5 Parametrisierte Komplexität

Es sei daran erinnert, dass ein „Vertex Cover“ eines Graphen $G = (V, E)$ eine Teilmenge $C \subseteq V$ der Knotenmenge ist, die von jeder Kante $e \in E$ mindestens einen Randknoten enthält. Mit Hilfe der Knoten aus C kann man gewissermaßen alle Kanten des Graphen kontrollieren. Es sei weiter daran erinnert, dass eine unabhängige Menge (independent set) eines Graphen $G = (V, E)$ eine Teilmenge $U \subseteq V$ der Knotenmenge ist, die aus paarweise nicht benachbarten Knoten besteht. Im Gegensatz dazu ist eine Clique in G ist eine Teilmenge $U \subseteq V$ der Knotenmenge, die aus paarweise benachbarten Knoten besteht. Offensichtlich gilt folgendes für die Entscheidungsprobleme IS (Independent Set) und VC (Vertex Cover):

- U ist eine Clique in $G = (V, E)$ gdw U eine unabhängige Menge in $\bar{G} = (V, \bar{E}) = (V, (V \times V) \setminus E)$ ist. Daher ergibt sich mit der Reduktionsabbildung $(G, k) \mapsto (\bar{G}, k)$ die Karp-Reduktion CLIQUE $\leq_{pol}$ IS: eine Clique der Größe k in G wäre nämlich eine unabhängige Menge der Größe k in $\bar{G}$ und umgekehrt. Die NP-Härte von CLIQUE verberbt sich somit auf IS.

- U ist eine unabhängige Menge in $G = (V, E)$ gdw $\bar{U} = V \setminus U$ ein Vertex Cover in G ist. Daher ergibt sich mit der Reduktionsabbildung $(G, k) \mapsto (G, |V| - k)$ die Karp-Reduktion IS $\leq_{pol}$ VC: eine unabhängige Menge der Größe k in G liefert und mit ihrem Komplement nämlich ein Vertex Cover der Größe $|V| - k$ in G und umgekehrt. Die NP-Härte von IS vererbt sich daher auf VC.

In Bezug auf Bezug auf Entscheidungs- oder Optimierungsalgorithmen sind CLIQUE, IS und VC „isomorphe“ Probleme. Wir wissen bereits, dass dies in Bezug auf Approximationsalgorithmen falsch ist, da VC zur Klasse APX der (mit Güte 2) approximierbaren Probleme gehört, wohingegen IS (unter der $P \neq NP$ Voraussetzung) keine Approximationsalgorithmen mit konstanter Güte zulässt. In diesem Abschnitt wird sich ein ähnlicher Gegensatz ergeben: VC gehört zur Klasse FPT der sogenannten „Fest-Parameter behandelbaren“ Probleme, wohingegen „IS (unter der $P \neq NP$ Voraussetzung) nicht zu FPT gehört. Die Aussage über IS gilt entsprechend auch für das folgende Problem:

Dominating Set (DS) Knotenüberwachung durch möglichst wenige Knoten

> **Eingabe** Ein Graph $G = (V, E)$ und eine Kostenschranke k
>
> **Frage** Kann man mit k Knoten alle Knoten überwachen, d.h., existiert eine Teilmenge $D \subseteq V$ (genannt „dominating set“) der Maximalgröße k, so dass jeder Knoten $v \in V \setminus D$ mindestens einen Nachbarn in D besitzt.
>
> **Kommentar** Dieses Problem findet Anwendungen bei der Überwachung von Rechnern in einem Rechnernetz.

Oberflächlich betrachtet sind die Probleme VC und DS enge Verwandte, wie durch die folgende polynomielle Reduktion unterstrichen wird:

Lemma 5.1 *VC* $\leq_{pol}$ *DS.*

Beweis Es sei (G, k) mit $G = (V, E)$ eine gegebene Eingabeinstanz von VC. Wir verwenden die (effizient berechenbare) Eingabetransformation $(G, k) \mapsto (G', k + n)$, wobei G' aus G durch die in Abbildung 1 dargestellte lokale Ersetzung hervorgeht.

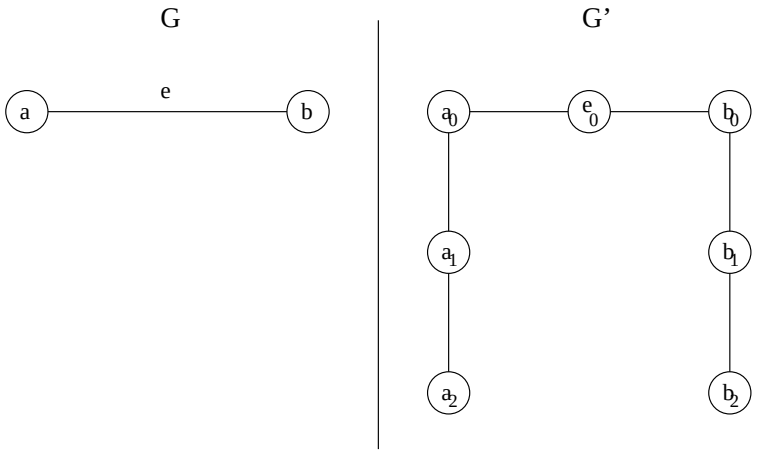

Abbildung 1: Lokale Ersetzung bei der polynomiellen Reduktion von VC auf DS.

Wie aus der Abbildung ersichtlich hat G' die Knotenmenge $V_0 \cup V_1 \cup V_2 \cup E_0$ (mit der offensichtlichen Bedeutung von V_0, V_1, V_2, E_0). Die Kantenmenge von G' ist ebenfalls leicht aus der Abbildung abzulesen. Wir werden im Folgenden die Knoten aus V mit den Knoten aus V_0 identifizieren. Der Beweis wird abgeschlossen durch die
Behauptung Es gibt genau dann ein Vertex Cover der Maximalgröße k in G, wenn es eine Dominating Set der Maximalgröße $k + n$ in G' gibt.

Nehmen wir an, dass wir in G jede Kante mit Knoten aus $U \subseteq V$ überwachen können und $|U| \leq k$. Dann können wir in G' jeden Knoten mit den Knoten aus $U \cup V_1$ überwachen. Es gilt $|U \cup V_1| = |U| + n \leq k + n$.

