Einblick in die Komplexitätstheorie

Hans Ulrich Simon

1 Grundlagen und das Theorem von Cook

Während eine Theorie der Berechenbarkeit die Grenze zwischen entscheidbaren und unentscheidbaren Problemen exploriert, ist die zentrale Frage der Komplexitätstheorie, welche Probleme effizient lösbar sind und welche inhärent einen hohen Ressourcenverbrauch erfordern (s. Abbildung 1).

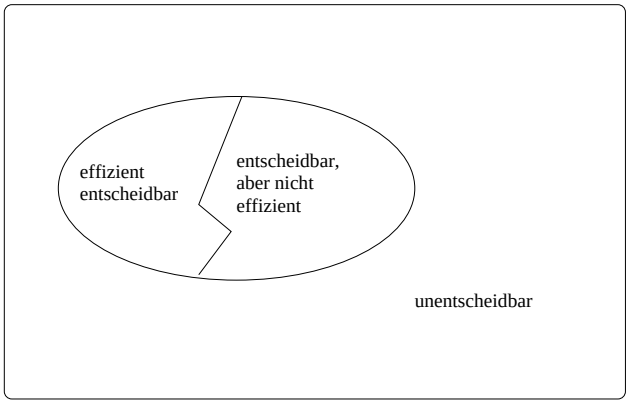

Abbildung 1: Grobunterteilung aller Probleme.

Der Ressourcenverbrauch kann und soll i.A. nicht exakt, sondern nur größenordnungsmäßig, erfasst werden. Wir werden aus diesem Grund von der Landau'schen O-Notation Gebrauch machen. Für eine Funktion f von $\mathbb{N}_0$ nach $\mathbb{N}_0$ sei die Funktionsmenge $O(f)$ definiert wie folgt:

$$O(f) = \{g : \mathbb{N}_0 \to \mathbb{N}_0 | \ \exists a, n_0 \in \mathbb{N}_0, \forall n \geq n_0 : \ g(n) \leq af(n)\}.$$

Funktionen g aus $O(f)$ heißen *größenordnungsmäßig durch f beschränkt*. Aus historischen Gründen schreibt man $g = O(f)$ anstelle von $g \in O(f)$. Falls der Grenzwert in (1) existiert, dann gilt

$$g = O(f) \Leftrightarrow \lim_{n\to\infty} \frac{g(n)}{f(n)} < \infty \ . \tag{1}$$

Zum Beispiel gilt $10n^2 + 1000n = O(n^2)$, da

$$\lim_{n\to\infty} \frac{10n^2 + 1000n}{n^2} = 10 < \infty \ .$$

Salopp gesprochen kann man sagen, dass das große O Konstanten und Terme nichtdominanter Größenordnung „schluckt". Aus diesem Grunde „überlebt" von dem Ausdruck

$10n^2 + 1000n$ nur der dominante Term n^2. Natürlich gilt auch $10n^2 + 1000n = O(n^3)$, da

$$\lim_{n\to\infty} \frac{10n^2 + 1000n}{n^3} = 0 < \infty \ ,$$

aber $O(n^2)$ ist die genauere Abschätzung der Größenordnung. Die O-Notation sollte aus der Vorlesung „Diskrete Mathematik“ oder einer anderen mathematischen Grundveranstaltung bekannt sein.

Die wichtigsten Ressourcen sind Platz (= Speicherplatz) und Zeit (= Rechenzeit). Seien S, T Funktionen von $\mathbb{N}_0$ nach $\mathbb{N}_0$ und M eine DTM.

Definition 1.1 (Platzschranke, Zeitschranke)

1. *M heißt $S(n)$-*platzbeschränkt*, wenn eine Rechnung von M auf einer Eingabe der Maximallänge n maximal $O(S(n))$ Bandzellen verbraucht.*

2. *M heißt $T(n)$-*zeitbeschränkt*, wenn eine Rechnung von M auf einer Eingabe der Maximallänge n maximal $O(T(n))$ Schritte dauert.*

Eine DTM ist um so „platz- bzw. zeiteffizienter“ je langsamer $S(n)$ bzw. $T(n)$ mit n wächst. Eine grobe Unterscheidung ist polynomielles versus exponentielles Wachstum.

Veranschaulichung: Seien c, k Konstanten. $T(n) = c \cdot n$ bedeutet, dass eine Verdoppelung der Eingabelänge die Zeitschranke verdoppelt. $T(n) = c \cdot n^2$ (bzw. $T(n) = c \cdot n^k$) bedeutet, dass eine Verdoppelung der Eingabelänge, die Zeitschranke vervierfacht (bzw. ver-2^k-facht). $T(n) = c \cdot 2^n$ bedeutet, dass die Zeitschranke sich schon bei Eingaben der Länge $n + 1$ auf $c \cdot 2^{n+1} = 2c2^n$ verdoppelt.

Obwohl auch Rechenzeiten der Form $T(n) = 1000000000 \cdot n$ oder $T(n) = n^{1000}$ im Grunde inakzeptabel sind, läßt sich dennoch sagen, dass exponentielles Wachstum $T(n) = c \cdot 2^n$ schon für moderate Werte von n auch schnellste Rechenanlagen für Milliarden von Jahren beschäftigt. Man denke etwa an die *Weizenkornlegende*[1]. Aus diesen Gründen hat sich die Sichtweise durchgesetzt, polynomielles Wachstum noch als „effizient“ zu betrachten, aber superpolynomielles oder gar eponentielles Wachstum für inpraktikabel zu halten.

Definition 1.2 (Deterministische Komplexitätsklassen)

1. *$DSpace(S)$ ist die Klasse aller Sprachen, die von einer $S(n)$-platzbeschränkten DTM akzeptiert werden können.*

2. *$DTime(T)$ ist die Klasse aller Sprachen, die von einer $T(n)$-zeitbeschränkten DTM akzeptiert werden können.*

[1] Der Legende nach gewährte der indische Herrscher Shiram (3./4. Jahrhundert n. Chr.) dem Erfinder des Schachspiels, Sissa ibn Dahir, einen freien Wunsch. Jener wünschte sich Weizenkörner: auf das 1. Feld eines Schachbrettes ein Korn, auf das 2. Feld die doppelte Menge, also zwei, auf das dritte wiederum dopelt so viele, also vier, und so weiter. Dies sieht auf den ersten (naiven) Blick nach einem bescheidenen Wunsch aus. In Wirklichkeit aber addieren sich die Weizenkörner zu der Zahl $2^{64} - 1 = 18.446.744.073.709.551.615$. Eine Wagenkolonne mit dieser Menge würde mehr als 230.000-mal um die Erde reichen.

3. *$P = \cup_{k\geq 1} DTime(n^k)$ ist die Klasse aller deterministisch in Polynomialzeit akzeptierbaren Sprachen.*

4. *$PSpace = \cup_{k\geq 1} DSpace(n^k)$ ist die Klasse aller deterministisch in polynomiellem Platz akzeptierbaren Sprachen.*

Notation: Ab jetzt notieren wir die von einer TM M akzeptierte Sprache als $L(M)$ (statt, wie früher, als $T(M)$), weil wir Symbol T für Zeitschranken reservieren möchten.

