Weitere Komplexitätsklassen und Beziehungen zwischen ihnen

Nach Ingo Wegener: „Complexity Theory – Exploring the Limits of Efficient Algorithms“

Im Seminar „Theoretische Informatik und Mathematik“

FH-Bonn-Rhein-Sieg

Jens Mahnke

1 Einleitung

Die vorliegende Ausarbeitung betrachtet das zehnte Kapitel des Buches „Complexity Theory – Exploring the Limits of Efficient Algorithms“ von Ingo Wegener und fasst dieses zusammen. Die Struktur dieses Kapitels lässt sich in drei Schritten zusammenfassen: zuerst werden die grundlegenden Voraussetzungen aus Kapitel 3 kurz wiederholt und auf die in NP bzw. co-NP liegenden Komplexitätsklassen eingegangen. Danach werden allgemein Orakelklassen betrachtet, welche oberhalb von NP liegen können und die so erhaltenen Erkenntnisse in Kapitel 4 zu einer erweiterten polynomiellen Hierarchie zusammengefasst und später noch mit den Komplexitätsklassen BPP und NP in Verbindung gesetzt. Der Hauptteil des Kapitels beschäftigt sich mit der polynomiellen Hierarchie, welche die Klasse NP in verschiedene Unterklassen zwischen P und NPC aufteilt und durch die Orakelklassen nach „oben“ erweitert. Abschließend wird eine kurze Zusammenfassung des behandelten Themas nachgestellt.

2 Fundamentale Betrachtungen

In den vorhergehenden Kapiteln aus [1] wurden Reduktionen erläutert, die Beziehungen zwischen Problemen beschreiben. Mit Hilfe der Reduktionen können Probleme auch zu einer Komplexitätsklasse C in Beziehung gesetzt und so gezeigt werden, das ein Problem C-leicht, C-schwer, C-vollständig oder C-äquivalent ist. Die NP-Vollständigkeit ist unter der Annahme P≠NP eine sehr gute Methode zur Klassifizierung von Problemen, dabei ist nur die fehlende praktische Relevanz der Algorithmen ein Problem. Im Folgenden wird immer davon ausgegangen, dass NP-vollständige Probleme nicht innerhalb von P liegen, wohl aber innerhalb von BPP. Somit hätte der Beweis für BPP = P einen ähnlichen Effekt wie der Beweis von P = NP, hat dann aber nicht das Ansehen in der Komplexitätswelt, da dort NP als die wichtigste Komplexitätsklasse angesehen wird. Auf den Zusammenhang zwischen NP und BPP wird dann in Kapitel 6 näher eingegangen.

3 Komplexitätsklassen innerhalb von NP und coNP

Wenn NP = P ist, dann folgt, da P = coP ist, coNP = P und somit existiert nur noch eine effizient lösbare Problemklasse. Der andere Fall, also P ≠ NP, ist sehr viel interessanter für Untersuchungen. Aus [1] Kapitel 5.1 folgt, dass innerhalb von P drei Äquivalenzklassen existieren, die polynomiell äquivalent sind:

- alle Probleme, die keine Eingabe akzeptieren
- alle Probleme, die alle Eingaben akzeptieren
- alle anderen Probleme

Für NP ≠ P ergibt sich eine vierte Äquivalenzklasse, die der NP-vollständigen Probleme bezüglich der polynomiellen Reduktion. Diese Aussage führt direkt zu der Frage, ob die Klasse NPI = NP – (P ∪ NPC) leer ist, das heißt, das alle Probleme entweder NPC

oder P sind. Die Abkürzung NPI steht dabei für NP-incomplete. Das Problem an dieser Stelle ist nun, dass es unmöglich ist zu beweisen, ob NPI leer ist oder nicht, ohne vorher gezeigt zu haben, dass P ≠ NP ist. Aber es gibt, bzw. gab einige Probleme, bei denen vermutet wird bzw. wurde, dass sie zu NPI gehören, z.B.:

- lineare Programmierung (LP)
- Primzahltests (PRIMES)
- Graphenisomorphie (GI)

Bei den ersten beiden Problemen wurde bereits gezeigt, dass sie innerhalb von P liegen. Bei GI gibt es derzeit keine Hinweise darauf, dass dieses Problem in P liegt. Weiterhin wird vermutet, dass GI nicht in NPC ist. Damit ist GI ein Kandidaten für NPI. Dies wird durch das Existenztheorem von Ladner von 1975, welches hier ohne Beweis zitiert wird untermauert:

Theorem 10.2.1: Wenn P ≠ NP ist, dann ist NPI nicht leer und enthält Probleme, die in Bezug auf $\leq_P$ nicht vergleichbar sind.

Das heißt, dass unter der Voraussetzung P ≠ NP innerhalb von NPI verschiedene $\equiv_P$-Äquivalenzklassen existieren. Mit Hilfe von polynomiellen Reduktionen können nahezu alle Probleme entweder in die Klassen NP oder die Klasse P eingeordnet werden, aber unter der Voraussetzung P ≠ NP muss eine aussagekräftige Struktur von Problemen zwischen NPC und P liegen, eben NPI.

Durch die duale Eigenschaft von NP und coNP sind diese beiden Komplexitätsklassen gleich oder keine der beiden beinhaltet die andere, da aus NP $\subseteq$ coNP folgt, dass coNP $\subseteq$ cocoNP = NP ist. In [1] Kapitel 5.3 wurde NP mit Hilfe eines polynomiell beschränkten Existenzquantors und der Polynomzeit Eigenschaft charakterisiert: Für L $\in$ NP und L' $\in$ P und das Polynom p gilt

$$L = \left\{ x \mid \exists z \in \{0,1\}^{p(|x|)} : (x,z) \in L' \right\}$$

Um nun in dieser Notation ein Problem aus coNP darzustellen, wird formal der Existenzquantor durch den Allquantor ersetzt. Die P ≠ NP These besagt, dass ohne den Existenzquantor weniger Sprachen beschreibbar sind, als mit diesem. Weiterhin kann man davon ausgehen, dass ein Einfaches ersetzen des Existenzquantors durch den Allquantor nicht erlaubt ist, was dann der NP ≠ coNP These entspricht. Dies bedeutet, dass bei Ungleichheit von P und NP mit Hilfe des Existenzquantors mehr Sprachen beschieben werden können als ohne, wo hingegen bei Gleichheit von P und NP es keinen Unterschied macht. Wenn nun gezeigt werden soll, dass NP = coNP ist, reicht es zu zeigen, das NP $\subseteq$ coNP ist, da dann NP im gesamten zu coNP gehört.