Nehmen wir an, dass wir in G' jeden Knoten mit Knoten aus $U' \subseteq V_0 \cup V_1 \cup V_2 \cup E_0$ überwachen können und $|U'| \leq k+n$. Wir können oBdA annehmen, dass in U' keine Knoten aufgenommen werden, die in ihrer „Überwachungspotenz“ von anderen Knoten dominiert werden. Zum Beispiel ist v_1 „potenter“ als v_2, da v_1 die Knoten v_0, v_1, v_2 überwacht und v_2 lediglich die Knoten v_1, v_2. Somit gilt oBdA $U' \cap V_2 = \emptyset$. Da andererseits alle Knoten aus V_2 überwacht werden müssen und v_2 nur von sich selbst oder von v_1 aus überwacht werden kann, gilt $V_1 \subseteq U'$. Durch V_1 werden bereits alle Knoten aus V_0 überwacht, aber noch keine Knoten aus E_0. Das Restproblem ist also auf die Überwachung der Knoten aus E_0 zusammengeschrumpft. Unter diesen Bedingungen gilt für jede Kante $e = \{a, b\}$, dass a_0 (bzw. b_0) „mindestens so potent“ ist wie e_0. Knoten a_0 (bzw. b_0) könnte nämlich potentiell neben e_0 noch weitere Knoten aus E_0 überwachen. Wir können also oBdA annehmen, dass U' die Form $U' = V_1 \cup U_0$ mit $U_0 \subseteq V_0$ hat, wobei die maximal k Knoten aus U_0 die Überwachung von E_0 gewährleisten. Es folgt, dass $U = \{v |\ v_0 \in U_0\}$ ein Vertex Cover der Maximalgröße k in G ist. **qed.**

Folgerung 5.2 *DS ist NP-vollständig.*

In Abschnitt 5.1 werden wir die Fest-Parameter Behandelbarkeit von VC herausarbeiten. In Abschnitt 5.2 präsentieren wir eine formale Definition der Klasse FPT und gehen auf den dazu passenden Reduktionsbegriff ein, mit Hilfe dessen die Nichtmitgliedschaft von IS und DS in FPT (unter der $P \neq NP$ Voraussetzung) bewiesen werden könnte.

5.1 Ein Fest-Parameter behandelbares Beispielproblem

Der Star dieses Unterabschnittes ist das Problem VC. Wir wollen im folgenden herausarbeiten, dass VC gegenüber CLIQUE, IS oder DS eine Sonderrolle einnimmt.

Beginnen wir mit einer Gemeinsamkeit der genannten Probleme. Wenn wir die Kostenschranke k zu einer Konstanten einfrieren, dann werden Probleme von der Art CLIQUE, IS, VC oder DS trivialisiert. Wir können nämlich mit *Exhaustive Search* alle $\binom{n}{k}$ Teilmengen $U \subseteq V$ der Größe k der Reihe nach durchmustern und jeweils auf die entsprechende Eigenschaft testen. Falls k eine von $n = |V|$ unabhängige Konstante ist, dann gilt $\binom{n}{k} = O(n^k)$ und wir erhalten eine polynomielle Zeitschranke.

Nun ist n^k zwar ein Polynom, aber $\Omega(n^k)$ Rechenschritte sind für große Konstanten k (selbst bei moderater Eingabelänge n) nicht tolerierbar. Der Ansatz der parametrisierten Komplexität ist zu überprüfen, ob nicht auch eine Zeitschranke der Form $f(k)n^c$ eingehalten werden kann. Hierbei ist f eine beliebige, nur von k (aber nicht von n) abhängige Funktion und $c \geq 1$ ist eine von n und k unabhängige Konstante. Für konstantes k und $c = 1$ hätten wir zum Beispiel eine lineare Laufzeitschranke vorliegen, im Vergleich zu n^k eine radikale Verbesserung.

Für VC geht dieser Plan voll auf:

Satz 5.3 (Downey und Fellows, 1992) *VC kann in $O(2^k n)$ Schritten (auf einer Random Access Maschine mit uniformem Kostenmaß) gelöst werden.*

Beweisskizze Wir skizzieren nur die Idee, die den Faktor 2^k ins Spiel bringt. Von jeder Kante $e = \{v, w\}$ enthält ein Vertex Cover den Knoten v oder den Knoten w. Wir können daher einen binären Entscheidungsbaum T aufbauen. Jeder Knoten von T repräsentiert ein partielles Vertex Cover, das anfangs (im Wurzelknoten von T) noch leer ist. Jede neue binäre Entscheidung führt dazu, dass T sich an einem bisherigen Blattknoten z wieder binär verzweigt und das von z repräsentierte partielle Vertex Cover um einen Knoten erweitert wird. Um das Wachstum von T levelweise voranzutreiben, protokollieren wir an jedem Knoten z die korrespondierende partielle Lösung U_z und die Menge der noch unüberwachten Kanten E_z. Wir brechen den Wachstumsprozess von T an einem Blatt z ab, wenn (der Misserfolgsfall) $|U_z| = k, E_z \neq \emptyset$ oder wenn (der Erfolgsfall) $E_z = \emptyset$. T braucht (wegen der Kostenschranke k) nicht über Tiefe k hinaus wachsen und enthält daher $O(2^k)$ Knoten. Falls T zu wachsen aufhört, ohne dass der Erfolgsfall eingetreten ist, dann existiert kein Vertex Cover der Größe k. Im Erfolgsfall hingegen haben wir ein Vertex Cover der Maximalgröße k aufgespürt. Der Overhead, der an jedem Knoten von T zu leisten ist, ist polynomiell in n beschränkt. Eine

verfeinerte Implementierung und eine genauere Analyse zeigen, dass eine Laufzeitschranke der Form $O(2^k n)$ eingehalten werden kann. □

Die in diesem Beweis benutzte Datenstruktur trägt den Namen „beschränkter Suchbaum (bounded search tree)“. Eine weitere Schlüsseltechnik ist die sogenannte „Reduktion auf einen Problemkern“, deren Anwendung auf Vertex Cover zu folgendem Resultat führt:

Satz 5.4 (Buss, 1989) *VC kann in $O(k^k + n)$ Schritten (auf einer Random Access Maschine mit uniformem Kostenmaß) gelöst werden.*

Beweisskizze Wir skizzieren nur die Idee, die zu der „additiven“ Form $f(k) + g(n)$ der Zeitschranke führt. Die Reduktion auf den Problemkern von Vertex Cover mit fester Kostenschranke k nutzt die folgenden offensichtlichen Tatsachen aus:

- Jeder Knoten von einem k überschreitenden Grad muss in das „Vertex Cover“ C aufgenommen werden.
- Bei einem Graphen ohne isolierte Knoten ist jedes „vertex cover“ auch eine „dominating set“. Wenn jeder Knoten maximal k Nachbarn hat und es gibt ein „Vertex Cover“ (und somit eine „dominating set“) der Größe k', dann kann es maximal $k'(k+1)$ Knoten geben.