Definition 1.1 lässt sich auch auf NTM's übertragen. Eine NTM M heißt $S(n)$-platzbeschränkt (bzw. $T(n)$-zeitbeschränkt), wenn zu jeder Eingabe $x \in L(M)$ der Maximallänge n eine akzeptierende Rechnung existiert, die maximal $O(S(n))$ Bandzellen (bzw. $O(T(n))$ Rechenschritte) benötigt.

Definition 1.3 (Nichtdeterministische Komplexitätsklassen)

1. *$NSpace(S)$ ist die Klasse aller Sprachen, die von einer $S(n)$-platzbeschränkten NTM akzeptiert werden können.*

2. *$NTime(T)$ ist die Klasse aller Sprachen, die von einer $T(n)$-zeitbeschränkten NTM akzeptiert werden können.*

3. *$NP = \cup_{k\geq 1} NTime(n^k)$.*

Es hat sich gezeigt, dass

$$NSpace(S) \subseteq DSpace(S^2).$$

Für jede NTM existiert also eine deterministische Simulation mit höchstens quadratischem „blow-up“ des Speicherplatzes (Satz von Savitch). Daher erübrigt sich eine Unterscheidung von $DPSpace$ und $NPSpace$.
Zentrale Fragen der Komplexitätstheorie sind die folgenden:

1. Welche Probleme erfordern (im Wesentlichen) die gleichen Ressourcen an Zeit bzw. Platz und gehören daher in dieselbe Komplexitätsklasse ?

2. Wie ist das Verhältnis von Platz und Zeit? Kann man durch Mehraufwand an Speicherplatz Rechenzeit einsparen und umgekehrt?

3. Wie ist das Verhältnis von Determinismus und Nichtdeterminismus? Der Satz von Savitch garantiert platzeffiziente deterministische Simulationen von NTM's. Gibt es auch zeiteffiziente deterministische Simulationen?

Wir wollen der zweiten Frage anhand des berühmten ($P \stackrel{?}{\neq} NP$)-Problems nachgehen.[2] Viele fundamentale Anwendungsprobleme (s. Liste im Anhang) gehören zur Klasse NP. Diese

[2]Es handelt sich um eines der sieben „Millenium Prize Problems“ auf der Liste des „Clay Mathematics Institute“ (www.claymath.org).

Probleme können bis heute nicht (deterministisch) in Polynomialzeit gelöst werden. Die meisten ExpertInnen glauben daher

$$P \neq NP \ .$$

Man kann bis heute aber diese Vermutung nicht beweisen. Immerhin ist es aber im Rahmen der NP-Vollständigkeitstheorie gelungen, „härteste Vertreter“ der Problemklasse NP dingfest zu machen. Diese werden NP-vollständige Probleme genannt. Wir werden in Kürze zeigen, dass das Erfüllbarkeitsproblem der Aussagenlogik (Satisfiability-Problem oder kurz SAT) NP-vollständig ist. Falls ein deterministischer Polynomialzeitalgorithmus für ein NP-vollständiges Problem (wie zum Beispiel SAT) existieren würde, dann würde logisch zwingend P=NP folgen. Im Umkehrschluss impliziert die Annahme $P \neq NP$, dass es für NP-vollständige Probleme keine deterministischen Polynomialzeitalgorithmen gibt. Dies kann dann als Legitimation dienen, es mit Heuristiken[3] zu probieren.

Der Anhang enthält eine kleine Liste von NP-vollständigen Problemen. Ein wichtiges Werkzeug zur Entwicklung der Theorie sind die „polynomiellen Reduktionen“:

Definition 1.4 (Polynomielle Reduktion) *Seien $L_1, L_2 \subseteq \Sigma^*$.*

1. *Wir sagen, L_1 ist* polynomiell reduzierbar *auf L_2 (in Zeichen: $L_1 \leq_{pol} L_2$), wenn eine Abbildung $f : \Sigma^* \to \Sigma^*$ existiert, so dass folgendes gilt:*

   (1) $\forall w \in \Sigma^* : \ w \in L_1 \Leftrightarrow f(w) \in L_2$.

   (2) f ist von einer polynomiell zeitbeschränkten DTM berechenbar.

2. *Eine Sprache $L_0 \subseteq \Sigma^*$ heißt* NP-vollständig*, falls*

   (1) $L_0 \in NP$.

   (2) $\forall L \in NP : \ L \leq_{pol} L_0$.

   Falls die zweite Bedingung gilt, aber L_0 nicht notwendig zur Klasse NP gehört, dann heißt L_0 NP-hart*.*

Die folgenden Beobachtungen sind leicht zu beweisen.

Bemerkung 1.5 *1. Die Relation „$\leq_{pol}$“ ist transitiv. Ketten von Reduktionen ergeben also wieder eine Reduktion.*

2. *Aus $L_1 \leq_{pol} L_2$ und L_1 ist NP-hart folgt, dass auch L_2 NP-hart ist.*

3. *Aus $L \leq_{pol} L_0$ (via Reduktionsabbildung f) und $L_0 \in P$ folgt $L \in P$. Die Frage, ob $w \in L$, kann nämlich entschieden werden wie folgt:*

   (a) Berechne $f(w)$.

   (b) Entscheide, ob $f(w) \in L_0$.

[3] Algorithmen ohne allgemeine Erfolgsgarantie, die in der Praxis ganz gut zu funktionieren scheinen

4. *Wenn ein NP-hartes Problem (deterministisch) in Polynomialzeit gelöst werden kann, dann folgt P=NP.*

Die Liste NP-vollständiger Probleme im Anhang ist durch folgende Evolution entstanden:

1. Das Theorem von Cook — gewissermaßen der „Urknall“ der NP-Vollständigkeitstheorie — lieferte mit SAT das erste „natürliche“ NP-vollständige Problem.

2. Von SAT ausgehend hat sich mit polynomiellen Reduktionen mit der Zeit eine Art „Stammbaum“ NP-vollständiger Probleme gebildet. Ein Teil dieser Stammbaums ist in Abbildung 2 zu sehen. (Die Abkürzungen beziehen sich dabei auf die Problemliste im Anhang.)

