseteq [0,1]$. Da f polynomialzeitberechenbar ist, gibt es eine Zwei-Orakel-Turingmaschine M, die f in Zeit $p(n)$ für ein Polynom p berechnet. Sei

$$A := \{< d_1, e > \mid (d_1, e) \in [0,1]^2,\ e \in \mathbb{D}_{n+1},\ d_1 \in \mathbb{D}_{p(n+2)} \text{ für ein } n \geq 0 \text{ und für die gilt } \exists d_2 \in \mathbb{D}_{p(n+2)}\ e \leq M^{b_{d_1}, b_{d_2}}(n+2)\}.$$

Berechnet M eine Approximation an den Funktionswert $f(x,y)$ mit Fehler $\leq 2^{-(n+2)}$ kann die Maschine maximal Approximationen der Länge $p(n+2)$ für x und y abfragen, da andernfalls schon die Laufzeit für die Abfrage größer als die Gesamtlaufzeit wäre. A enthält also alle Paare (d_1, e), für die e kleiner als $f(x,y) \pm 2^{-(n+2)}$ für alle y und alle x mit

$|x - d_1| \leq 2^{-p(n+2)}$ ist.
Vergleicht man die Definition von A mit der strukturellen Definition von NP

$$B \in NP \text{ gdw. } B = \{x \in \{0,1\}^* \mid \exists y \in \{0,1\}^{p(x)} < x, y > \in K\} \text{ mit } K \in P,$$

folgt, dass $A \in NP$. Um dies einzusehen, betrachtet man die Paare $< x, y >$ mit $x =< d_1, e >$ und $y = d_2$. Eine (gewöhnliche!) Turingmaschine kann jetzt einfach den Algorithmus von M ausführen um zu entscheiden, ob $e \leq M^{b_{d_1}, b_{d_2}}(n+2)$. d_1 und d_2 werden jetzt aus der Eingabe gelesen, statt von einem Orakel geliefert. Dies ändert aber natürlich nichts an der polynomiellen Laufzeit der Maschine.

Sei nun $d_1 \in \mathbb{D}_{p(n+2)} \cap [0,1], x \in [0,1]$ mit $|d_1 - x| \leq 2^{-p(n+2)}$ und $e = max\{e_1 \in \mathbb{D}_{n+1} \cap [0,1] \mid < d_1, e_1 > \in A\}$.
Wir zeigen jetzt, dass

(i) $|e - g(d_1)| \leq 2^{-(n+1)}$

(ii) $|g(d_1) - g(x)| \leq 2^{-(n+2)}$

Daraus folgt dann $|e - g(x)| \leq 2^{-n}$.

Beweis. Wir beginnen mit dem Beweis von (i):
Nach Definition von e gilt $e \leq f(d_1, d_2) + 2^{-(n+2)}$ für ein $d_2 \in \mathbb{D}_{p(n+2)}$. $g(d_1)$ ist das Maximum von $f(d_1, y)$ für $y \in [0,1]$. Also ist insbesondere $f(d_1, d_2) \leq g(d_1)$. Daraus folgt $e \leq g(d_1) + 2^{-(n+2)} + 2^{-(n+2)} = g(d_1) + 2^{-(n+1)}$.
Andererseits ist $g(d_1) = f(d_1, y_1)$ für ein $y_1 \in [0,1]$. Wähle $d_2 \in \mathbb{D}_{p(n+2)} \cap [0,1]$, so dass $|d_2 - y_1| \leq 2^{-p(n+2)}$. Aus der Definition von A folgt, dass $|f(d_1, d_2) - f(d_1, y_1)| \leq 2^{-(n+2)}$.
Somit ist $e \geq f(d_1, d_2) - 2^{-(n+2)} \geq f(d_1, y_1) - 2^{-(n+1)} = g(d_1) - 2^{-(n+1)}$.
Insgesamt gilt also $|e - g(d_1)| \leq 2^{-(n+1)}$

Nun der Beweis von (ii):
Sei $g(d_1) = f(d_1, y_1)$ und $g(x) = f(x, y_x)$ für entsprechende $y_1, y_x \in [0,1]$. Da $p(n)$ eine Modulfunktion für f ist folgt aus $|d_1 - x| \leq 2^{-p(n+2)}$, dass $|f(d_1, y_x) - f(x, y_x)| \leq 2^{-(n+2)}$. Also gilt $g(d_1) \geq f(d_1, y_x) \geq f(x, y_x) - 2^{-(n+2)} = g(x) - 2^{-n+2}$. Andererseits gilt $g(d_1) - 2^{-(n+2)} = f(d_1, y_1) - 2^{-(n+2)} \leq f(x, y_1) \leq g(x)$.
Insgesamt ist also $|g(d_1) - g(x)| \leq 2^{-(n+2)}$.