Dies legt folgenden Algorithmus nahe:

1. Teste, ob es mehr als k Knoten mit jeweils mehr als k Nachbarn gibt. Falls JA, dann kann es kein „Vertex Cover“ der Größe k geben. Falls NEIN, dann nimm die $p \leq k$ Knoten mit jeweils mehr als k Nachbarn in das (anfangs leere) „Vertex Cover“ C auf, reduziere die Kostenschranke auf $k' := k - p$, entferne aus G alle Knoten aus C und die zu ihnen inzidenten Kanten sowie alle resultierenden isolierten Knoten (die zu einer Überwachung der Kanten des Restgraphen nichts beitragen können) und erhalte so den „Restgraphen“ G'.
2. Teste, ob G' mehr als $k'(k+1)$ Knoten enthält. Falls JA, dann kann es in G' kein „Vertex Cover“ der Größe k' (und somit in G kein „Vertex Cover“ der Größe k) geben. Falls NEIN, dann mache weiter wie folgt.
3. Teste, ob G' (ein Graph mit maximal $k'(k+1)$ Knoten) ein „Vertex Cover“ C' der Größe k' besitzt. Falls JA, dann gib $C \cup C'$ als „Vertex Cover“ der Größe k von G aus. Falls NEIN, dann existiert kein „Vertex Cover“ der geforderten Größe.

Bei geschickter Implementierung kann die Reduktion auf den Problemkern (hier: die Berechnung des Restgraphen G') in Linearzeit geschehen. Die Berechnung eines „Vertex Cover“ einer Größe von maximal $k' \leq k$ (sofern vorhanden) kann bei geschickter Implementierung in $O(k^k)$ Schritten geschehen. Dies führt zu der Zeitschranke $O(k^k + n)$. □

Folgerung 5.5 (Balasubramanian, Downey, and Fellows, 1992) *VC kann in* $O(2^k k^2 + n)$ *Schritten (auf einer Random Access Maschine mit uniformem Kostenmaß) gelöst werden.*

Beweis Reduziere zuerst auf den Problemkern und wende auf diesen die Methode mit den beschränkten Suchbäumen an. **qed.**

Der folgende Satz stammt aus dem Jahre 2006 und liefert eine sehr komfortable Zeitschranke für Vertex Cover:

Satz 5.6 — *ohne Beweis* —
VC kann in $O\left(kn + 1.2738^k\right)$ *Schritten (auf einer Random Access Maschine mit uniformem Kostenmaß) gelöst werden.*

Diese Schranke ist um eine „Galaxie“ besser als die Zeitschranke n^k eines naiven Verfahrens!

Für welche Probleme außerhalb von P lässt sich ein Eingabeparameter k dingfest machen, so dass Lösungsalgorithmen mit einer Zeitschranke der Form $f(k)n^c$ existieren? Für welche Probleme existieren solche Eingabeparameter k nicht? Um diese Frage korrekt zu beantworten bedarf es einiger abstrakter Definitionen, die den Kern der Fest-Parameter Behandelbarkeit herausarbeiten.

5.2 Probleme innerhalb und außerhalb FPT

Definition 5.7 *Ein* parametrisiertes Problem *(sprich: eine* parametrisierte formale Sprache*) liegt vor, wenn die Eingabeparameter in einen* Variablen-Parameter Teil *und einen* Fest-Parameter Teil *zerfallen. Syntaktisch korrekte Eingaben sind also Paare* (x, p) *— genauer: Kodierungen solcher Paare — wobei* x *die Kollektion der Variablen-Parameter und* p *die Kollektion der Fest-Parameter repräsentiert.*

Definition 5.8 *Eine parametrisierte Sprache heißt* Fest-Parameter behandelbar (fixed-parameter tractable)*, wenn ihr Mitgliedschaftsproblem in* $f(k)n^c$ *Schritten lösbar ist. Hierbei ist* n *die Länge von* x*,* k *die Länge von* p*,* f *eine nur von* k *(aber nicht von* n*) abhängige Funktion und* $c \geq 1$ *eine von* n *und* k *unabhängige Konstante. Die Klasse der Fest-Parameter behandelbaren parametrisierten Sprachen bezeichnen wir als FPT.*

Aus der Definition von FPT folgt sofort:

$$P \subseteq \text{FPT} \tag{1}$$

Bemerkung 5.9 *Die Definition von FTP ist robust gegen einige Modifikationen. Zum Beispiel ändert sich die Klasse FPT nicht, wenn wir* p *und* k *identifizieren (also etwa voraussetzen, dass* p *die unäre Kodierung einer Zahl* k *ist). Von dieser Vereinfachung machen wir meistens Gebrauch und sprechen von dem Fest-Parameter* k *der Eingabe. Weiterhin ändert sich FPT nicht, selbst wenn wir Laufzeitschranken der Form* $f(k) + n^c$ *(anstelle* $f(k) \cdot n^c$*) fordern (wie 1997 von Cai, Chen, Downey und Fellows gezeigt wurde).*

Mit diesen Definitionen können wir das Resultat des vorigen Abschnitts zusammenfassen wie folgt:

Satz 5.10 *$VC \in FPT$.*

Obwohl wir im Rahmen dieser Vorlesung nicht näher darauf eingehen können, sei bemerkt, dass FPT eine sehr reichhaltige Klasse ist, und dass die Nachweise zur Mitgliedschaft in FPT einige grundlegende Methoden und Tricks zum Algorithmenentwurf hervorgebracht haben. Andererseits gibt es auch eine breite Palette von Problemen, die (vermutlich) nicht zu FPT gehören. In diese Kategorie der *Fest-Parameter harten* Probleme gehören zum Beispiel CLIQUE, IS und DS. Die Vermutung der Fest-Parameter Härte eines Problems wird wieder mit einem geeigneten Reduktionsbegriff gestützt. Anders als bei der Klasse *NPC* haben sich aber mehrere harte Klassen gebildet, die oberhalb von FPT eine Hierarchie bilden. Auch hierauf können wir im Rahmen dieser Vorlesung nicht näher eingehen. Wir definieren lediglich den Reduktionstypus, den man in der Fest-Parameter Komplexitätstheorie zugrunde legen sollte:

Definition 5.11 *Seien L, L' parametrisierte formale Sprachen. L heißt* Fest-Parameter transformierbar *in L', falls eine in $f(k)n^c$ Schritten berechenbare* Fest-Parameter Eingabetransformation $(x, k) \mapsto (x', k')$ *mit $k' = g(k)$ und $(x, k) \in L \Leftrightarrow (x', k') \in L'$ existiert. Hierbei ist n die Länge von x, f, g sind nur von k (aber nicht von n) abhängige Funktionen und $c \geq 1$ ist eine von n und k unabhängige Konstante.*

Zum Beispiel ist die (aus der Vorlesung „Theoretische Informatik“ bekannte) polynomielle Reduktion von CLIQUE nach IS (Übergang zum Graphen mit der gleichen Knoten- aber komplementären Kantenmenge) eine Fest-Parameter Transformation (mit $g(k) = k' = k$). Die polynomiellen Reduktionen von IS nach VC (mit $k' = n - k$) und von VC nach DS (mit $k' = n + k$) hingegen sind keine Fest-Parameter Transformationen, da Fest-Parameter k' jeweils eine (verbotene) Abhängigkeit von n aufweist.