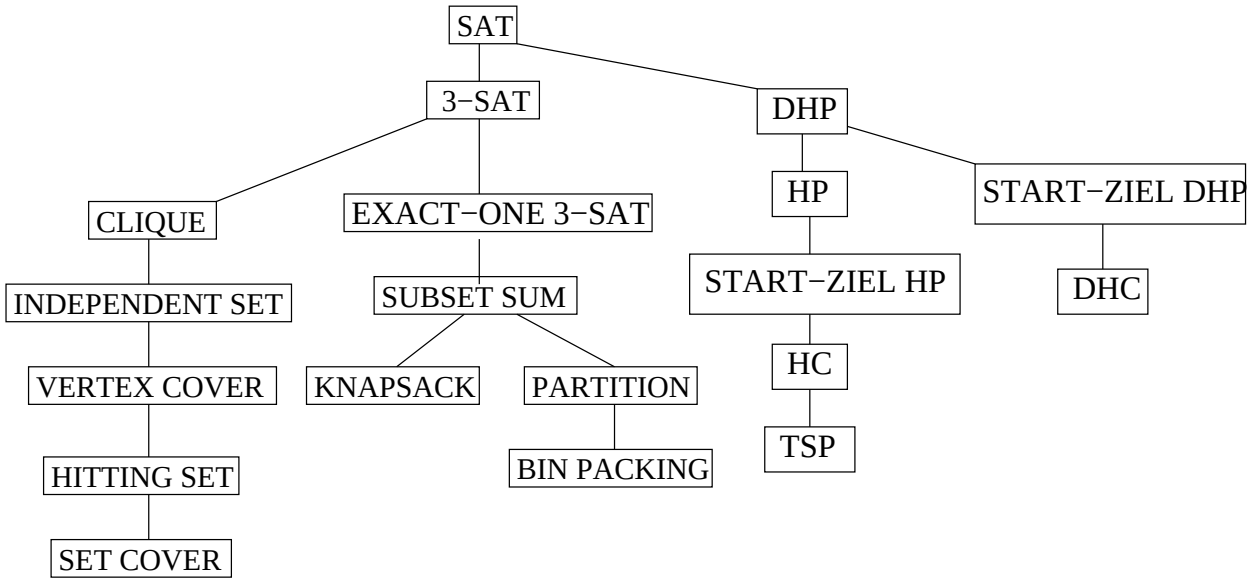

Abbildung 2: Stammbaum NP-vollständiger Probleme.

Eine lange Liste mit NP-vollständigen Problemen ist in dem Buch *Garey and Johnson, Computers and Intactability, A Guide to the Theory of NP-Completeness, Freeman and Company, 1979*, zu finden. Es enthält zudem eine großartige Beschreibung der Theorie der NP-Vollständigkeit.

SAT ist ein Problem im Zusammenhang mit Booleschen Formeln (die aus anderen Grundvorlesungen bekannt sein sollten). Ein *Literal* ist eine negierte oder nicht negierte Boolesche Variable. Eine *Klausel* ist eine Disjunktion (Logisches Oder) von Literalen, und eine *Formel in konjunktiver Normalform* (kurz: *CNF-Formel*) ist eine Konjunktion (Logisches Und) von Klauseln. Beachte, dass eine Klausel auch aus einem einzigen Literal und eine CNF-Formel auch aus einer einzigen Klausel bestehen darf.

Beispiel 1.6

$$\begin{aligned} F_0 &= (x_1 \vee x_3) \wedge (x_1 \vee \overline{x_3}) \wedge \overline{x_1} \\ F_1 &= (x_1 \vee \overline{x_2} \vee x_3) \wedge (\overline{x_1} \vee \overline{x_3}) \wedge x_2 \end{aligned}$$

sind CNF-Formeln.

SAT ist folgendes Problem:

Eingabe eine CNF-Formel F.

Frage Ist die Formel *erfüllbar*, d.h., existiert eine Belegung von $x_1, \ldots, x_n$ mit 0 oder 1, so dass F zu 1 ausgewertet wird?

Beispiel 1.7 (fortgesetzt) $x_1 = 1, x_2 = 1, x_3 = 0$ erfüllt die obige Formel F_1.
Die Formel F_0 hingegen ist nicht erfüllbar. Wieso ?

Satz 1.7 (Theorem von Cook) *SAT ist NP-vollständig.*

Beweis Der Beweis nutzt aus, dass (wie früher in der Vorlesung schon mal besprochen) eine NTM M mit Eingabewort w oBdA nach dem „Rate-Verifikationsprinzip“ arbeiten kann:

Ratephase: Rate *nichtdeterministisch* einen Bitstring r.

Verifikationsphase: Verifiziere *deterministisch* mit Hilfe von r, dass w das Zielkriterium der Sprache $L(M)$ erfüllt.

Dabei existiert genau für die Wörter aus $L(M)$ mindestens ein Ratestring, der M in der Verifikationsphase zum Akzeptieren bringt. Der Beweis des Theorems von Cook besteht nun aus zwei Teilen.

1. SAT $\in$ NP.
   Gegeben die CNF-Formel F, rate (nichtdeterministisch) die erfüllende Belegung r und werte danach F (deterministisch und in Polynomialzeit) aus, um die Erfüllbarkeit von F anhand von $F(r) = 1$ zu verifizieren.

2. Für alle $L \in$ NP: $L \leq_{\text{pol}}$ SAT.
   Sei $L \in$ NP und M eine polynomiell zeitbeschränkte NTM für L. Dann gilt für geeignete Konstanten c_1, c_2:

   - Sei $R = R(n) = n^{c_1}$. Bei Eingabe w der Länge n rät M zunächst einen Binärstring $r \in \{0,1\}^R$ und produziert die erweiterte Eingabe $r\$w$ mit Trennsymbol \$ in Zelle 0.
   - Gestartet auf Inschrift $r\$w$ (im Startzustand z_0 und Kopfposition 0) rechnet M deterministisch genau $T = T(n) = n^{c_2}$ Schritte.
   - M hat einen eindeutigen (akzeptierenden) Endzustand ACC.

Unser Ziel ist es, eine Formel $F = F_w$ in konjunktiver Normalform mit folgender Eigenschaft zu berechnen:

$$w \in L \Leftrightarrow F_w \text{ erfüllbar.}$$

Idee: F beschreibt die deterministische Rechnung von M auf $r\$w$ und ist erfüllbar genau dann, wenn diese Rechnung für mindestens einen Ratestring r akzeptierend ist.

Um die Rechnung von M zu beschreiben, benutzen wir die folgenden Booleschen Variablen:

Variable X	Interpretation von $X = 1$
$Z(i, z)$	Zustand z zum Zeitpunkt i
$H(i, j)$	Kopfposition j zum Zeitpunkt i
$S(i, j, c)$	Bandsymbol c in Zelle j zum Zeitpunkt i

Dabei gilt $0 \leq i \leq T, -T \leq j \leq T, z \in Z, c \in \Gamma$. Die Anzahl der Variablen bei Eingabelänge $|w| = n$ ist offensichtlich von der Größenordnung $O((T(n))^2)$.

Wir setzen verschiedene Typen von Klauseln ein.

Typ I: Korrektheit der Beschreibung einer Konfiguration.

Typ II: Korrekte Startkonfiguration und akzeptierende Endkonfiguration.

Typ III: Korrekte Überführung einer Konfiguration in die direkte Folgekonfiguration.

Dabei haben wir hinter jedem Typ die Aufgabe genannt, die die betreffenden Klauseln erfüllen müssen. Es folgt eine detaillierte Beschreibung aller Klauseln. Die Klauseln vom Typ I sollen folgendes erzwingen:

(1) Für alle i existiert genau ein z mit $Z(i, z) = 1$.

Interpretation: Zu jedem Zeitpunkt befindet sich M in genau einem Zustand.

(2) Für alle i existiert genau ein j mit $H(i, j) = 1$.

Interpretation: Zu jedem Zeitpunkt gibt es genau eine Position, auf der sich M's Kopf befindet.

(3) Für alle i, j existiert genau ein c mit $S(i, j, c) = 1$.