Theorem 10.2.2: Wenn L $\in$ NPC und L $\in$coNP folgt NP = coNP.

Das heißt, wenn das härteste Problem aus NP (liegt also in NPC) auch in coNP liegt, dann liegen alle NP Probleme auch in coNP. Der Beweis wird über die Konstruktion einer Polynomzeit beschränkten nicht-deterministischen Turingmaschine für L' $\in$ NP und L' $\in$ coNP, also auch $\overline{L}$ $\in$ NP. Die konstruierte Turingmaschine akzeptiert Eingaben w $\in$ $\overline{L}'$ und weist Eingaben w $\notin$ $\overline{L}'$ ab. Da L $\in$ NPC und L' $\in$ NP ist, folgt, das L' polynomiell reduzierbar auf L ist gilt:

$$w \notin L' \Leftrightarrow f(w) \notin L \text{ und somit}$$

$$w \in \overline{L}' \Leftrightarrow f(w) \in \overline{L}$$

Da L ∈ coNP kann nun in Polynomzeit überprüft werden ob $f(w) \in \overline{L}$ und erhalten auch den nichtdeterministischen Polynomzeittest für $w \in \overline{L}$'.

NP ≠ coNP impliziert weiterhin, dass eine Sprache L ∈ NP ∩ coNP weder NP-vollständig noch coNP-vollständig ist. Für GI weiß man, dass GI ∈ NP ist, aber nicht ob GI ∈ oder ∉ coNP ist, es hilft also nicht bei der Entscheidung ob GI ∈ NPC ist oder nicht. Weiterhin impliziert NP ≠ coNP das NP-vollständige Probleme nicht zu coNP gehören können, weshalb es auch nicht möglich ist eines der bekannten NP-vollständigen Probleme mit Allquantoren zu repräsentieren und damit zu zeigen, dass es zu coNP gehört.

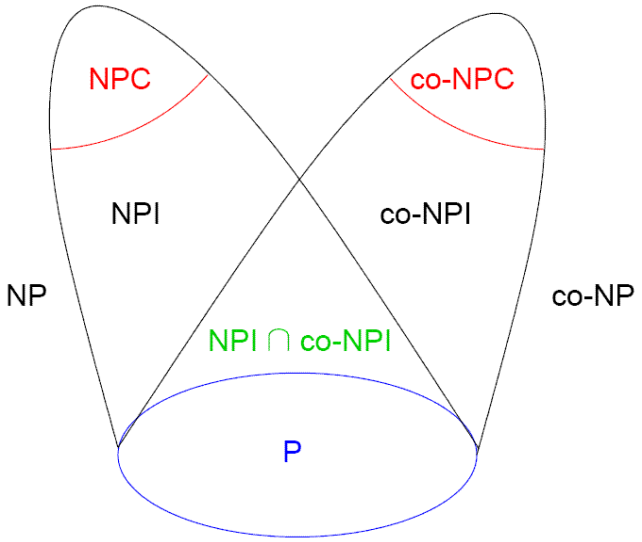

Abbildung 1 Komplexitätsklassen NP∪coNP mit NP∩coNP≠P aus [1]

Zusammenfassend ist der Zusammenhang zwischen NP und coNP in der Abbildung 1 dargestellt.

4 Orakelklassen

Das erste Mal wurden Orakel bereits bei der Einführung der Reduktionen in [1] Kapitel 4.1 betrachtet. Sie beschreiben einen unbekannten Unterprogrammaufruf, wobei die Kosten für diesen Aufruf mit 1 berechnet werden. Die Orakelklassen stehen für die „Welt“ oberhalb von NP ∪ coNP. Diese Orakelklassen werden in der folgenden Definition definiert:

Definition 10.3.1: Die Komplexitätsklasse P(L) definiert für das Entscheidungsproblem L, welches alle Entscheidungsprobleme L' mit L' $\leq_T$ L enthält, d.h. alle Probleme L', die mit Hilfe eines Orakels für L in Polynomzeit mit einem Algorithmus entschieden werden können. Die Komplexitätsklasse P(C) für eine Klasse C von Entscheidungsproblemen ist die Vereinigung aller P(L) mit L ∈ C.

Wenn nun in C ein $\leq_P$- oder $\leq_T$-vollständiges Problem L* liegt, folgt P(C) = P(L*), da jede Anfrage an das Orakel L ∈ C durch einen Aufruf des Polynomzeit Algorithmus welches das Orakel L* befragt ersetzt werden, insbesondere gilt P(NP) = P(SAT). P(NP) ist also eine strenge Oberklasse von NP, insbesondere ist coNP ⊆ P(NP), da die Antwort des Orakels einfach umgekehrt werden kann, daher ist P(NP) auch eine Oberklasse von NP ∪ coNP. Leider kann nichts davon bewiesen werden, denn falls P = NP ist, folgt P(NP) = P(P) = P. Da bisher immer P und auch NP betrachtet wurden, wird nun NP(L) und NP(C) definiert:

Definition 10.3.2: Die Komplexitätsklasse NP(L) definiert für das Entscheidungsproblem L, welches alle Entscheidungsprobleme L' enthält, für die ein nichtdeterministischer polynomzeit Algorithmus mit Orakel L existiert. Die Komplexitätsklasse NP(C) für eine Klasse C von Entscheidungsproblemen ist die Vereinigung aller NP(L) mit L ∈ C.

Es gilt also P(P) = P und NP(P) = NP, da jede Anfrage an das Orakel durch (deterministische) Polynomzeit Algorithmen ersetzt werden kann.

Als beispielhaften Beweis kann das Minimal-Circuit-Problem (MC) herangezogen werden, dabei sei S ein

Schaltkreis, für den gilt, es gibt keine kleineren Schaltkreis, der die von S berechnete Funktion auch berechnet.

Theorem 10.3.3: *MC $\in$ coNP(NP), und vermutlich MC $\notin$ NP oder NP(NP)*

Beweis: Sei MC = {S Schaltkreis | $\forall$ Schaltkreise S' mit size(S') < size(S) $\exists$ Eingabe a: S(a) ≠ S'(a)}, dabei sind die size(S) und S(x) in polynomieller Zeit überprüfbar.