Offensichtlich gilt:

Lemma 5.12 *Falls L Fest-Parameter transformierbar in L' ist und $L' \in FPT$, dann folgt $L \in FPT$.*

Umgekehrt bedeutet dies natürlich, dass sich eine etwaige Fest-Parameter Härte von L auf L' vererben würde.

Wir bemerken abschließend, dass die Hierarchie die von FPT und den diversen Klassen von Fest-Parameter harten Problemen gebildet wird „windschief“ zur polynomiellen Platz-Zeit Hierarchie liegt. Probleme aus *NP* können bereits Fest-Parameter hart sein. Andererseits gehören sogar einige PSPACE-vollständige Probleme zur Klasse FPT.

6 Zwischenbilanz zum Thema „P versus NP“

Bei großen Softwareprojekten entstehen nach einer Strukturierungs- und Modellierungsphase meist klar umrissene Teilprobleme. Ein Lernziel beim Studium der *Effizienten Algorithmen* und der *Komplexitätstheorie* ist

- zu erkennen, wenn ein Teilproblem mit einem effizienten Algorithmus lösbar ist,
- zu erkennen, wenn dies nicht der Fall ist (zum Beispiel aufgrund der *NP*-Härte des Problems).

NP-harte Probleme verschwinden nicht, wenn wir sie als *NP*-hart erkannt haben. Es stellt sich also weiterhin die Frage, wie mit harten Problemen umzugehen ist. In den voran gegangenen Abschnitten haben wir ein paar Methoden zum Umgang mit NP-harten Problemen (sowie die theoretischen Grenzen dieser Methoden) kennen gelernt:

Einschränkung des Problems Suche nach (praktisch relevanten) und effizient lösbaren Teilproblemen eines NP-vollständigen Problems.
Zum Beispiel sind 2-SAT (im Gegensatz zu 3-SAT) und 2-dimensionales Matching (im Gegensatz zum 3-dimensionalen Matching) effizient lösbar. PRECEDENCE CONSTRAINED SCHEDULING wird effizient lösbar, wenn wir nur Bäume (Sortier- oder Montagebäume) als partielle Relation auf der Menge der Aufgaben zulassen. Manche Zahlenprobleme (wie PARTITION oder KNAPSACK) sind pseudopolynomiell lösbar und somit polynomiell lösbar im Falle „kleiner“ Zahlparameter in der Eingabe. Die Grenzen dieses Ansatzes ergeben sich durch extrem eingeschränkte aber immer noch NP-harte Teilprobleme bzw., im Falle der Zahlprobleme, durch das Phänomen der starken NP-Vollständigkeit.

Aufgabe des Anspruches der Optimalität Versuche ein NP-hartes Optimierungsproblem mit einem Approximationsalgorithmus zu lösen, welcher in Polynomialzeit eine „gute“ (aber i.A. nicht optimale) Lösung liefert.
Die Grenzen dieses Ansatzes (den wir am Beispiel der Probleme TSP und KNAPSACK diskutiert haben) ergeben sich durch das Phänomen der NP-harten Approximation.

Fest-Parameter Behandelbarkeit Versuche aus den Eingabeparametern einen sogenannten „Fest-Parameter“ k zu isolieren, der für die NP-Härte des Problems verantwortlich ist und suche einen Algorithmus mit einer Zeitschranke der Form $f(k)n^c$.
Diesen Ansatz haben wir am Beispiel von Vertex Cover diskutiert. Seine Grenzen findet er im Phänomen der Fest-Parameter Härte.

Ein anderer (bisher von uns nicht diskutierter) Ansatz besteht darin ein mächtigeres Maschinenmodell als die Deterministische Turing Maschine (DTM) zu verwenden. Wir diskutieren ein paar konkrete Vorschläge:

Probabilistisches Maschinenmodell Statte M mit einer *perfekten Münze* aus. M hat pro Schritt zwei Handlungsalternativen und wählt durch Münzwurf eine davon zufällig

aus. Dieser Ansatz führt zum Modell der *Probabilistischen Turing Maschine (PTM)*. Um das Mitgliedschaftsproblem für Sprache L zu lösen, muss eine PTM M Eingaben $x \in L$ mit "hinreichend hoher" Wahrscheinlichkeit akzeptieren und Eingaben $x \notin L$ mit "hinreichend hoher" Wahrscheinlichkeit verwerfen. Wir gehen in einem späteren Stadium der Vorlesung noch genauer auf PTMs und die entsprechenden Komplexitätsklassen ein.

Parallelrechnermodell Verwende p parallel arbeitende Prozessoren, die miteinander kommunizieren können, um kooperativ ein gemeinsames Problem zu lösen. In diese Rubrik fallen neben der *Parallel Random Access Machine (PRAM)* und den diversen *Rechnernetzwerken* auch die sogenannten (in den letzten zwei Dekaden in Mode gekommenen) *DNA-Rechner*. Ein *NP*-hartes Problem lässt sich allerdings auch von einem Parallelrechner nicht zufriedenstellend lösen, da jeder Faktor an Zeitgewinn mit einem mindestens ebenso großen Faktor an zusätzlichem Hardwareaufwand bezahlt wird. Wir können im Rahmen dieser Vorlesung auf Parallelrechner nicht näher eingehen.

Qauntenrechner Die DTM und dazu verwandte Modelle sind mathematische Abstraktionen von physikalischen Rechnern, die den Gesetzen der klassischen Mechanik gehorchen. In den letzten zwei Dekaden wurde eine neue Generation von Rechnern vorgeschlagen, die den Gesetzen der Quantenmechanik gehorchen. Ein Quantenrechner ist zu einem bestimmten Zeitpunkt nicht in **einer** Konfiguration, sondern in einer **Überlagerung von vielen** Konfigurationen. Erst durch externe Beobachtung kollabiert die Überlagerung zu einer einzigen Konfiguration (gemäß einer zur Überlagerung assoziierten Wahrscheinlichkeitsverteilung). Quantenrechner sind vermutlich wesentlich mächtiger als DTMs. Zum Beispiel konnte Peter Shor nachweisen, dass das Faktorisierungsproblem (Berechnung der Primfaktorzerlegung einer gegebenen ganzen Zahl) und das Diskrete-Logarithmus-Problem auf Quantenrechnern effizient lösbar sind. Auf der vermeintlichen Härte dieser Probleme beruhen die meisten aktuell verwendeten asymmetrischen Kryptosysteme. Es ist zur Zeit noch völlig unklar, ob Quantenrechner physikalisch realisiert werden können. Im Rahmen dieser Vorlesung werden wir auf Quantenrechner nicht weiter eingehen können.