Interpretation: Zu jedem Zeitpunkt speichert jede Bandzelle genau ein Symbol.

In allen drei Fällen haben wir das gleiche Grundmuster:

Genau ein $y \in \{y_1, \ldots, y_m\}$ hat Wert 1. Dies ist äquivalent zu folgenden Klauseln

$$(y_1 \vee \ldots \vee y_m) \wedge \bigwedge_{1 \leq i < j \leq m} (\overline{y_i} \vee \overline{y_j}),$$

$1 + \binom{m}{2} = O(m^2)$ an der Zahl.

Wenn wir dieses Grundmuster auf (1), (2) und (3) anwenden, so benötigen wir insgesamt $O((T(n))^3)$ Klauseln.

Wir kommen zu den Klauseln vom Typ II (von denen viele aus einem einzigen Literal bestehen). Die Klausel

(4) $Z(T, \mathrm{ACC})$

kontrolliert, dass M am Ende der Rechnung sich im (akzeptierenden) Endzustand befindet. Die Klauseln

(5) $Z(0, z_0)$

(6) $H(0, 0)$

(7)
$S(0, -j, 0) \vee S(0, -j, 1)$ für $j = 1, \ldots, R$ (bel. Ratestring)
$S(0, 0, \$)$ (Trennsymbol)
$S(0, j, w_j)$ für $1 \leq j \leq n$ (Eingabe)
$S(0, j, \Box)$ für $-T \leq j < -R, n < j \leq T$ (Blanks)

kontrollieren, dass M im Zustand z_0 mit Kopf in Position 0 und Bandinschrift $r\$w$ (eingerahmt von Blanks im relevanten Speicherbereich) startet. (4), ..., (7) zusammengenommen sind nur $O(T(n))$ viele Klauseln.

Die Klauseln vom Typ III sollen folgendes erzwingen:

(8) Die nicht gelesenen Speicherzellen haben einen unveränderten Inhalt:

$$\overline{H(i, j)} \wedge S(i, j, c) \Rightarrow S(i + 1, j, c).$$

Die Schreibweise mit Implikation „$\Rightarrow$“ ist wegen

$$(y_1 \wedge \ldots \wedge y_m \Rightarrow y) \Leftrightarrow (\overline{y_1} \vee \ldots \vee \overline{y_m} \vee y)$$

leicht in Klauselschreibweise transformierbar. Dies machen wir uns auch im folgenden zunutze.

(9) Die eintretenden Veränderungen müssen der Überführung $\delta(z, c) = (z', c', d)$ mit $d \in \{L, R, N\}$ entsprechen:

$Z(i, z) \wedge H(i, j) \wedge S(i, j, c) \Rightarrow$
(a) $Z(i + 1, z')$
(b) $S(i + 1, j, c')$
(c) $H(i + 1, j + s(d))$ mit $s(d) = \begin{cases} -1, \text{ falls } d = L, \\ 0\ , \text{ falls } d = N, \\ 1\ , \text{ falls } d = R. \end{cases}$

(8), (9) zusammengenommen sind $O((T(n))^2)$ weitere Klauseln. Die Formel F_w besteht nun aus den in (1), ..., (9) genannten Klauseln, $O((T(n))^3)$ an der Zahl. Es ist leicht einzusehen, dass F_w deterministisch und in Polynomialzeit aus w konstruiert werden kann. Das Cooksche Theorem ergibt sich dann direkt aus dem Beweis von:

$$w \in L \overset{!}{\Leftrightarrow} F_w \text{ erfüllbar.}$$

$\Rightarrow$: Falls $w \in L$, dann besitzt M für einen geeigneten Ratestring $r \in \{0,1\}^R$ eine akzeptierende Rechnung auf w. Der Ratestring r und die Rechnung von M auf $r\$w$ liefern eine erfüllende Belegung für F_w (vgl. die Interpretation der Variablen).

$\Leftarrow$: Aus der Belegung von $S(0,-R,b),\ldots,S(0,-1,b)$ mit $b = 0,1$ läßt sich der Ratestring r ablesen, für den M auf $r\$w$ eine akzeptierende Rechnung vollzieht. **qed**

2 Grundlegende NP–vollständige Probleme

Ausgehend von dem Erfüllbarkeitsproblem der Aussagenlogik (SAT) werden wir in diesem Abschnitt mit Hilfe geeigneter polynomieller Reduktionen die *NP*-Vollständigkeit einiger grundlegender Probleme beweisen. Im Wesentlichen verifizieren wir den in Abbildung 2 gezeigten Ausschnitt des Stammbaums *NP*-vollständiger Probleme. Da die Mitgliedschaft dieser Probleme zur Klasse *NP* jeweils sehr leicht nachzuweisen ist, konzentrieren wir uns auf den Nachweis der *NP*-Härte.

Eine *k-Klausel* ist eine Boolesche Klausel, die genau k paarweise verschiedene Literale enthält. *k-SAT* ist das Teilproblem von SAT, bei welchem als Eingabe nur Kollektionen von k-Klauseln zugelassen sind. Um uns den Entwurf der weiteren polynomiellen Reduktionen zu erleichtern, zeigen wir zunächst, dass sogar 3-SAT *NP*-vollständig ist. Am Rande sei bemerkt, dass 2-SAT zur Klasse P gehört.

Satz 2.1 *SAT* $\leq_{pol}$ 3-*SAT.*

Beweis Die Reduktion von SAT auf 3-SAT benutzt die Methode der *lokalen Ersetzung.* Dabei wird jede Klausel

$$C = z_1 \vee \ldots \vee z_k \text{ mit } z_1,\ldots,z_k \in \{x_1,\bar{x}_1,\ldots,x_n,\bar{x}_n\}$$

durch eine Kollektion K_C von 3-Klauseln ersetzt, die in folgendem Sinne zu C äquivalent ist:

(A) Jede C erfüllende Belegung führt zu einer erfüllenden Belegung für alle 3-Klauseln in K_C.

(B) Jede Belegung, welche alle 3-Klauseln in K_C erfüllt, muss auch eines der Literale in C erfüllen.

Wir beschreiben die Abbildung $C \mapsto K_C$ im Rahmen einer Fallunterscheidung.

Fall 1: $k = 3$.

Glück gehabt: C ist bereits eine 3-Klausel.

Fall 2: $k < 3$.

C ist also zu kurz und muss um weitere Literale „ausgepolstert“ werden. Im Falle $C = z_1$ benutzen wir zwei neue (also nur in K_C vorkommende) Hilfsvariable a, b und K_C enthalte die 3-Klauseln

$$z_1 \vee a \vee b \ , \ z_1 \vee a \vee \bar{b} \ , \ z_1 \vee \bar{a} \vee b \ , \ z_1 \vee \bar{a} \vee \bar{b} \ .$$

Im Falle $C = z_1 \vee z_2$ benutzen wir eine neue Hilfsvariable a und K_C enthalte die 3-Klauseln

$$z_1 \vee z_2 \vee a \ , \ z_1 \vee z_2 \vee \bar{a} \ .$$

Man überlegt sich leicht, dass die Bedingungen (A) und (B) erfüllt sind.

Fall 3: $k > 3$.