Zu Zeigen: coMC $\in$ NP(NP)

coMC = {S Schaltkreis | $\exists$ Schaltkreise S' mit size(S') < size(S) $\forall$ Eingabe a: S(a) ≠ S'(a)}

L := {(S, S'), size(S') < size(S) | $\exists$ a: S'(a) ≠ S(a)}

=> L $\in$ NP

Es wird also ein NP(L) Algorithmus für coMC mit der Eingabe S benötigt:

1. Wähle nichtdeterministisch S' mit size(S') < size(S).
2. Teste mit Unterprogramm, ob (S, S') $\in$ L.
3. Negiere die Antwort des Orakels.

In der Literatur findet sich auch eine andere Notation für denselben Zusammenhang: dort wird P(C) auch als P^C und NP(C) als NP^C notiert.

5 Polynomielle Hierarchie

In den vorhergehenden Kapiteln wurde die Grundlage gelegt um eine Vielzahl an Komplexitätsklassen zu definieren und diese Klassen in einen Zusammenhang zu bringen. Dazu wird an dieser Stelle eine Notation dieser Klassen eingeführt:

Definition 10.4.1: Sei Σ_1:= NP, Π_1 := coNP und Δ_1 := P. Für k ≥ 1, sei

- *Σ_{k+1} := NP(Σ_k),*
- *Π_{k+1} := coΣ_{k+1} = coNP(Σ_k) und*
- *Δ_{k+1} := P(Σ_k)*

Die polynomielle Hierarchie (PH) ist die Vereinigung aller Σ_k für k ≥ 1. Für k = 0 sei $\Sigma_0 = \Pi_0 = \Delta_0 = P$.

Zum Beispiel ist Σ_3=NP(NP(NP)), Π_3=coNP(NP(NP)) und Δ_3=P(NP(NP)). Mit Hilfe der Definition 10.4.1 gilt dann mit Theorem 10.3.3, das MC $\in \Pi_2$ ist. In der folgenden Abbildung ist der Zusammenhang der eben definierten Komplexitätsklassen illustriert.

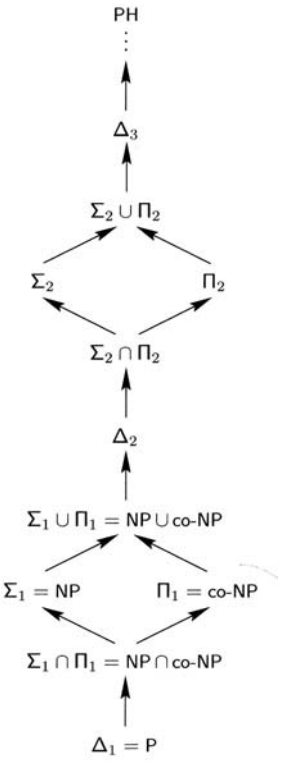

Abbildung 2 Komplexitäts-Landkarte der polynomiellen Hierarchie

Die Pfeile auf der Abbildung stehen jeweils für die Inklusion, dabei wird eine echte Hierarchie vermutet. In den folgenden vier Lemma werden vier beweisbare Aussagen zum Zusammenhang der Klassen gemacht und bewiesen. Darauf folgt die Nennung von theoretischen Hypothesen zu diesen Klassen, welche nicht beweisbar sind.

5.1 Lemma zur Polynomiellen Hierarchie

Lemma 10.4.2.1:

$$\Delta_k = co\Delta_k = P(\Delta_k) \subseteq \Sigma_k \cap \Pi_k \subseteq \Sigma_k \cup \Pi_k \subseteq \Delta_{k+1} = P(\Pi_k)$$

Beweis: Das erste Gleichheitszeichen (a) steht für P = coP, was durch negieren der Antwort erreicht wird. Weiterhin ist (b)

$\Delta_k \subseteq \mathrm{K}_k \cap \Pi_k$

$P(\Delta_k) = P(P(\Sigma_{k-1})) = P(\Sigma_{k-1}) = \Delta_k$, da ein polynomieller Algorithmus der einen polynomzeit Algorithmus mit Orakel L ∈ Σk+1 befragt, nichts anderes als ein polynomieller Algorithmus mit Orakel L ∈ Σk+1. Der nächste Schritt ist (c) $\Delta_k \subseteq \mathrm{K}_k \cap \Pi_k$, was über zwei Gleichungen gezeigt werden kann:

- $\Delta_k = P(\Sigma_{k-1}) \subseteq NP(\Sigma_{k-1}) = \Sigma_k$
- $\Delta_k = P(\Sigma_{k-1}) \subseteq coNP(\Sigma_{k-1}) = \Pi_k$

Der folgende Schritt (d) $\Sigma_k \cap \Pi_k \subseteq \Sigma_k \cup \Pi_k$ ist trivial.

Der nächste Beweisschritt (e) $\Sigma_k \cup \Pi_k \subseteq \Delta_{k+1}$ kann wieder über zwei Gleichungen gezeigt werden:

- $\Sigma_k \subseteq P(\Sigma_k) = \Delta_{k+1}$
- $\Pi_k = co\Sigma_k \subseteq co\Delta_{k+1} = \Delta_{k+1}$

Im letzten Beweisschritt (f) muss noch gezeigt werden, dass $\Delta_k = P(\Pi_k)$ gilt. Nach Definition gilt $\Delta_k = P(\Sigma_k)$, was einem Orakel L ∈ Σ_k entspricht. Dies ist wiederum gleich $P(\Pi_k)$, was einem Orakel L ∈ coΣ_k entspricht und durch einfaches negieren der Antworten entsteht.

Lemma 10.4.2.2: $\Delta_{k+1} = NP(\Pi_k) = NP(\Delta_{k+1})$

Beweis: Das erste Gleichheitszeichen ist analog zu 5.1 (f). Für den nächsten Schritt gilt „$\subseteq$ “, da $\Pi_k \subseteq \Delta_{k+1}$ gilt. Für „$\supseteq$ “ gilt, das das Orakel aus $\Delta_{k+1} = P(\Sigma_k)$ durch ein Orakel aus Σ_k ersetzbar ist, indem die polynomielle Rechnung ausgeführt wird und nur L ∈ Σ_k als Orakel benutzt wird.