Ein weiterer Ansatz, den wir im Rahmen dieser Vorlesung nicht behandeln können ist die

Average-Case Analyse Average-case Analyse steht im Gegensatz zur worst-case Analyse. Hier wird die Forderung aufgegeben, auf allen Eingaben erfolgreich sein zu müssen. Man verwendet ein Wahrscheinlichkeitsmaß auf den Eingabeinstanzen und ist zum Beispiel zufrieden, wenn die mittlere Laufzeit polynomiell ist. Ein Argument **für average-case Analyse** (oder zumindest gegen worst-case Analyse) ist, dass die von polynomiellen Reduktionen produzierten Eingabeinstanzen i.A. kunstvoll und bizarr anmuten. Es könnte also sein, dass genau der Eingabetypus, welcher die *NP*-Härte bezeugt, in Anwendungen nicht (oder selten) anzutreffen ist. Ein Argument **gegen average-case Analyse** ist, dass die Auswahl eines analysierbaren Wahrscheinlichkeitsmaßes meist willkürlich und durch die Anwendung nicht zu rechtfertigen ist.

Smoothed Analysis Es handelt sich um eine Mischform von average-case und worst-case Analyse, bei welcher gemittelt wird über Eingaben, die aus leichten zufälligen Abänderungen von worst-case Eingaben entstehen. Zum Beispiel konnte gezeigt werden, dass der Simplex-Algorithmus zum Optimieren unter linearen Randbedingungen zwar im worst-case eine exponentielle Laufzeit hat. Die „Smoothed Analysis“ jedoch führt im statistischen Mittel zu einer polynomiellen Zeitschranke.

Schließlich gibt es den sogenannten „heuristischen Ansatz“ zum Lösen von NP-harten Optimierungsproblemen.

Heuristiken und Metaheuristiken Eine *Heuristik* ist ein Algorithmus, der in der Praxis ganz gut zu funktionieren scheint, ohne dass dies durch eine mathematisch beweisbare Qualitätsgarantie abgesichert wäre. *Metaheuristiken* sind vielseitig verwendbare Heuristiken, die durch geeignete Adjustierung einiger programmierbarer Parameter auf die Lösung von konkreten Problemen spezialisiert werden können. Daneben gibt es ad-hoc Heuristiken, die für ein konkretes Problem zusammengeschustert wurden. Beispiele für Metaheuristiken sind:

- Branch and Bound
- Lokale Optimierung
- Simulated Annealing
- Tabu Search
- Neuronale Netze oder Hopfield Netze
- Genetische Algorithmen
- Great Deluge (große Überschwemmung)
- Roaming Ants (umherstreifende Ameisen)

Auch diesen Ansatz werden wir aus Zeitgründen nicht weiter verfolgen können.

7 Die Struktur von NP

Falls $P = NP$, dann gibt es zur Struktur von NP nicht viel zu berichten. Alle Betrachtungen dieses Abschnittes basieren auf der

Vermutung 1 $P \neq NP$.

Wir verwenden im Folgenden Vermutung 1 als Voraussetzung, ohne dies jedesmal explizt kennntlich zu machen. Wir machen also öfter eine Aussage A, die streng genommen die Form: $P \neq NP \Rightarrow A$ haben müsste.

Es sei daran erinnert, dass NPC die Klasse der NP-vollständigen Probleme bezeichnet. Die Klasse

$$NPI := NP \setminus (P \cup NPC)$$

bezeichne die Klasse der Sprachen aus NP, die einerseits nicht zu P gehören, andererseits aber auch nicht NP-vollständig sind.

Aus der NP-Vollständigkeitstheorie folgt direkt, dass die Existenz einer NP-vollständigen Sprache in P die Gleichheit von P und NP zur Folge hätte. Vermutung 1 zugrunde gelegt, müssen also NPC und P disjunkte Teilmengen von NP sein. Mit der Definition von NPI ergibt sich eine erste Strukturaussage:

- NP zerfällt in die paarweise disjunkten Klassen P, NPC und NPI.

Diese Einsicht ist in Abbildung 2 visualisiert.

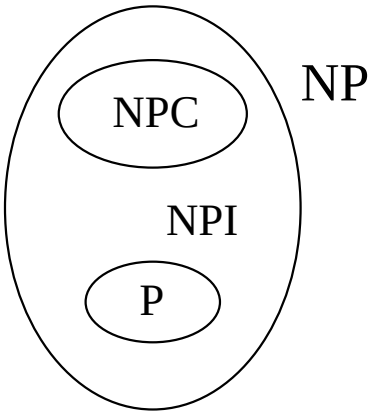

Abbildung 2: Die Struktur von NP.

Wir kennen bereits zahlreiche Sprachen (Probleme) in P bzw. NPC. Wie schaut es aber mit NPI aus: könnte es sein, dass NPI leer ist? Dies würde bedeuten, dass jede Sprache aus $NP \setminus P$ bereits NP-vollständig sein müsste. Der folgende Satz schließt diese Möglichkeit aus:

Satz 7.1 (Ladner, 1975) *$NPI \neq \emptyset$.*

Beweis Der Beweis benutzt die Technik der Diagonalisierung und basiert auf folgenden Tatsachen:

- Die Binärstrings bezitzen die „natürliche“ Aufzählung $\varepsilon, 0, 1, 00, 01, 10, 11, 000, \ldots$. Der i-te Binärstring dieser Aufzählung wird im Folgenden als $\alpha(i)$ notiert.

- Jede Boolesche CNF-Formel kann auf natürliche Weise über dem Binäralphabet $\{0, 1\}$ kodiert werden. Binärstrings, die keine ordentlichen CNF-Codewörter sind, können mit einer Formel für die Nullkonstante identifiziert werden. Auf diese Weise repräsentiert jeder Binärstring α eine CNF-Formel F_α.

- Wie wir aus dem Kapitel über die Universelle Turing-Maschine (UTM) wissen, kann auch jede DTM auf natürliche Weise durch einen Binärstring kodiert werden. Strings, die keine ordentlichen DTM-Codewörter sind, können mit einem Akzeptor für die leere Menge identifiziert werden. Auf diese Weise repräsentiert jeder Binärstring α eine

DTM M_α. T Rechenschritte von M_α sind durch $O(|\alpha| T \log T)$ Rechenschritte der UTM simulierbar.