C ist also zu lang und muss in 3-Klauseln aufgesplittert werden. Wir demonstrieren die allgemeine Konstruktion am Beispiel $k = 7$. K_C verwendet die neuen Hilfsvariablen $h_2, \ldots, h_{k-2}$ und enthält die folgenden 3-Klauseln:

$$z_1 \vee z_2 \vee h_2, \ \bar{h}_2 \vee z_3 \vee h_3, \bar{h}_3 \vee z_4 \vee h_4, \ \bar{h}_4 \vee z_5 \vee h_5, \ \bar{h}_5 \vee z_6 \vee z_7.$$

Für allgemeines k hat die letzte dieser 3-Klauseln die Form $\bar{h}_{k-2} \vee z_{k-1} \vee z_k$.
Wir verifizieren Bedingung (A). Sei eine Belegung von $x_1, \ldots, x_n$ gegeben, die C mit Hilfe des Literals z_j erfüllt. Wir erweitern diese Belegung auf die Hilfsvariablen, indem wir $h_j, h_{j+1}, \ldots$ mit Null belegen und $h_{j-1}, h_{j-2}, \ldots$ mit Eins. Es ist leicht zu sehen, dass damit alle 3-Klauseln aus K_C erfüllt sind.
Wir verifizieren Bedingung (B). Es genügt zu zeigen, dass eine C nicht erfüllende Belegung der Variablen $x_1, \ldots, x_n$ nicht zu einer K_C erfüllenden Belegung fortgesetzt werden kann. Da C nicht erfüllt ist, ist keines der Literale $z_1, \ldots, z_k$ erfüllt. Was die Belegung der Hilfsvariablen betrifft, argumentieren wir mit der Logik des *Zugzwanges*. Um $z_1 \vee z_2 \vee h_2$ zu erfüllen, **müssen** wir $h_2 = 1$ setzen. Um $\bar{h}_2 \vee z_3 \vee h_3$ zu erfüllen, **müssen** wir $h_3 = 1$ setzen. Iterative Anwendung dieses Argumentes führt zu dem Zwang auch $h_4, \ldots, h_{k-2}$ auf Eins zu setzen. Dann ist aber die letzte 3-Klausel $\bar{h}_{k-2} \vee z_{k-1} \vee z_k$ — im Beispiel $\bar{h}_5 \vee z_6 \vee z_7$ — nicht erfüllt. **Schachmatt!**

Wenn wir die Substitution $C \mapsto K_C$ auf jede Klausel einer gegebenen CNF-Formel F anwenden, erhalten wir eine Kollektion von 3-Klauseln, die erfüllbar ist genau dann wenn F erfüllbar ist. **qed**

Beim Problem EXACT-ONE 3-SAT wird danach gefragt, ob zu einer gegebenen Kollektion von 3-Klauseln eine Belegung existiert, welche in jeder Klausel exakt ein Literal erfüllt.

Satz 2.2 *3-SAT* $\leq_{pol}$ *EXACT-ONE 3-SAT.*

Beweis Bevor wir mit dem Beweis im engeren Sinne beginnen, machen wir uns mit der „Logik“ von EXACT-ONE 3-SAT vertraut:

- Wenn wir in den Klauseln

$$g_0 \vee g \vee g' \text{ und } g_0 \vee \bar{g} \vee \overline{g'} \tag{2}$$

  jeweils exakt ein Literal erfüllen wollen, dann sind genau folgende Belegungen möglich:

$$g_0 = 0 \ , \ g = 0 \ , \ g' = 1 \ , \quad \text{oder } g_0 = 0, \ g = 1 \ , \ g' = 0$$

  In jedem Fall *muss* g_0 auf Null gesetzt werden, was wir im Folgenden auch voraussetzen.

- Wenn wir in den Klauseln

$$a \vee h \vee g_0 \ , \ b \vee h \vee h' \tag{3}$$

  jeweils exakt ein Literal erfüllen wollen, dann sind (unter Verwendung von $g_0 = 0$) genau folgende Belegungen möglich:

$$\begin{aligned} a = 0 \ , \ h = 1 \ , b = h' = 0 \quad & \text{oder } a = 1 \ , \ h = 0 \ , \ b = 0 \ , \ h' = 1 \\ & \text{oder } a = 1 \ , \ h = 0 \ , \ b = 1 \ , \ h' = 0 \end{aligned}$$

  Insbesondere gilt: Wenn a auf Null gesetzt wird, dann muss b auch auf Null gesetzt werden. Wird aber a auf 1 gesetzt, dann sind für b beide Belegungen möglich.

Wir bezeichnen die 3-Klauseln in (3) als „(a, b)-Spielzeug“. Im Beweis weiter unten werden wir das (a, b)-Spielzeug mehrfach (mit verschiedenen Literalpaaren in der Rolle von (a, b)) einsetzen. Jedes Spielzeug benutzt „private“ Hilfsvariable h, h', die es nicht mit anderen Spielzeugen teilt. Die Variable g_0 hingegen ist global und wird (in der Bedeutung von Null bzw. FALSE) in allen Spielzeugen verwendet.
Nun zum eigentlichen Beweis! Unsere Aufgabe besteht darin, eine gegebene Eingabeinstanz $F = (C_1, \ldots, C_m)$ von 3-SAT effizient in eine Eingabeinstanz F' von EXACT-ONE 3-SAT zu transformieren, so dass F erfüllbar ist genau dann, wenn man in jeder Klausel aus F' exakt 1 Literal erfüllen kann. Zu diesem Zweck nehmen wir die folgenden 3-Klauseln in F' auf:

- Die 3-Klauseln aus (2): dadurch verfügen wir über die „globale Variable“ g_0 in der Bedeutung FALSE.

- Erstens für jede 3-Klausel

$$C_i = z_{i,1} \vee z_{i,2} \vee z_{i,3} \ , \ z_{i,1}, z_{i,2}, z_{i,3} \in \{x_1, \bar{x}_1, \ldots, x_n, \bar{x}_n\}$$

  die 3-Klausel

$$C_i' = h_{i,1} \vee h_{i,2} \vee h_{i,3}$$

  mit neuen (also nur in C_i' vorkommenden) Hilfsvariablen, und zweitens, für jedes $j \in \{1, 2, 3\}$, alle Klauseln eines $(z_{i,j}, h_{i,j})$-Spielzeuges, wodurch folgendes erreicht wird:

(A) Wenn $z_{i,j}$ auf Null gesetzt wird, dann muss $h_{i,j}$ auch auf Null gesetzt werden. Im Umkehrschluss heißt das natürlich, dass mit $h_{i,j} = 1$ auch $z_{i,j}$ auf Eins gesetzt werden muss.

(B) Wenn $z_{i,j}$ auf Eins gesetzt wird, dann sind für $h_{i,j}$ beide Belegungen, Null und Eins, möglich.

Wir argumentieren nun, dass F erfüllbar ist genau dann, wenn eine Belegung existiert, die in jeder der 3-Klauseln C_i' exakt ein Literal erfüllt:

- Es sei eine F erfüllende Belegung gegeben. Wir wählen in jeder Klausel C_i genau ein erfülltes Literal $z_{i,j(i)}$ aus, setzen $h_{i,j(i)}$ auf 1 und die weiteren Hilfsvariablen in C_i' auf 0. Wegen der Bedingungen (A),(B) ist dies konsistent zu den Klauseln in den diversen Spielzeugen.