□

Da $|e - g(x)| \leq 2^{-n}$, ist $g(x)$ in polynomieller Zeit berechenbar, wenn e in polynomieller Zeit bestimmt werden kann. Es bleibt also zu zeigen, dass es möglich ist, e in polynomieller Zeit zu berechnen. Wir betrachten dazu eine Orakelturingmaschine mit A als Orakel. Diese kann e durch binäre Suche mit polynomiell vielen Anfragen bestimmen. Ist $P = NP$, so ist insbesondere $A \in P$. Also $g \in P^P = P$, d.h. g kann in polynomieller Zeit berechnet werden.

Wir kommen zum Beweisteil (b) => (c).
Das Problem die Funktion $h(x)$ zu berechnen, kann auf die Berechnung von $g(x)$ reduziert werden.
Dazu wird für jedes $f \in P_{C[0,1]}$ eine Funktion $f_1 : [0,1]^2 \to \mathbb{R}$ definiert durch

$$f_1(x,y) = \begin{cases} f(0), & \text{falls } y > x \\ f(x-y), & \text{falls } y \leq x \end{cases}$$

Da f in polynomieller Zeit berechenbar ist, ist auch f_1 in polynomieller Zeit berechenbar. Also gilt $f_1 \in P_{C[0,1]^2}$ und für alle $x \in [0,1]$ ist $g(x) := max\{f_1(x,y) \mid 0 \leq y \leq 1\} = max\{f(y) \mid 0 \leq y \leq 1\} = max\{f(y) \mid 0 \leq y \leq x\} =: h(x)$.

$(c) \implies (d)$ gilt, da für jedes $f \in P_C^\infty[0,1]$ auch $f \in P_{C[0,1]}$ ist.

Beweis $(d) \implies (a)$:
Unser Ziel ist es nun, zu jedem $A \in NP$ eine in polynomieller Zeit berechenbare und unendlich oft differenzierbare Funktion f zu konstruieren, so dass aus $k(x) := max\{f(y) \mid 0 \leq y \leq 1\} \in P_{C[0,1]}$ folgt, dass $A \in P$. Mithilfe der Funktion k soll also zu gegebenen $x \in \{0,1\}^*$ entschieden werden können, ob $x \in A$.
Zur Konstruktion der Funktion f benötigen wir zunächst einige Hilfsfunktionen.

Lemma 3.1. Es gibt eine Funktion $f \in P_{C^\infty[0,1]}$ mit folgenden Eigenschaften:

(1) $f(0) = 0$ und $f(1) = 1$,

(2) $f^{(n)}(0) = f^{(n)}(1) = 0$ für alle $n \geq 1$,

(3) f ist monton wachsend auf $[0, 1]$,

(4) $f^{(n)}$ ist in $P_{C[0,1]}$ für alle $n \geq 1$.

Beweis. Sei

$$h(x) = \begin{cases} e^{-\frac{1}{x^2}}, & \text{falls } x > 0 \\ 0, & \text{falls } x \leq 0. \end{cases}$$

Dann erfüllt die Funktion $f(x)$ mit

$$f(x) = \frac{h(x - \frac{1}{4})}{h(\frac{3}{4} - x) + h(x - \frac{1}{4})}$$

die gewünschten Eigenschaften. □

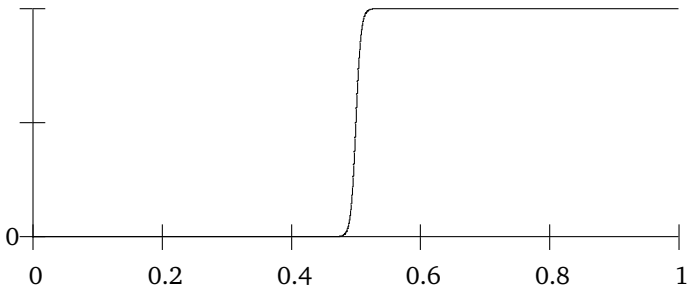

Abbildung 2: Die Funktion f(x)