- Der natürliche Code α einer DTM kann durch Anhängen von 01^n mit einer beliebig großen Anzahl n von Einsen „gepolstert“ werden. Dann hat jede DTM unendlich viele Codewörter. Die Rechenzeit der UTM hängt aber weiterhin nur von der Länge der nicht-gepolsterten Kodierung ab.

SAT bezeichne die Menge der Binärstrings, die eine erfüllbare CNF-Formel repräsentieren. Für jede Funktion $H : \mathbb{N} \to \mathbb{N}$ definieren wir die folgende Variante von SAT:

$$\mathrm{SAT}_H := \{\alpha 01^{n^{H(n)}} |\ \alpha \in \mathrm{SAT} \wedge n = |\alpha|\}$$

Für die folgende induktiv definierte (und trickreiche) Wahl der Funktion H gilt (wie wir unter der Voraussetzung P $\neq$ NP noch zeigen werden): $\mathrm{SAT}_H \in NPI$:

- Für $n \leq 4$ setze $H(n) = 1$.
- Für $n \geq 5$ setze $H(n)$ auf die kleinste Zahl $i < \log\log n$ mit folgender Eigenschaft: Für alle Binärstrings α mit $|\alpha| < \log n$ gibt $M_{\alpha(i)}$ nach maximal $i|\alpha|^i$ Schritten das Indikatorbit aus, welches anzeigt, ob $\alpha \in \mathrm{SAT}_H$. Falls kein in diesem Sinne „erfolgreicher“ Index i existiert, dann setze behelfsweise $H(n) = \lceil \log\log n \rceil$.

Es ist offensichtlich, dass $H(n)$ eine in n monoton wachsende Funktion ist. Somit existiert der Grenzwert $\lim_{n\to\infty} H(n)$ (evtl. ∞). Die zentralen Bausteine für den Rest des Beweises sind die folgenden weiteren Eigenschaften der Funktion H:

Behauptung 1: $H(n)$ ist in poly(n) Schritten berechenbar.

Behauptung 2: $\mathrm{SAT}_H \in P \Rightarrow \lim_{n\to\infty} H(n) < \infty$.

Behauptung 3: $\mathrm{SAT}_H \notin P \Rightarrow \lim_{n\to\infty} H(n) = \infty$.

Wir demonstrieren zunächst, wie sich mit Hilfe der drei Behauptungen $\mathrm{SAT}_H \in NPI$ ableiten lässt. Aus Behauptung 1 ergibt sich $\mathrm{SAT}_H \in NP$. Wir haben auszuschließen, dass SAT_H zu P oder zu NPC gehört. Wir zeigen, dass die Annahme einer Mitgliedschaft in P oder NPC jeweils zu einem Widerspruch zur $P \neq NP$ Voraussetzung führt. Angenommen, $\mathrm{SAT}_H \in P$. Dann hat $H(n)$ gemäß Behauptung 2 einen endlichen Grenzwert, sagen wir C. Das bedeutet aber, dass eine CNF-Formel F_α der Kodierungslänge $n = |\alpha|$ nur mit maximal n^C Einsen (polynomiell in n viele) ausgepolstert wird. Somit kann ein Polynomialzeitalgorithmus für SAT_H benutzt werden, um SAT in Polynomialzeit zu lösen. Daraus ergäbe sich $P = NP$ im Widerspruch zur $P \neq NP$ Voraussetzung. Nehmen wir nun an, dass $\mathrm{SAT}_H \in NPC$. Dies impliziert $\mathrm{SAT}_H \notin P$, was gemäß Behauptung 3 wiederum $\lim_{n\to\infty} H(n) = \infty$ impliziert. Eine CNF-Formel F_β der Kodierungslänge $|\beta|$ wird demnach im Rahmen der Sprache SAT_H mit superpolynomiell vielen Einsen ausgepolstert. Wegen der (angenommenen) NP-Härte von SAT_H (und wegen der Polynomialzeitberechenbarkeit von H) muss eine in Polynomialzeit

berechenbare Reduktionsabbildung der Form $\alpha \mapsto \beta 01^{m^{H(m)}}$ mit $m = |\beta|$ existieren, die SAT auf SAT_H reduziert. Das Anhängen von superpolynomiell in m vielen Einsen kann nur dann in poly($|\alpha|$) Schritten geschehen, wenn β signifikant kürzer als α ist. Zum Beispiel kann $|\beta| \leq \sqrt{|\alpha|}$ gefolgert werden. Iteratives Anwenden der Reduktionsabbildung führt dann aber zu einem Polynomialzeitalgorithmus für SAT im Widerspruch zur $P \neq NP$ Voraussetzung. Die Schlussfolgerung aus dieser Diskussion ist, dass $\mathrm{SAT}_H \in NPI$. Jedoch haben wir abschließend noch die drei Behauptungen zu beweisen.

Beweis von Behauptung 1: Es gibt $\log \log n$ Kandidatenzahlen i und weniger als $2n$ Kandidatenstrings α (da die Anzahl der Binärstring einer Maximallänge k gleich $2^{k+1}-1 < 2^{k+1}$ ist). Für jedes fest gewählte Paar (i, α) ist im Wesentlichen folgendes zu tun:

- Verwende die UTM, um die Rechnung von $M_{\alpha(i)}$ auf Eingabe α für $i|\alpha|^i < \log\log(n)\log(n)^{\log\log(n)}$ Schritte zu simulieren.

Für jeden fest gewählten Binärstring α ist zu testen, ob $\alpha \in \mathrm{SAT}_H$. Dies kann wie folgt geschehen:

- Teste (Test 1), ob α das Symbol 0 enthält und, gegebenenfalls, zerlege α gemäß $\alpha = \beta 01^m$.
- Interpretiere β als CNF-Formel F_β und teste (Test 2) diese auf Erfüllbarkeit (wegen $|\beta| < |\alpha| \leq \log n$ in poly(n) Schritten machbar).
- Berechne rekursiv[1] $H(|\beta|)$ und teste (Test 3), ob $m = |\beta|^{H(|\beta|)}$.
- Setze das Indikatorbit für „$\alpha \in \mathrm{SAT}_H$?“ auf 1, falls α alle drei Tests besteht; andernfalls setze das Indikatorbit auf 0.

Um das kleinste „erfolgreiche“ i zu finden, müssen die Indikatorbits zu den α-Strings mit den Indikatorbits verglichen werden, welche die DTMs $M_{\alpha(i)}$ angesetzt auf α produzieren. Bei vernünftiger Implementierung des skizzierten Algorithmus zur Berechnung von $H(n)$ ergibt sich eine polynomiell in n beschränkte Laufzeit. (Wir verzichten an dieser Stelle auf die Angabe weiterer Details.)