- Es sei eine Belegung gegeben, die in jeder der Klauseln C_i' genau ein Literal $h_{i,j(i)}$ erfüllt. Aus (A) folgt, dass dann auch $z_{i,j}$ erfüllt ist, d.h., die gegebene Belegung erfüllt F.

Somit liefert $F \mapsto F'$ eine passende Reduktionsabbildung. **qed**

Die Reduktion im Beweis benutzte erneut die Methode der lokalen Ersetzung, da jede 3-Klausel C_i in F durch eine Kollektion von 3-Klauseln ersetzt wird und diese Substitutionen unabhängig voneinander durchgeführt werden können. Daneben gibt es aber noch eine „Ergänzungskomponente“, die aus den 3-Klauseln in (2) besteht. Die Ergänzungskomponente hat die Aufgabe, eine Variable g_0 mit dem (erzwungenen) Wert 0 zur Verfügung zu stellen.

Die polynomiellen Reduktionen von SAT auf 3-SAT und von 3-SAT auf EXACT-ONE 3-SAT verlaufen im Bereich der Booleschen Logik. Die folgende polynomielle Reduktion führt von einem Problem der Booleschen Logik zu dem graphentheoretischen Problem CLIQUE: zu einem gegebenen Graphen $G = (V, E)$ und einer gegebenen Zahl k soll entschieden werden, ob G k paarweise miteinander verbundene Knoten (eine sogenannte „k-Clique“) enthält.

Satz 2.3 *3-SAT* $\leq_{pol}$ *CLIQUE.*

Beweis Sei $C = (C_1, \ldots, C_m)$ mit $C_i = z_{i1} \vee z_{i2} \vee z_{i3}$ und $z_{ij} \in \{x_1, \bar{x}_1, \ldots, x_n, \bar{x}_n\}$ eine Eingabe für 3-SAT. Bevor wir die Reduktionsabbildung beschreiben, welche C auf eine korrespondierende Eingabe für CLIQUE abbildet, führen wir eine vorbereitende Überlegung durch:
Zwei Literale z, z' heißen *kompatibel*, falls $z' \neq \bar{z}$. Mehrere Literale $z_1, \ldots, z_r$, $r \geq 2$, heißen *kompatibel*, wenn sie paarweise kompatibel sind. Die folgende Beobachtung enthält den Schlüssel zur Wahl der Reduktionsabbildung.

Beobachtung: Es gibt genau dann eine Belegung, die $z_1, \ldots, z_r$ erfüllt, wenn $z_1, \ldots, z_r$ kompatibel sind.

Die zum Klauselsystem C korrespondierende Eingabe (G, k) mit $G = (V, E)$ für CLIQUE soll folgendermaßen aussehen. V enthält $3m$ Knoten $(i, j), 1 \leq i \leq m, 1 \leq j \leq 3$, die die Literale in den Klauseln darstellen. E enthält die Kante zwischen (i, j) und (i', j'), wenn $i \neq i'$ ist (es handelt sich um Knoten aus verschiedenen Klauseln) und $z_{ij} \neq \bar{z}_{i'j'}$ ist (die betreffenden Literale sind kompatibel). Schließlich sei $k = m$. Natürlich ist die Reduktionsabbildung $C \mapsto (G, m)$ in polynomieller Zeit berechenbar.

Bleibt zu zeigen, dass C genau dann erfüllbar ist, wenn G eine Clique der Größe m (genannt m-Clique) enthält.
Gehen wir aus von einer Belegung, die alle Klauseln erfüllt. Dann ist in jeder Klausel C_i mindestens ein Literal, sagen wir $z_{i,j(i)}$, erfüllt. Folglich sind die Literale $z_{i,j(i)}$, $1 \leq i \leq m$, kompatibel. Dann bilden die Knoten $(i, j(i))$, $1 \leq i \leq m$, in G eine m-Clique.
Nehmen wir umgekehrt an, dass G eine m-Clique enthält. Der Konstruktion von G entnehmen wir, dass die Knoten der Clique Literale aus verschiedenen Klauseln repräsentieren und die repräsentierten Literale, sagen wir $z_{i,j(i)}$ mit $1 \leq i \leq m$, kompatibel sind. Folglich gibt es eine Belegung, die diese Literale und somit auch die Klauseln $C_1, \ldots, C_m$ erfüllt. **qed**

Die in diesem Beweis verwendete Reduktion ist eine sogenannte Reduktionsabbildung mit *verbundenen Komponenten*, da wir zunächst Komponenten für die einzelnen Klauseln bilden, diese aber durch die Kanten des Graphen verbinden.

Die polynomielle Reduktionskette

$$\begin{aligned} \text{CLIQUE} &\leq_{pol} \text{INDEPENDENT SET} \\ &\leq_{pol} \text{VERTEX COVER} \\ &\leq_{pol} \text{HITTING SET} \\ &\leq_{pol} \text{SET COVER} \end{aligned}$$

empfehlen wir als Übung.

SUBSET SUM ist das Problem, zu gegebenen Zahlen $a_1, \ldots, a_n \in \mathbb{N}_0$ und einer Summenzahl $S \in \mathbb{N}_0$ zu entscheiden, ob eine Auswahl $J \subseteq \{1, \ldots, n\}$ mit $\sum_{j \in J} a_j = S$ existiert.

Satz 2.4 *EXACT-ONE 3-SAT* $\leq_{pol}$ *SUBSET SUM.*

Beweis Es sei $F = (C_1, \ldots, C_m)$ mit 3-Klauseln C_i sei eine Eingabeinstanz von EXACT-ONE 3-SAT. Wir nummerieren die in den Klauseln vorkommenden Literale von 1 bis $2n$:

$$z_1 = x_1, \ldots, z_n = x_n \ , \ z_{n+1} = \bar{z}_1, \ldots, z_{2n} = \bar{x}_n$$

Für jedes Literal z_j definieren wir

$$S_j := \{i \in \{1, \ldots, m\} |\ z_j \text{ ist Literal in } C_i\} \ .$$

Wir haben die Aufgabe, F effizient in eine Eingabeinstanz E von SUBSET SUM zu transformieren, so dass sich in F jede Klausel exakt einmal erfüllen lässt genau dann, wenn eine geeignete Auswahl der Zahlen in E sich zu der geforderten Summenzahl S aufaddiert. E wird gebildet wie folgt:

- Den Index j' identifizieren wir mit j, falls $1 \leq j \leq n$ und mit $j-n$, falls $n+1 \leq j \leq 2n$.
- Für jedes Literal z_j nehmen wir die Zahl

$$a_j := \sum_{i \in S_j} (2n)^i + (2n)^{m+j'}$$

in E auf.