Lemma 10.4.2.3: $\Pi_{k+1} = coNP(\Pi_k) = coNP(\Delta_{k+1})$

Beweis: Diese Gleichung entsteht, wenn man „co“ auf die Gleichung aus Lemma 10.4.2.2 anwendet und nach der Definition auflöst.

Lemma 10.4.2.4: $\Sigma_k \subseteq \Pi_k \Rightarrow \Sigma_k = \Pi_k$

Beweis: Wenn gilt $\Sigma_k \subseteq \Pi_k \Rightarrow \underbrace{co\Sigma_k}_{\Pi_k} \subseteq \underbrace{co\Pi_k}_{\Sigma_k}$, womit gezeigt wurde, dass $\Pi_k \subseteq \Sigma_k$ gilt und somit $\Sigma_k = \Pi_k$ gilt.

Aus diesen Aussagen kann nun die in Abbildung 2 dargestellte Komplexitätslandschaft hergeleitet werden. Für die Pfeile wurde gezeigt, dass es sich um Teilmengen oder gleich Beziehungen handelt, aber es wird vermutet, dass es sich bei der Beziehung nur um echte Teilmengen handelt. Weiterhin wird vermutet, dass die Klassen Σ_k und Π_k nicht vergleichbar bezüglich der Mengeninklusion sind, dies wird im folgenden Kapitel als Hypothesen formalisiert.

5.2 Komplexitätstheoretische Hypothesen zur PH

Es gilt $\forall k$:

- $\Sigma_k \neq \Sigma_{k+1}$
- $\Pi_k \neq \Pi_{k+1}$
- $\Sigma_k \neq \Pi_k$

- $\Delta_k \neq \Sigma_k \cap \Pi_k \neq \Sigma_k \neq \Sigma_k \cup \Pi_k \neq \Delta_{k+1}$

Diese Hypothesen sind nicht beweisbar, geben aber einen genaueren Blick auf die in Abbildung 2 dargestellte polynomielle Hierarchie und beschreiben die echte Teilmengenbeziehung vertikal und horizontal. Somit sind diese Klassen nicht vergleichbar. Bisher wurde aus drei Sichten die Klasse NP betrachtet:

- nichtdeterministische Algorithmen
- randomisierter Algorithmus aus RP
- logikorientierte Sicht mit Existenzquantor und Prädikat

Im folgenden Kapitel wird die logikorientierte Sicht auf die Polynomielle Hierarchie betrachtet.

5.3 Logikorientierte Sicht auf die PH

Die Klasse Σ_k wurde bisher nur durch nichtdeterministische Orakelalgorithmen betrachtet. Das folgende Theorem definiert die Logikorientierte Sicht auf die Klasse Σ_k.

Theorem 10.4.3: Ein Entscheidungsproblem L gehört zur Klasse Σ_k wenn ein Polynom p und ein Entscheidungsproblem L' $\in$ P existiert, so dass für $A=\{0,1\}^{p(|x|)}$ gilt:

$$L = \{x \mid \exists y_1 \in A \forall y_2 \in A \exists y_3 \in A ... Q y_k \in A : (x, y_1, y_2, y_3, ..., y_k) \in L'\}$$

Wobei Q für einen Quantor (All- oder Existenz-) steht, so dass eine alternierende Folge von Quantoren entsteht. Bei geradem k entspricht das Q einem Allquantor und bei ungeradem k einem Existenzquantor.

Der Beweis des Theorems erfolgt über die vollständige Induktion über k. Für k=1 ist Σ_k=NP, was dann der logikorientierten Sicht auf NP entspricht. Für k ≥ 2 kann ein nichtdeterministische Algorithmus ein y_1 aus A generieren. Unter der Voraussetzung, dass L aus P ist kann der Algorithmus auf das folgende Orakel zugreifen

$$L^* = \{(x, y_1) \mid \exists y_2 \in A \forall y_3 \in A ... Q y_k \in A : (x, y_1, y_2, y_3, ..., y_k) \in \bar{L}'\}$$

und befragt es nach (x, y_1) und negiert dessen Antwort.

Wenn x aus L ist, existieren einige y_1 für die

$$\forall y_2 \in A \exists y_3 \in A ... Q y_k \in A : (x, y_1, y_2, y_3, ..., y_k) \in L'$$

wahr wird und somit (x, y_1) nicht aus L* ist, was zur Folge hat, das der nichtdeterministische Algorithmus x akzeptiert.

Ist x nicht aus L wird für alle y_1 aus A

$$\exists y_2 \in A \forall y_3 \in A ... Q y_k \in A : (x, y_1, y_2, y_3, ..., y_k) \in L'$$

falsch, weshalb der Algorithmus x abweist. Da y_1 polynomielle Länge hat, ist die Laufzeit polynomiell beschränkt.

Wenn L aus Σ_k ist, besitzt L einen nichtdeterministischen polynomzeit Algorithmus A_L mit Orakel L' aus Σ_{k-1} und einem Entscheidungsproblem B:

$$L' = \{z \mid \exists y_2 \in A \forall y_3 \in A ... Q y_k \in A : (z, y_1, y_2, y_3, ..., y_k) \in B\}$$

Es existieren zwei Polynome q und r, so dass A_L bei Eingabe x das Orakel L' maximal q(|x|) mal mit der Maximallänge r(|x|) befragt. Nun müssen A_L und L' so angepasst werden, dass auf jedem Pfad für alle z der Länge |x| genau q(|x|) Anfragen mit der Länge r(|x|) an das Orakel gemacht werden. Wenn x aus L ist, existiert ein Pfad w mit den zugehörigen Orakel Anfragen b und den Antworten a:

$$\exists w, b_1, ..., b_{q(|x|)}, a_1, ..., a_{q(|x|)}$$

Sei C* die Menge aller ($x, w, b_1, ..., b_{q(|x|)}, a_1, ..., a_{q(|x|)}$), so dass die i-te Anfrage von A_L bei Eingabe x an Pfad w b_i ist und $b_1,\ldots,b_{i-1}$ die vorherigen Anfragen und $a_1,\ldots,a_{i-1}$ die bisherigen Antworten des Orakel sind

und die Eingabe x akzeptiert wird. C* ist also aus P. Bleibt noch zu zeigen, dass auf die Anfrage b_i die korrekte Antwort a_i gegeben wird. Dabei treten zwei Fälle auf:

1. $a_i = 1$

Mit Hilfe der Formel:

$$\exists y_2^i \in A \forall y_3^i \in A ... Q y_k^i \in A : (b_i, y_2^i, ..., y_k^i) \in B$$

kann die Antwort a_i auf Korrektheit geprüft werden.