Mithilfe dieser Funktion lässt sich nun einfach eine unendlich oft differenzierbare und in polynomieller Zeit berechenbare "Hügelfunktion" h_1 konstruieren. Sei dazu

$$h_1 : [0, 1] \to \mathbb{R}, h_1(x) = \begin{cases} g_1(2x), \text{ falls } 0 \leq x \leq \frac{1}{2} \\ g_1(2 - 2x), \text{ falls } \frac{1}{2} \leq x \leq 1 \end{cases}$$

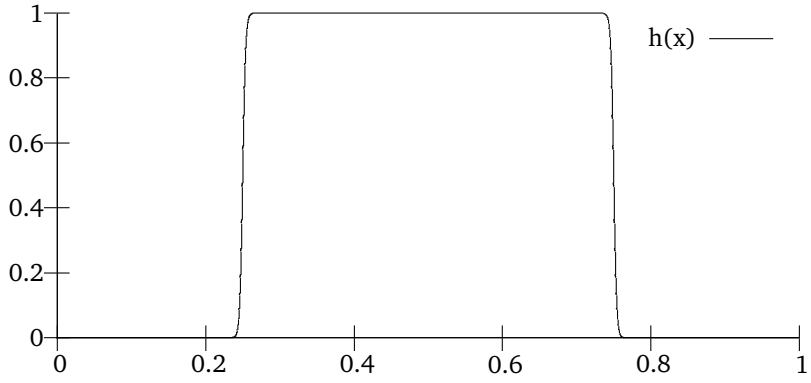

Abbildung 3: Die Hügelfunktion h_1 ist in polynomialzeitberechenbar und unendlich oft differenzierbar.

Wir kommen jetzt zum eigentlichen Beweis:
Ist $A \in NP$, gibt es nach Definition eine Polynomfunktion p, so dass $A = \{x \in \Sigma^* \mid \exists y \in \Sigma^{p(|x|)} : < x, y > \in K\}, K \in P$. Die Idee bei der folgenden Konstruktion ist es, jedes solche Paar $< x, y >$ eindeutig einem Subintervall $I_{x,y} \subseteq [0, 1]$

zuzuordnen. Auf diesen Subintervallen soll dann eine Funktion definiert werden, die in $I_{x,y}$ einen Hügel hat, wenn y ein Zeuge dafür ist, dass $x \in A$. Die Spitze dieses Hügels soll dann das Maximum der Funktion bis zu einem gewissen Punkt $s \in [0,1]$ sein, den man einem $x \in \Sigma^*$ zuordnen kann. Aus der Berechnung von $k(s)$ lässt sich dann entscheiden, ob $x \in A$.
Eine Funktion mit den genannten Eigenschaften werden wir jetzt konstruieren. Wir teilen zunächst das Intervall $[0,1]$ in unendlich viele Subintervalle auf, so dass jedes Intervall eindeutig einem String $s \in \{0,1\}^*$ zugeordnet werden kann. Sei dazu für jedes $n \in \mathbb{N}, n \geq 1$ $a_n = 1 - 2^{-(n-1)}$ und für $s \in \{0,1\}^n$ $u_s = a_n + i \cdot 2^{-2n}$ und $v_s = u_s + 2^{-2n}$ (i ist dabei die zu s gehörige natürliche Zahl, wenn man s als n-bit Binärdarstellung einer Zahl interpretiert). Jedes $s \in \{0,1\}^*$ kann jetzt also dem Intervall $[u_s, v_s]$ zugeordnet werden. Die Intervalle $[u_s, \frac{u_s+v_s}{2}]$ werden nun nochmals in $2^{p(n)}$ gleich große Subintervalle $[y_{s,t}, z_{s,t}]$ mit $y_{s,t} = u_s + i \cdot 2^{-(p(n)+2n+1)}$ und $z_{s,t} = y_{s,t} + 2^{-(p(n)+2n+1)}$ unterteilt. Durch diese Konstruktion ist es möglich jedem Paar $< s,t >$ mit $s \in \{0,1\}^*$ und $t \in \{0,1\}^{p(|s|)}$ eindeutig ein Intervall $[y_{s,t}, z_{s,t}] \subseteq [0,1]$ zuzuordnen.