- Die Summenzahl definieren wir durch

$$S := \sum_{i=1}^{m} (2n)^i + \sum_{j=1}^{n} (2n)^{m+j} \ .$$

Beachte, dass S für jede Klausel C_i exakt einen Term, nämlich $(2n)^i$, und für jedes Literalpaar $(x_j, \bar{x}_j) = (z_j, z_{m+j})$ exakt einen Term, nämlich $(2n)^{m+j}$, enthält.

Wir verifizieren im Folgenden, dass es sich um eine erfolgreiche Reduktionsabbildung handelt.

Erfolgreiche Belegung impliziert erfolgreiche Summenbildung: Gegeben sei eine 0, 1-Belegung der Variablen $x_1, \ldots, x_n$ (und somit auch der Literale $z_1, \ldots, z_{2n}$), welche in jeder Klausel C_i exakt ein Literal erfüllt. Dann definieren wir

$$J := \{j \in \{1, \ldots, 2n\} |\ z_j = 1\} \ .$$

In die Summe $\sum_{j \in J} a_j$ wird offensichtlich jeder Term der Form $(2n)^i$ genau einmal aufgenommen (da C_i von genau einem Literal erfüllt wird). Eine analoge Bemerkung gilt für jeden Term der Form $(2n)^{m+j}$, (da entweder$z_j = x_j = 1$ oder $z_{m+j} = \bar{x}_j = 1$). Wir erhalten exakt die Summenzahl S.

Erfolgreiche Summenbildung impliziert erfolgreiche Belegung: Umgekehrt können wir aus einer Auswahl $J \subseteq \{1, \ldots, 2n\}$ mit $\sum_{j \in J} a_j = S$ eine Belegung ablesen, welche in jeder Klausel C_i exakt ein Literal erfüllt. J muss nämlich für jedes $j \in \{1, \ldots, n\}$ entweder j oder $m+j$ enthalten (damit der Term $(2n)^{m+j}$ genau einmal in die Summe eingeht). Im ersten Fall setzen wir $x_j = 1$, im zweiten Fall setzen wir $\bar{x}_j = 1$ (also $x_j = 0$). Da in die Summe bei erfolgreicher Summenbildung jeder Term der Form $(2n)^i$ exakt einmal eingeht, erfüllt diese Belegung jede Klausel exakt einmal.

qed

Der Beweis benutzte lokale Ersetzung mit Ergänzungskomponente: jedes Literal wird durch eine Zahl ersetzt; die Summenzahl ist die Ergänzungskomponente.

Im Königreich der Zahlenprobleme angekommen, lassen sich weitere Reduktionen relativ leicht finden.

Satz 2.5 *SUBSET SUM* $\leq_{pol}$ *KNAPSACK.*

Beweis Obwohl der Beweis trivial ist führen wir ihn aus didaktischen Gründen zweimal.

Es folgt Beweis Nummer 1. Es sei $\mathcal{I} = (a_1, \ldots, a_n, S)$ eine Eingabe für SUBSET SUM. Die Frage ist also, ob eine Auswahl $I \subseteq \{1, \ldots, n\}$ mit $\sum_{i \in I} a_i = S$ existiert. Die korrespondierende Eingabe $\mathcal{I}'$ für KNAPSACK sei gegeben durch die Gewichte $a_1, \ldots, a_n$ mit Gewichtsschranke S sowie die Nutzenwerte $a_1, \ldots, a_n$ mit Nutzenschranke S. Die Frage ist also, ob eine Auswahl $I \subseteq \{1, \ldots, n\}$ mit $\sum_{i \in I} a_i \leq S$ und $\sum_{i \in I} a_i \geq S$ existiert. Die erste Ungleichung drückt aus, dass das Gesamtgewicht des (imaginierten) Rucksackes die Gewichtsschranke S nicht überschreitet. Die zweite Ungleichung drückt aus, dass die Nutzenschranke S nicht unterschritten wird. Offensichtlich ist die Reduktionsabbildung $\mathcal{I} \mapsto \mathcal{I}'$ in Polynomialzeit berechenbar und es gilt $\mathcal{I} \in$ SUBSET SUM $\Leftrightarrow \mathcal{I}' \in$ *KNAPSACK*.

Es folgt Beweis Nummer 2. SUBSET SUM ist der Spezialfall von KNAPSACK, bei dem die Nutzenwerte mit den Gewichtswerten und die Nutzenschranke mit der Gewichtsschranke übereinstimmt. **Fertig!** **qed**

Wir lernen hier eine neue Reduktionsmethode kennen, die im Vergleich zu *lokaler Ersetzung* oder gar *verbundenen Komponenten* sympathisch einfach ist: *Spezialisierung*. Darunter verstehen wir eine Reduktion von L_1 nach L_2, die dokumentiert, dass eine spezielle Wahl der Eingabeparameter von L_2 ein Teilproblem erzeugt, dass „isomorph“ zu L_1 ist. Der Begriff „Isomorphie“ sollte hier nicht formal-mathematisch aufgefasst werden. Es ist vielmehr gemeint, dass man durch „scharfes Hinsehen“ erkennt, dass die spezielle Wahl der Eingabe für L_2 zu erkennen gibt, dass L_1 ein Teilproblem von L_2 ist. Da ein Teilproblem immer auf triviale Weise auf das Gesamtproblem polynomiell reduzierbar ist, kann man sich die wortreiche Beschreibung der Reduktionsabbildung sparen.

Damit wir nicht aus der Übung kommen, hier eine weitere Reduktion mit *lokaler Ersetzung* plus *Ergänzungskomponente*.

Satz 2.6 *SUBSET SUM* $\leq_{pol}$ *PARTITION.*

Beweis Schade, dass wir nicht PARTITION auf SUBSET SUM polynomiell reduzieren sollen! Dann könnten wir nämlich demonstrieren, dass wir das Prinzip der *Spezialisierung* verstanden haben. PARTITION ist nämlich der Spezialfall von SUBSET SUM mit $S = (a_1 + \cdots + a_n)/2$.

Okay, okay, das ist die falsche Richtung. Um SUBSET SUM auf PARTITION zu reduzieren, müssen wir irgendwie erzwingen, dass die zu bildende Teilsumme genau die Hälfte der Gesamtsumme ist. Sei also $\mathcal{I} = (a_1, \ldots, a_n, S)$ die Eingabe für SUBSET SUM. Sei $A = a_1 + \cdots + a_n$. Die korrespondierende Eingabe für PARTITION sei $\mathcal{I}' = (a_1, \ldots, a_n, S+1, A-S+1)$. Die Zahlen $S+1$ und $A-S+1$ sind dabei die angekündigten Ergänzungskomponenten. Im folgenden sei $A' = A + (S+1) + (A-S+1) = 2A+2$ die Gesamtsumme aller Zahlen in $\mathcal{I}'$. Die Reduktionsabbildung $\mathcal{I} \mapsto \mathcal{I}'$ ist offensichtlich in Polynomialzeit berechenbar. Es genügt also die folgende Aussage zu beweisen:

Behauptung: Es gibt genau dann eine Auswahl $I \subseteq \{1, \ldots, n\}$ mit $\sum_{i \in I} a_i = S$, wenn sich die Zahlen aus $\mathcal{I}'$ in zwei Mengen mit gleichen Teilsummen $A'/2 = A+1$ zerlegen lassen.