2. $a_i = 0$

In diesem Fall kann die Formel:

$$\forall y_2^i \in A \exists y_3^i \in A ... Q y_k^i \in A : (b_i, y_2^i, ..., y_k^i) \in \overline{B}$$

zur Überprüfung genutzt werden.

Da der Wert von a_i aber nicht im Voraus bekannt ist, müssen die Formeln zu einer einzigen Formel kombiniert werden:

$$\exists y_1^i \in A \forall y_2^i \in A \exists y_3^i \in A ... Q y_k^i \in A : (b_i, a_i, y_1^i, ..., y_k^i) \in B^*$$

um in einem Algorithmus anwendbar zu sein. Hierbei steht B* für alle Vektoren mit $a_i = 1$ und $(b_i, y_1^i, ..., y_{k-1}^i) \in B$, und alle Vektoren mit $a_i = 0$ und $(b_i, y_2^i, ..., y_k^i) \in \overline{B}$. Somit enthält das Entscheidungsproblem B alle

$$(x, w, b_1, a_1, y_1^1, ..., y_k^1, ..., b_{q(|x|)}, a_{q(|x|)}, y_1^{q(|x|)}, ..., y_k^{q(|x|)})$$

mit $(x, w, b_1, ..., b_{q(|x|)}, a_1, ..., a_{q(|x|)}) \in C^*$

und $(b_i, a_i, y_1^i, ..., y_k^i) \in B^*$ für $1 \leq i \leq q(|x|)$. Für die praktische Sicht muss noch dafür gesorgt werden, dass hinter jedem Quantor ein boolescher Vektor der Polynomlänge p'(|x|) steht.

Durch die Anwendung von DeMorgans Gesetzen erhält man aus dem soeben bewiesenen Theorem das folgende Korollar:

Korollar 10.4.4: Ein Entscheidungsproblem L ist in Π_k, wenn und nur wenn ein Polynom p und ein Entscheidungsproblem L' aus P existiert, so dass A = $\{0,1\}^{p(|x|)}$ gesetzt werden kann und es gilt:

$$L = \{x \mid \forall y_1 \in A \exists y_2 \in A, ..., Q y_k \in A : (x, y_1, ..., y_k) \in L'\}$$

Bei ungeradem k entspricht das Q einem Allquantor und bei geradem k einem Existenzquantor.

Somit enthalten die Klassen Σ_k und Π_k die Probleme mit den folgenden Eigenschaften: k-1 alternierende Quantoren, polynomiell viele Variablen und in polynomzeit entschieden werden können. Der Unterschied liegt beim ersten Quantor. Die Hypothese dass alle diese Klassen verschieden sind basiert auf der Hypothese, dass jeder neue Quantor die Beschreibungsstärke solcher Formeln erhöht und das es wichtig ist, welcher Quantor als erstes auftritt.

Theorem 10.4.5: Wenn $\Sigma_k = \Pi_k$, dann ist $PH = \Sigma_k$.

Das heißt, dass unter der Annahme $\Sigma_k = \Pi_k$ die in Abbildung 2 dargestellten Komplexitäts-Landkarte oberhalb von $\Sigma_k \cap \Pi_k$ kollabiert und alle Oberklassen gleich $\Sigma_k \cap \Pi_k$ sind. Somit ist die Annahme $\Sigma_{k+1} \neq \Sigma_k$ aus Komplexitätstheoretischer Sicht eine strengere Annahme als $\Sigma_k \neq \Sigma_{k-1}$. Die $NP \neq P$ Hypothese ist dabei die schwächste aller Annahmen und es folgt für NP=P, dass $\Sigma_1 = \Pi_1$ und PH=P.

Beweis: Zu zeigen ist das aus $\Sigma_k = \Pi_k$ folgt, dass $\Sigma_{k+1} = \Pi_{k+1} = \Sigma_k$ ist. Der Beweis erfolgt über vollständige Induktion über k. Hier wird der Bweis Beispielhaft für k = 4 dargestellt, also z.z. $\Sigma_4 = \Pi_4$.

$$\exists \forall \exists \forall P = \forall \exists \forall \exists P$$

Die Obenstehende Formel beschreibt ein Entscheidungsproblem P aus P mit polynomiell vielen Variabeln. Wenn man nun Σ_5 betrachtet, wird das Problem als $\exists(\forall\exists\forall\exists P)$ beschrieben und zu $\exists\exists\forall\exists\forall P$ aufgelöst. Die beiden führenden Quantoren können, da diese gleich sind, zu einem Quantor zusammengefasst werden. Somit kann jedes Σ_5 Problem als $\exists\forall\exists\forall P$ geschrieben werden und gehört somit zu Σ_4. Daraus folgt analog $\Sigma_5 = \Sigma_4 = \Pi_4$ und $\Pi_5 = \Pi_4 = \Sigma_4$.

Korollar 10.4.6: Wenn $\Sigma_k = \Sigma_{k+1}$ ist, dann gilt $PH = \Sigma_k$.

Dieses Korollar betrachtet die erste Hypothese aus Kapitel 5.2 genauer und die Folgen für die polynomielle Hierarchie, welche dann zusammenfällt.

Beweis: Es gilt $\Sigma_k \subseteq \Pi_{k+1}$ und aus $\Sigma_k = \Sigma_{k+1}$ folgt, dass $\Sigma_{k+1} \subseteq \Pi_{k+1}$ gilt und mit Lemma 10.4.2 gilt $\Sigma_{k+1} = \Pi_{k+1}$, wegen dem Zusammenhang der Klassen. Somit folgt mit dem vorherigen Theorem 10.4.5 ($PH = \Sigma_{k+1}$) das $PH = \Sigma_k$ gilt.