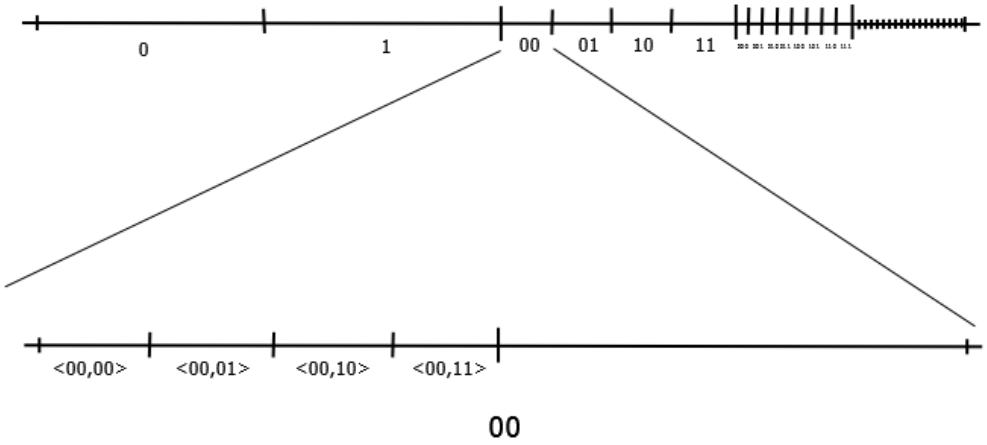

Abbildung 4: Das Intervall $[0,1]$ wird zunächst in unendlich viele Subintervalle $[u_s, v_s]$ geteilt, so dass jedem $s \in \{0,1\}^*$ eindeutig ein Intervall zuegordnet werden kann (oberer Teil der Abbildung). Danach wird jedes Intervall $[u_s, \frac{v_s}{2}]$ nochmal in $2^{p(n)}$ gleich große Subintervalle geteilt. Im unteren Teil der Abbildung ist beispielhaft die Unterteilung für das Intervall $[u_{00}, v_{00}]$ mit $p(2) = 2$ dargestellt.

Die folgende Funktion $f : [0,1] \to \mathbb{R}$ soll, wie oben beschrieben, die Eigenschaft haben, dass anhand des Maximums im Intervall $[0, \frac{u_s+v_s}{2}]$ entschieden werden kann, ob $s \in A$. Sei dazu g_1 die Funktion aus Lemma 3.1 und h_1 die oben definierte Hügelfunktion. Die Funktion f lässt sich dann folgendermaßen definieren:
Für $x \in [\frac{u_s+v_s}{2}, v_s]$ ist

$$f(x) = u_s + g_1\left(\frac{2}{v_s - u_s}x - \frac{v_s + u_s}{v_s - u_s}\right)$$

Für $x \in [y_{s,t}, z_{s,t}]$ ist

$$f(x) = \begin{cases} u_s & \text{, falls } < s,t > \notin K \\ u_s + 2^{-(p(n)+2n+2)} h_1(2^{p(n)+2n+1}(x - y_{s,t})) & \text{, falls } < s,t > \in K \end{cases}$$

Es gilt $f \in P_{C^\infty[0,1]}$, da h_1, $g_1 \in P_{C^\infty[0,1]}$ und $h^{(n)}(0) = g^{(n)}(0) = h^{(n)}(1) = g^{(n)}(1) = 0$ für alle $n \in \mathbb{N}$ (dadurch ist gewährleistet, dass die stückweise definierten Funktionsteile in allen Ableitungen zusammenpassen). f hat einen Hügel der Höhe $2^{-(p(n)+2n+2)}$ mit $n = |s|$ in $[y_{s,t}, z_{s,t}]$, wenn t ein Zertifikat dafür ist, dass $s \in A$. Andernfalls ist f dort flach. Bis auf die Hügel ist f monoton steigend, insbesondere ist immer $f(y) - f(x) \geq 2^{-(p(n)+2n+2)}$ für $x \leq \frac{u_s+v_s}{2}$ und $y > v_s$. Also ist $s \in A$ genau dann, wenn das Maximum von f in $[0, \frac{u_s+v_s}{2}]$ größer ist als u_s. Ist $k \in P_{C[0,1]}$, dann lässt sich also $s \in A$ folgendermaßen in polynomieller Zeit entscheiden:

(i) Berechne Approximation e für $k(\frac{u_s+v_s}{2})$ mit $|e - k(\frac{u_s+v_s}{2})| \leq 2^{-(p(n)+2n+4)}$.

(ii) Prüfe, ob $e > u_s$ (ist $s \in A$ ist $e = u_s + 2^{-(p(n)+2n+4)}$ andernfalls ist $e = u_s$).