Beweis von Behauptung 2: Unter der Annahme $\mathrm{SAT}_H \in P$ existiert eine DTM M, welche die Mitgliedschaft in SAT_H korrekt in Polynomialzeit entscheidet. Für eine hinreichend groß gewählte Konstante j vollzieht M auf Eingaben der Länge n maximal jn^j Rechenschritte. Da M (wie jede andere DTM auch) unendlich viele Kodierungsstrings besitzt, existiert ein Index $k \geq j$ mit $M = M_{\alpha(k)}$. Aus der Definition von H ergibt sich dann aber, dass die Funktion $H(n)$ durch k nach oben beschränkt ist.

Beweis von Behauptung 3: Wir führen einen indirekten Beweis und zeigen, dass ein endlicher Grenzwert k von $H(n)$ impliziert, dass $\mathrm{SAT}_H \in P$. Grenzwert k bedeutet, dass es einen Index n_0 gibt, so dass $H(n) = k$ für alle $n \geq n_0$. Aus der Definition von H

[1] Bei der genauen Zeitanalyse muss ausgenutzt werden, dass der Parameter $|beta|$, auf welchem der rekursive Aufruf erfolgt, wesentlich kleiner ist als der Parameter n, mit dem wir gestartet waren.

ergibt sich dann, dass $M_{\alpha(k)}$ das Mitgliedschaftsproblem für die Sprache SAT_H mit Laufzeitschranke kn^k löst (wobei die endlichen vielen Strings einer Länge unterhalb von n_0 von der endlichen Kontrolle übernommen werden können).

Damit ist der Beweis des Satzes abgeschlossen. **qed.**

Wir wollen nun die Stuktur von *NP* differenzierter untersuchen.

Definition 7.2 *Wir sagen zwei Sprachen L_1 und L_2 sind* polynomiell äquivalent*, wenn sie wechselseitig aufeinander polynomiell reduzierbar sind. In Zeichen:*

$$L_1 \overset{pol}{\sim} L_2 :\Leftrightarrow L_1 \leq_{pol} L_2 \text{ und } L_2 \leq_{pol} L_1 \ .$$

Diese Relation ist (offensichtlich) eine Äquivalenzrelation. Eine Äquivalenzklasse bezüglich „$\overset{pol}{\sim}$“ heiße Schwierigkeitsgrad *bezüglich „$\overset{pol}{\sim}$“ oder kurz* p-Grad*.*

Wenn wir polynomielle Unterschiede zwischen Laufzeiten ignorieren, können wir zwei polynomiell äquivalente Sprachen als (im Wesentlichen) gleich schwer betrachten. Die folgende Definition liefert eine partielle Ordnung auf den p-Graden;

Definition 7.3 *Wir sagen L_1 ist* leichter *als L_2 bezüglich polynomieller Reduktion, wenn zwar L_1 polynomiell reduzierbar auf L_2 ist, aber nicht umgekehrt. In Zeichen (mit „$\neg$“ für „logische Negation“):*

$$L_1 <_{pol} L_2 :\Leftrightarrow L_1 \leq_{pol} L_2 \text{ und } \neg(L_2 \leq_{pol} L_1) \ .$$

Wir sagen der p-Grad von L_1 ist leichter *als der p-Grad von L_2, wenn L_1 leichter als L_2 bezüglich polynomieller Reduktion ist.*

Es ist leicht einzusehen, dass $<_{pol}$ auf Sprachen eine irreflexive, asymmetrische partielle Ordnung bildet, die auch auf p-Graden wohldefiniert (also nicht abhängig von der Auswahl des Repräsentanten eines p-Grades) ist. In diesem Lichte besteht *NP* aus einer Hierarchie von p-Graden. Eine Teilhierarchie ist in Abbildung 3 zu sehen. Die folgenden Aussagen sind nicht schwer **zu zeigen**. **Übg.**

- Die leere Menge und Σ^* bilden zwei triviale p-Grade, angesiedelt auf der untersten Stufe der Hierarchie.
- $P \setminus \{\emptyset, \Sigma^*\}$, also die restlichen Sprachen aus P, bilden den nächst-einfachsten p-Grad, den wir als P-Grad bezeichnen.
- *NPC*, also die *NP*-vollständigen Probleme, bilden den schwersten p-Grad in *NP*.
- Da *NPI* nicht leer ist, können wir willkürlich eine Sprache L aus *NPI* rausgreifen. Ihr p-Grad muss echt zwischen dem P-Grad und *NPC* liegen.

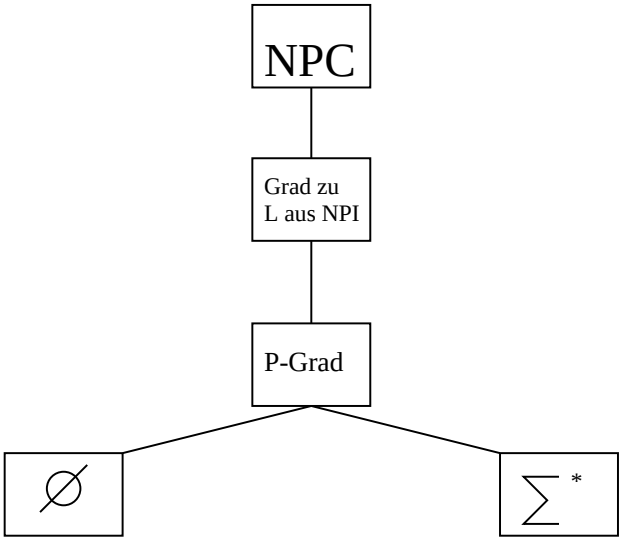

Abbildung 3: Eine Teilhierarchie der p-Grade.

Es stellt sich nun die Frage, ob die Struktur der p-Grade zwischen P und NPC nicht wesentlich komplexer ist, als dies durch Abbildung 3 suggeriert wird. Dies ist in der Tat der Fall, wie folgende Sätze belegen:

Satz 7.4 (Ladner, 1975) *1. Es gibt unendlich lange Ketten von p-Graden zwischen P und NPC.*

2. Es gibt unvergleichbare p-Grade zwischen P und NPC.

Die erste Aussage bedeutet, dass es Sprachen $L_0, L_1, L_2, \ldots, L_* \in NP$ gibt, so dass $L_0 \in P$, $L_* \in NPC$ und

$$L_0 <_{pol} L_1 <_{pol} L_2 <_{pol} \cdots <_{pol} L_* \ .$$

Die Sprachen $L_1, L_2, \ldots$ spannen eine unendliche Kette zwischen den p-Graden P und NPC auf. Die zweite Ausage macht deutlich, dass $<_{pol}$ nur eine partielle, nicht aber totale, Ordnung auf den p-Graden liefert. Der Beweis dieses Satzes, den wir in unserer Vorlesung auslassen, basiert (wie schon der Beweis von Satz 7.1) auf der Diagonalisierungstechnik.