Im Folgenden werden die theoretischen Ergebnisse auf die Praxis bzw. einem konkreten Problem angewendet. Mit Hilfe des Theorem 10.4.3 kann ein kanonischer Generator für Erfüllbarkeitsprobleme der Größe k, vergleichbar mit SAT_{CIR} erstellt werden. Das heißt, die Probleme beschäftigen sich mit Schaltkreisen C die mit k Variablenvektoren $(x_1,\ldots,x_k)$ der Länge n für $A=\{0,1\}^n$ belegt werden. So erhält man:

$$\exists x_1 \in A \forall x_2 \in A ... Q x_k \in A : C(x) = 1$$

Dabei gibt C(x) den Wert des Schaltkreises für die Eingabe x an. Da die Überprüfung von "C(x) = 1" in Polynomzeit erfolgt, folgt mit Theorem 10.4.3, dass $SAT_{CIR}^k \in \Sigma_k$ ist. Im Gegensatz zu MC, welches in $\Pi_2 - \Pi_1$ vermutet wird, wird SAT_{CIR}^k in $\Sigma_k - \Sigma_{k-1}$ vermutet. Aber für sehr große k werden wohl keine praktisch relevanten Probleme in $\Sigma_k - \Sigma_{k-1}$ liegen. Trotzdem kann die Vermutung, das Probleme in $\Sigma_k - \Sigma_{k-1}$ liegen untersucht werden, indem man die Idee der NP-Vollständigkeit auf die $\Sigma_k - Vollständigkeit$ erweitert. Somit gilt für $\Sigma_k - vollständige$ Probleme L, das entweder $\Sigma_k = \Sigma_{k-1}$ oder $L \notin \Sigma_{k-1}$ gilt. Da davon ausgegangen wird, dass $\Sigma_k \neq \Sigma_{k-1}$ gilt, erhält man eine starke Anzeichen für $L \notin \Sigma_{k-1}$. Mit Hilfe der Methoden aus dem Beweis des Theorems von Cook erhält man das folgende Theorem:

Theorem 10.4.7: SAT_{CIR}^k ist $\Sigma_k - vollständig$.

Die ist gleichbedeutend mit $SAT_{CIR}^k \in \Sigma_{k-1} \Leftrightarrow \Sigma_{k-1} = \Sigma_k$ und zusammen mit NP=P => PH=P wird der Glaube an $P \neq NP$ weiter bestärkt. Somit existieren auf jeder Ebene der polynomiellen Hierarchie PH vollständige Probleme die die Klassen Σ_k und Σ_{k-1} voneinander trennen.

Zum Abschluss dieses Kapitels noch ein kleines „Gedankenspiel": Gilt NP(L) ≠ P(L)?

Für L ∈ P ist diese Frage gleichbedeutend mit NP ≠ P, also unmöglich zu beantworten. Aber selbst dieser Antwort kann positives entnommen werden, denn die Frage ob $\forall L : NP(L) \neq P(L)$ oder $\forall L : NP(L) = P(L)$ gilt wurde durch Diagonalisierungsbeweise negativ beantwortet. Es existieren Sprachen L_1 und L_2, so dass gilt:

$\exists L_1 : NP(L_1) \neq P(L_1)$ und $\exists L_2 : NP(L_2) = P(L_2)$.

Dies hat praktische Bedeutung für die Falsifizierung

von Beweisen für NP ≠ P, wenn dieser NP(L) ≠ P(L) impliziert, muss er falsch sein. Die Erkenntnis von Restriktionen bei der Beweisführung bei NP ≠ P schränkt die Möglichkeiten von vorneherein ein und schließt somit Wege die nicht zum Ziel führen können aus.

6 BPP, NP und die Polynomielle Hierarchie

Dieses Kapitel beschäftigt sich mit der Frage in welchem Zusammenhang die beiden Klassen BPP und NP stehen. NP und BPP basieren auf randomisierten Algorithmen, wobei NP auf einseitigen Fehlern mit hoher Fehlerwahrscheinlichkeit und BPP auf beidseitigen Fehlern mit beschränkter Fehlerwahrscheinlichkeit basieren. Dies legt die Vermutung nahe, dass die Klassen NP und BPP nicht vergleichbar bezüglich der Teilmengenbildung sind. Da aber BPP als „nicht viel größer“ als P angesehen wird, liegt eine Teilmengenbeziehung zwischen NP und BPP wieder nahe. Das bisher beste, beweisbare Resultat beim Vergleich von NP und BPP ist im folgenden Theorem dargestellt.

Theorem 10.5.1: $BPP \subseteq \Sigma_2 \cap \Pi_2$

Beweis: Es genügt zu zeigen, dass $BPP \subseteq \Sigma_2$ gilt, da dann folgt, dass $BPP = coBPP \subseteq co\Sigma_2 = \Pi_2$.

Sei L ∈ BPP, also mit Fehlerwahrscheinlichkeit kleiner 1/3, welche durch „probability amplification“ verbessert werden kann und für eine Eingabe der Länge n bei $2^{-(n+1)}$ liegt. Die Rechenzeit ist polynomiell beschränkt und alle Rechenwege haben genau die Länge p(n), wobei p(n) durch n ohne Rest teilbar ist. Für $n \geq n_0$ ist $p(n) \leq 2^n$. Es existieren bei Eingabe von x also akzeptierende (A(x)) und nicht akzeptierende (N(x)) Rechenwege, wobei für x ∈ L die Anzahl von N(x) sehr klein wird und für x ∉ L wird die Anzahl von A(x).

Sei B die Sprache aller Tripel (x, r, z) aus der Eingabe $x \in \{0,1\}^n$, k Berechnungspfade $(r_1,\ldots,r_k) \in \{0,1\}^p$ und einer Pfadtransformation $z \in \{0,1\}^p$, bei der $r_i \oplus z$ für mindestens ein i einen akzeptierenden Pfad enthält. Da in deterministischer polynomieller Zeit ein randomisierter Algorithmus mit polynomieller Laufzeitschranke auf polynomiell vielen Pfaden simuliert werden kann, gilt B ∈ P, falls k(n) auch polynomiell beschränkt ist.