Also lässt sich mithilfe der Funktion k für jedes beliebige $x \in \{0,1\}^*$ in polynomieller Zeit entscheiden, ob $x \in A$. Damit insgesamt A in polynomialzeit entschieden werden kann, muss noch gezeigt werden, dass k auch mit einer gewöhnlichen Turingmaschine (ohne Orakel) in polynomieller Zeit berechenbar ist. $\frac{u_s+v_s}{2}$ ist jedoch eine dyadisch rationale Zahl, deren

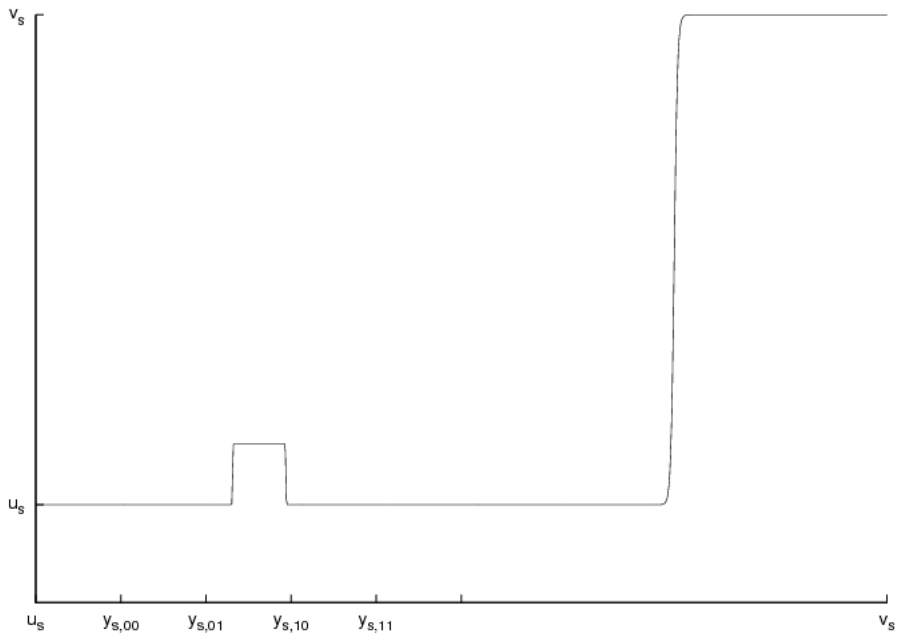

Abbildung 5: Die Funktion hat einen Hügel der Höhe $2^{-(p(n)+2n+2)}$ in $[y_{s,t}, z_{s,t}]$, wenn t ein Zeuge dafür ist, dass $s \in A$. Andernfalls ist f im Intervall $[u_s, \frac{u_s+v_s}{2}]$ konstant gleich u_s. Im Intervall $[\frac{u_s+v_s}{2}, v_s]$ steigt f bis zum Wert v_s. Mithilfe des Maximums von $f(x)$ mit $0 \leq x \leq \frac{u_s+v_s}{2}$ lässt sich entscheiden, ob $s \in A$.

Länge polynomiell in $|s|$ ist. Außerdem kann $\frac{u_s+v_s}{2}$ aus $x \in \Sigma^*$ in polynomieller Zeit berechnet werden. Eine Turingmaschine kann aus $x \in \Sigma^*$ die Binärdarstellung von $\frac{u_s+v_s}{2}$ berechnen und dann den polynomialzeitalgorithmus zur Berechnung von k ausführen. Damit ist $A \in P$.
Ist $P \neq NP$ gibt es eine in polynomieller Zeit berechenbare, unendlich oft stetig differenzierbare Funktion, deren Maximum nicht in polynomieller Zeit berechnet werden kann.

Anmerkung 3.2. Wir haben gesehen, dass das Problem, das Maximum einer polynomialzeitberechenbaren und unendlich oft differenzierbare Funktion so schwierig sein kann, wie ein NP-schweres Problem. Es lässt sich allerdings zeigen, dass dies nicht für analytische Funktionen gilt. Ist $f : [0,1] \to \mathbb{R}$ eine analytische und polynomialzeitberechenbare Funktion, so ist die Funktion $max\{f(x) \mid x \in [0,1]\}$ ebenfalls polynomialzeitberechenbar.

Literatur

[Fri84] H Friedman. The computational complexity of maximization and integration. *Advances in Mathematics*, 53(1):80–98, 1984.

[KF82] Ker-I. Ko and Harvey Friedman. Computational complexity of real functions. *Theoretical Computer Science*, 20(3):323 – 352, 1982.

[Ko91] Ker-I Ko. *Complexity theory of real functions*. Birkhauser Boston Inc., Cambridge, MA, USA, 1991.