Wir wollen im Folgenden das Bild durch die Betrachtung von co-NP abrunden. Die allgemeine Definition einer *Komplementärklasse* zu einer Sprachklasse $\mathcal{C}$ ist wie folgt:

$$\text{co-}\mathcal{C} := \{\bar{L} : L \in \mathcal{C}\} \ .$$

$\mathcal{C} = \text{co-}\mathcal{C}$ würde gerade bedeuten, dass $\mathcal{C}$ abgeschlossen unter Komplementbildung ist. Wir formulieren jetzt die Vermutung, dass NP diese Abschlusseigenschaft nicht besitzt:

Vermutung 2 $NP \neq \text{co-}NP$.

Die folgende Überlegung zeigt, dass Vermutung 2 stärker ist als Vermutung 1:

Bemerkung 7.5 *$NP \neq$ co-$NP \Rightarrow P \neq NP$.*

Beweis Wir führen einen indirekten Beweis, d.h., wir beweisen (unter Ausnutzung von $P =$ co-P) die Aussage $P = NP \Rightarrow NP =$ co-NP:

$$P = NP \Rightarrow \text{co-}NP = \text{co-}P = P = NP \ .$$

qed.

co-NP ist eine zu NP duale Klasse. Strukturen in NP haben ihr duales Pendant in co-NP. Eine erste Illustration dieses Sachverhaltes liefert

Lemma 7.6 *co-NPC, also die Komplementärklasse der NP-vollständigen Sprachen, stimmt überein mit der Klasse co-NP-vollständigen Sprachen.*

Beweis Wir nutzen die Implikation $L_1 \leq_{pol} L_2 \Rightarrow \bar{L}_1 \leq_{pol} \bar{L}_2$ aus. Somit sind die folgenden Aussagen äquivalent:

$$\begin{aligned} L_0 \in \text{co-}NPC &\Leftrightarrow \bar{L}_0 \in NPC \\ &\Leftrightarrow (\bar{L}_0 \in NP) \wedge (\forall L \in NP : L \leq_{pol} \bar{L}_0) \\ &\Leftrightarrow (L_0 \in \text{co-}NP) \wedge (\forall L \in NP : \bar{L} \leq_{pol} L_0) \\ &\Leftrightarrow (L_0 \in \text{co-}NP) \wedge (\forall L \in \text{co-}NP : L \leq_{pol} L_0) \\ &\Leftrightarrow L_0 \text{ ist co-}NP\text{-vollständig.} \end{aligned}$$

qed.

Definition 7.7 *Eine Beziehung $\leq_{POL}$ zwischen formalen Sprachen heißt* abstrakte polynomielle Reduktion*, wenn $\leq_{POL}$ transitiv ist und wenn $L \leq_{POL} L'$ und $L' \in P$ impliziert, dass $L \in P$.*

Wir merken kurz an, dass Lemma 7.6 allgemeiner hätte formuliert werden können: an der Stelle der polynomiellen Reduktion „$\leq_{pol}$“ könnte eine beliebige abstrakte polynomielle Reduktion „$\leq_{POL}$“ stehen, welche zusätzlich die Bedingung

$$L_1 \leq_{POL} L_2 \Rightarrow \bar{L}_1 \leq_{POL} \bar{L}_2$$

erfüllt. Cook- oder Levin-Reduktionen bezitzen diese Eigenschaft ebenfalls, wie sich leicht **zeigen ließe**. Somit ergibt sich die **Übg.**

Folgerung 7.8 *Die Komplementärklasse der unter Cook- bzw. Levin-Reduktionen NP-vollständigen Sprachen stimmt überein mit der Klasse unter Cook- bzw. Levin-Reduktionen co-NPC-vollständigen Sprachen.*

Das folgende Lemma gibt ein paar zur $NP \neq$ co-NP-Voraussetzung äquivalente Bedingungen an.

Lemma 7.9 *Folgende Aussagen sind äquivalent:*

1. $NP \neq co\text{-}NP$.
2. $NPC \cap co\text{-}NP = \emptyset$.
3. $co\text{-}NPC \cap NP = \emptyset$.
4. $NP \cap co\text{-}NP \subseteq NPI \cup P$.

Beweis Zur Äquivalenz der ersten beiden Aussagen genügt es freilich

$$NP = \text{co-}NP \Leftrightarrow NPC \cap \text{co-}NP \neq \emptyset$$

zu zeigen. $NP = \text{co-}NP \Rightarrow NPC \cap \text{co-}NP \neq \emptyset$ erhalten wir wie folgt:

$$NP = \text{co-}NP \Rightarrow NPC = \text{co-}NPC \Rightarrow NPC \cap \text{co-}NP = NPC \neq \emptyset \ .$$

Nehmen wir umgekehrt an, dass $NPC \cap \text{co-}NP \neq \emptyset$. Dann gibt es eine NP-vollständige Sprache L_0, die auch zu co-NP gehört. Da jede Sprache $L \in NP$ auf L_0 polynomiell reduzierbar ist, ergibt sich hieraus $NP \subseteq$ co-NP. Dies wiederum impliziert co-$NP \subseteq$ co-co-$NP = NP$ und wir erhalten $NP =$ co-NP.
Zur Äquivalenz von Aussage 2 und 3 genügt es freilich

$$NPC \cap \text{co-}NP \neq \emptyset \Leftrightarrow \text{co-}NPC \cap NP \neq \emptyset$$

zu zeigen. Hierzu nutze aus, dass für alle Sprachen L die Aussagen $L \in NPC \cap$ co-NP und $\bar{L} \in$ co-$NPC \cap NP$ äquivalent sind.
Die Äquivalenz der Aussagen 2 und 4 ist offensichtlich. **qed.**

Die durch Lemma 7.9 beschriebene Situation ist in Abbildung 4 noch einmal zusammengefasst.

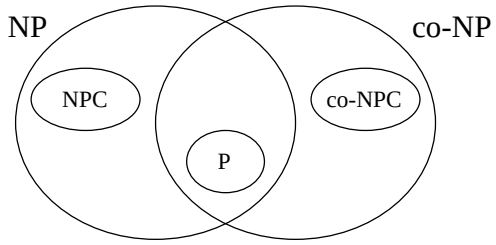

Abbildung 4: Struktur von $NP \cup$ co-NP unter der $NP \neq$ co-NP-Voraussetzung.