Mit Hilfe von B wird nun die Sprache L konstruiert:

$$L = \{x \mid \exists r = (r_1,...,r_k) \in \{0,1\}^{p*k} \forall z \in \{0,1\}^p : (x,r,z) \in B\}$$

Da nicht unbedingt alle x ∈ L eine allgemeinen akzeptierenden Pfad besitzen, kann man durch die Auswahl genügend vieler Pfade $(r_1,\ldots,r_k)$ davon ausgehen, das die Transformation mindestens einen akzeptierenden Pfad für x ∈ L erzeugt. Für den umgekehrten Fall x ∉ L existieren so wenige akzeptierende Pfade, dass nach der Transformation sehr wahrscheinlich x abgewiesen wird. Für k = p/n kann dies gezeigt werden:

Für x ∈ L sei R(x) die Menge der schlechten r, dass heißt, die Pfade auf denen x abgewiesen wird. Zu zeigen ist nun, dass $|R(x)| < 2^{p*k}$, womit gezeigt ist, dass für die guten r-Vektoren die Gleichung gilt. Sei $w_i = r_i \oplus z \Leftrightarrow w_i \oplus z = r_i$ und R(x) die Menge aller $(w_1 \oplus z,...,w_k \oplus z)$, so dass $z \in \{0,1\}^p$ und $w_i \in$ N(x) für alle i sind. Es gilt also:

$$|R(x)| \leq |N(x)|^k * 2^p$$

und da x ∈ L ist folgt

$$|N(x)| \leq 2^{p-(n+1)}.$$

Mit k = p/n:

$$\Rightarrow |R(x)| \leq 2^{(p-(n+1))*k} * 2^p$$
$$= 2^{p*k+p-n*k-k} = 2^{p*k-k}$$
$$\leq \frac{1}{2} * 2^{p*k}$$

Das heißt, dass mindestens die Hälfte der r-Vektoren gute Vektoren sind.

Für x $\notin$ L ist N(x) so groß, dass so gut wie alle r-Vektoren ($r_1,\ldots,r_k$) schlecht sind und zu zeigen ist, dass $\exists z \in \{0,1\}^p : (x,r,z) \notin B$, dass heißt $\forall i : r_i \oplus z \in N(x)$. Sei

$$Z_i(r) = \{z | r_i \oplus z \in N(x)\} = \{z \oplus r_i | z \oplus r \in N(x)\}$$

$\Rightarrow |Z_i(r)| = |N(x)| = |\{0,1\}^p - A(x)| \geq 2^p - 2^{p-(n+1)}$, da $|A(x)| \geq 2^{p-(n+1)}$ wegen der Fehlerwahrscheinlichkeit ist. Dies bedeutet, dass maximal $2^{p-(n+1)}$ z-Vektoren nicht in $Z_i(r)$ enthalten sind. Daraus folgt, dass maximal $k*2^{p-(n+1)}$ z-Vektoren nicht in mindestens einem $Z_j(r)$ enthalten sind. Wenn nun Werte von $k \leq 2^n$ betrachtet werden $k*2^{p-(n+1)} \leq 1/2 * 2^p$ und mindestens ein z-Vektor gehört zu $Z_i(r)$, mit $r_i \oplus z \in N(x) \quad \forall i$ und somit $(x,r,z) \notin B$. Daher besitzt L die Obenstehende Eigenschaft und es gilt $L \in \Sigma_2$.

Es wurde gezeigt, das $L \in \Sigma_2 = NP(NP)$ gilt und das NP-Orakel „$\exists z \in \{0,1\}^p : (x,r,z) \notin B$“ wurde dazu genutzt. Die Vektoren von r werden nichtdeterministische vom Algorithmus zufällig erzeugt und ergeben für x $\notin$ L eine Fehlerwahrscheinlichkeit von 0, für x $\in$ L ist die Fehlerwahrscheinlichkeit durch ½ beschränkt. Das heißt, dass der äußere Algorithmus ein RP Algorithmus ist und L zur Klasse RP(NP) gehört.

Definition 10.5.2: Für ein Entscheidungsproblem L enthält die Komplexitätsklasse RP(L) alle Entscheidungsprobleme L'; für die es einen RP-Algorithmus mit einem Orakel für L gibt. Analog werden die Klassen RP(C); ZPP(L); ZPP(C); BPP(L); BPP(C); PP(L); PP(C) definiert.

Mit Hilfe der Definition 10.5.2 werden die folgenden Theoreme hergeleitet.

Theorem 10.5.3: $BPP \subseteq RP(NP) \cap coRP(NP)$

Das Theorem besagt, dass BPP keine Probleme enthält, die „weit“ von NP entfernt sind.

Beweis: siehe Theorem 10.5.1, mindestens die hälfte aller z sind gute Vektoren.

Wenn $BPP \subseteq NP$ wäre, wären alle Probleme praktisch lösbar, was aber wohl nicht der Fall ist.

Stellt sich nun die Frage wie mächtig ist ein BPP-Algorithmus mit BPP-Orakel?

Theorem 10.5.4: BPP(BPP) = BPP

Beweis: Sei $L \in BPP(BPP)$, dann existiert ein Orakel $L' \in BPP$ mit $L \in BPP(L')$ und A sei der äußere durch p_1 beschränkte Algorithmus mit einer Fehlerwahrscheinlichkeit von 1/6 und A' der Algorithmus von L' mit einer Fehlerwahrscheinlichkeit von $1/(6*p_1(|x|))$. Wenn nun die Anfragen an das Orakel auf A' umgeleitet werden erhält man einen neuen polynomiellen randomisierten Algorithmus, der ohne Orakel funktioniert. Fehler des neuen Algorithmus entstehen nur bei Fehlern von A bzw. A'. Somit ergibt sich eine Fehlerwahrscheinlichkeit von $\frac{p_1(|x|)}{(6*p_1(|x|))} + \frac{1}{6} = \frac{1}{3}$. Somit ist der neue Algorithmus ein BPP-Algorithmus für L.

Mit Hilfe von Theorem 3.3.6 kann der Beweis analog für alle BPP-Probleme durchgeführt werden und so generalisiert werden. Das Theorem 10.5.4 ist ein weiteres Anzeichen dafür, dass sich BPP von NP unterscheidet und dabei nicht so mächtig ist wie NP.

Korollar 10.5.6: $NP \subseteq BPP \Rightarrow PH \subseteq BPP$

Beweis: Unter der Voraussetzung das Lemma 10.5.7 gilt kann der Beweis in vier Schritten geführt werden, in dem gezeigt wird, dass für alle k $\Sigma_k \subseteq BPP$ gilt:

$$\Sigma_{k+1} = NP(\Sigma_k) \subseteq NP(BPP) \subseteq BPP(NP) \subseteq BPP(BPP) = BPP$$

Lemma 10.5.7: $NP(BPP) \subseteq BPP(NP)$

Beweis: Sei L ∈ NP(BPP), d.h. Für n = |x| existiert ein B ∈ BPP und ein Polynom p mit $x \in L \Leftrightarrow \exists y \in \{0,1\}^{p(n)} : (x, y) \in B$. B sei ein randomisierter Algorithmus A_B mit der Rechenzeit q(n) und Fehlerwahrscheinlichkeit $\leq 2^{-p(n)-2}$. Sei nun $C := \{(x, y, r) \mid y \in \{0,1\}^{p(n)}, r \in \{0,1\}^{q(n)}$:

A_B akzeptiert (x, y) mit Zufallsvektor r}, wobei offensichtlich C ∈ P.

Sei nun A' folgender randomisierter Algorithmus:

Eingabe x, Erzeuge zufällig r ∈ $\{0,1\}^{q(n)}$, Orakelalgorithmus für $\exists y \in \{0,1\}^{p(n)} : (x, y, r) \in C$

Falls x ∈ L:

Existiert y_x mit (x, y_x) ∈ B. Dabei gilt:

Prob(A' akzeptiert x) = $\text{Prob}_r(\exists y : (x, y, r) \in C) \geq \text{Prob}_r(\exists y : (x, y, r) \in C) \geq 1\text{-}2^{-p(n)-2}$.

Falls x ∉ L:

Gilt für jedes y ∈ $\{0,1\}^{p(n)}$: $\text{Prob}_r((x, y, r) \in C) \leq 2^{-p(n)-2}$.

Somit ergibt sich für die Vereinigung über alle $2^{p(n)}$ mögliche y:

$$\text{Prob}_r(\exists y : (x, y_x, r) \in C) \leq 2^{p(n)} * 2^{-p(n)-2} = 1/4.$$

Der Algorithmus macht also beidseitige Fehler mit einer Schranke von ¼ und somit ist L ∈ BPP(NP).

Auch dieser Beweis kann verallgemeinert werden, indem die gesamte Kraft der Klasse BPP genutzt wird, da in diesem Beweis nur sehr beschränkt auf das NP-Orakel zugegriffen wurde.

An das folgende Theorem glaubt man nicht, es wurde aber bisher nicht das Gegenteil bewiesen.

Theorem 10.5.9: $NP \subseteq BPP \Rightarrow NP = RP$

Beweis: Da $RP \subseteq NP$ gilt, bleibt noch zu zeigen, dass $NP \subseteq BPP \Rightarrow NP \subseteq RP$ gilt. Es genügt nun zu zeigen, dass für ein beliebiges NPC-Problem L gilt $NP \subseteq BPP \Rightarrow L \in RP$, denn für alle L' ∈ NP gilt L' $\leq_p$ L und wenn L ∈ RP => L' ∈ RP. Im Folgenden wird die Beweisidee für das Färbungsproblem von Graphen aufgezeigt. Dazu werden drei Varianten des Färbungsproblems betrachtet:

<u>*GC*</u>: Bei der Eingabe G = (V;E), k antwortet der Algorithmus ob es eine legale Färbung von G mit k Farben gibt. Es handelt sich also um ein NP-vollständiges Entscheidungsproblem.

<u>*Lex-GC*</u>: Bei der Eingabe G = (V;E). Berechnet der Algorithmus den Färbungsvektor, der unter allen legalen Färbungen mit minimaler Färbungszahl X(G) der lexikographisch kleinste ist. Dieser soll f(G) heißen. Bei diesem Problem handelt es sich um ein eindeutig lösbares Suchproblem.

<u>*Min-GC*</u>: Bei der Eingabe G = (V;E) und $c \in \{1, ..., n\}^n, n = |V|$ prüft der Algorithmus ob c ≥ f(G) ist, dabei muss c keine legale Färbung sein.

Beweisidee:

1. Zeige, dass $Min - GC \in \Sigma_2$ gilt.
2. Aus der Voraussetzung folgt $\Sigma_2 \subseteq BPP$, nach Korollar 10.5.5
3. Also gilt

   $Min - GC \in BPP \Rightarrow Lex - GC \in BPP$
4. $Lex - GC \in BPP \Rightarrow GC \in RP$

Zum vierten Schritt fehlt noch die Angabe des RP-Algorithmus für GC:

Sei A ein BPP-Algorithmus für LEX-GC. Nun folgt die Beschreibung eines Algorithmus A_0 für GC:

1. Eingabe: G = (V;E); k.
2. Rufe A für G auf => liefert das Ergebnis c.
3. Akzeptiere, wenn c legal für G ist und höchstens k Farben benutzt, sonst verwerfe die Färbung.

Die Antwort „Akzeptieren“ ist korrekt, da wir eine passende Färbung kennen. Also macht A_0 einseitige Fehler. Wenn A auf G keinen Fehler macht, ist c = f(G). Dann ist c eine legale Färbung für G.

Falls c mehr als k Farben benutzt, gibt es keine Färbung mit k Farben und die Antwort „Verwerfen“ ist korrekt. => Fehlerwahrscheinlichkeit ≤ 1/3 (wie bei Algorithmus A). Die Lex-GC Antwort konnte fehlerhaft sein, wenn sie korrekt ist kann damit $X(G) \leq k$ überprüft werden und so der Fehler auf eine Antwortseite verlagert werden.

7 Zusammenfassung

Die strukturelle Komplexitätstheorie klärt das Verhältnis zwischen Komplexitätsklassen besser auf und ermöglicht Resultate für konkrete Probleme. Weiterhin wurde die Klasse NPI eingeführt, die zwischen den Klassen P und NPC liegt. Die Betrachtung der Orakelklassen bringt ein besseres Verständnis für die Beziehungen zwischen den interessanten Klassen wie NP, BPP und RP. Außerdem hat die Betrachtung der Klassen zwischen P und NPC bzw. oberhalb davon aufgezeigt, dass viele Beweisideen für P = NP bzw. P ≠ NP von vorneherein zum scheitern verurteilt sind und gar nicht erst verfolgt werden. Die Einführung der polynomiellen Hierarchie ermöglicht eine plastischere Vorstellung über den Zusammenhang aller vorgestellten Komplexitätsklassen. Bei der Untersuchung dieser Klassen wurde „nebenbei“ die P≠NP-These weiter erhärtet.

8 Literatur

[1] Wegener, Ingo: “Complexity Theory – Exploring the Limits of Efficient Algorithms”, Berlin/Heidelberg 2005.

[2] Wegener, Ingo: „Effiziente Algorithmen und Komplexitätstheorie mit dem Schwerpunkt Komplexitätstheorie“, Vorlesung im WS 2004/05 an der Universität Dortmund.