Sei zunächst I mit $\sum_{i\in I} a_i = S$ vorgegeben. Dann zerlegen wir die Zahlen aus $\mathcal{I}'$ in die Teilmengen

$$M_1 = \{a_i |\ i \in I\} \cup \{A - S + 1\} \text{ und } M_2 = \{a_j |\ j \notin I\} \cup \{S + 1\}.$$

M_1 liefert die Teilsumme $S+(A-S+1) = A+1$, und M_2 liefert die Teilsumme $(A-S)+(S+1) = A+1$. Beachte, dass die Ergänzungskomponenten gerade so gewählt waren, dass die Teilsummen S und $A-S$, die von SUBSET SUM herrührten, zu zwei gleichen Teilsummen ergänzt werden. Gute Sache diese Ergänzungskomponenten !

Nehmen wir nun umgekehrt an, dass eine Zerlegung der Zahlen von $\mathcal{I}'$ in zwei Mengen M_1, M_2 mit gleichen Teilsummen $A+1$ vorgegeben ist. Addition der beiden Ergänzungskomponenten liefert $(S+1)+(A-S+1) = A+2$. Diese beiden Zahlen können also unmöglich in der gleichen Menge M_i stecken. Also gilt oBdA $A-S+1 \in M_1$ und $S+1 \in M_2$. Da M_1 (wie auch M_2) zur Teilsumme $A+1$ führt, müssen die Zahlen aus $M_1 \cap \{a_1, \ldots, a_n\}$ sich zu S addieren. Voilà, $I = \{i |\ a_i \in M_1\}$ ist eine Lösung für die Eingabe von SUBSET SUM. **qed**

Als nächstes Problem gliedern wir BP in unseren Stammbaum ein.

Satz 2.7 *PARTITION$\leq_{pol}$BP.*

Beweis Entzückend! Jetzt kommen wir endlich dazu, die Kunst der *Spezialisierung* zu demonstrieren. PARTITION ist nämlich der Spezialfall von BIN PACKING, bei welchem wir n Objekte der Größen $a_1, \ldots, a_n$ in **zwei** Behälter der **speziellen** Behältergröße $(a_1+\cdots+a_n)/2$ verpacken sollen (Spezialisierungen durch Fettdruck hervorgehoben). (Kein Wort mehr, sonst ist die Eleganz des Beweises dahin.) **qed**

In unserem angekündigten Stammbaum fehlen noch die Probleme DHP, START-ZIEL DHP, DHC, HP, START-ZIEL HP, HC und TSP. (Zur Definition s. den Anhang.) Auf geht's!

Satz 2.8 *SAT$\leq_{pol}$DHP.*

Beweis Wir haben die Aufgabe, zu einer gegebenen Eingabeinstanz F von SAT, bestehend aus Klauseln $C_1, \ldots, C_m$ mit Literalen aus der Menge $\{x_1, \bar{x}_1, \ldots, x_n, \bar{x}_n\}$, einen gerichteten Graphen $G = (V, E)$ zu konstruieren, so dass F erfüllbar ist genau dann, wenn G einen gerichteten Hamiltonschen Pfad besitzt. Wir benutzen die Methode der verbundenen Komponenten. Für jede Variable x_j gibt es in G eine Komponente, genannt die „x_j-Kette“. Wenn x_j (negiert oder unnegiert) in m_j Klauseln vorkommt, dann ist dies eine doppelt verlinkte Kette der Länge m_j. Dies ist in Abbildung 3 illustriert. Jede Klausel C_i ist durch einen Knoten repräsentiert. Die Verbindungsstruktur zwischen den Klauselknoten und einer x_j-Kette ist ebenfalls in Abbildung 3 illustriert: für jedes negierte Vorkommen von x_j in einer Klausel C_i gibt es eine von rechts nach links führende Abzweigung aus der x_j-Kette nach C_i und wieder zurück. Für jedes unnegierte Vorkommen von x_j in einer Klausel gibt es eine entsprechende von links nach rechts führende Abzweigung. Abbildung 3 zeigt beispielhaft die Abzweigungen von der x_j-Kette zu den Klauselknoten C_3, C_5 und C_9, wobei unterstellt wird,

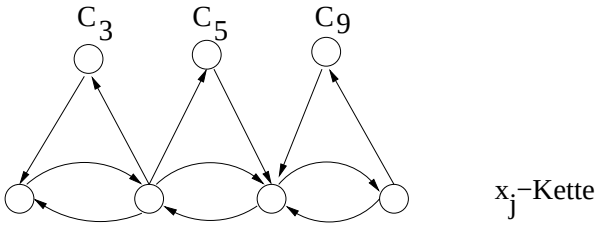

Abbildung 3: Eine x_j-Kette und die Verbindungen der x_j-Kette zu den Klauseln, in denen x_j vorkommt (im gezeigten Beispiel die Klauseln C_3, C_5, C_9).

dass x_j in C_3 und C_9 negiert und in C_5 unnnegiert vorkommt. Neben den bereits erwähnten Knoten in G gibt es einen ausgezeichneten Startknoten s und einen ausgezeichneten Zielknoten t. Abbildung 4 illustriert wie die x_j-Ketten untereinander und mit s und t verlinkt sind. Die einzige Chance, G mit einem Hamiltonschen Pfad zu durchlaufen, besteht offensichtlich

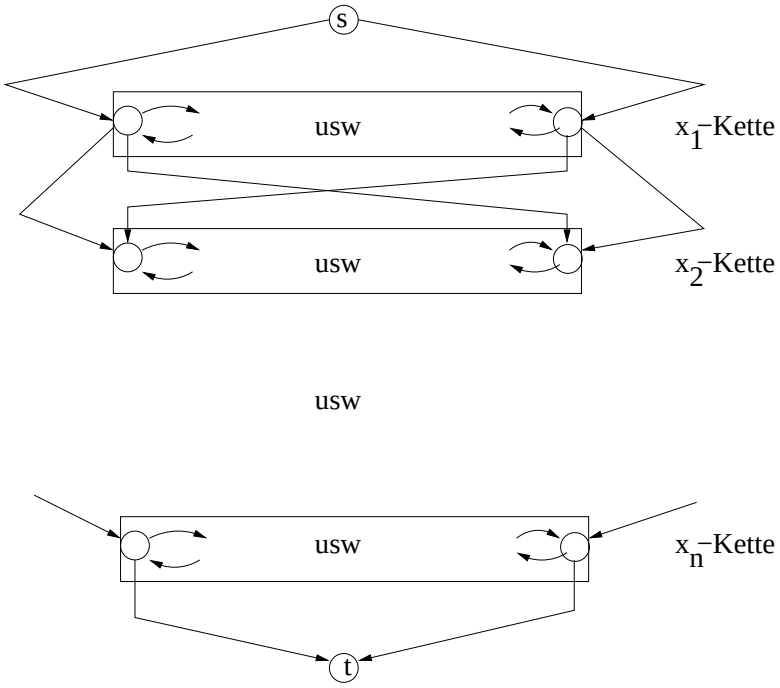

Abbildung 4: Die Verlinkung der x_j-Ketten untereinander und mit den Knoten s und t.

darin

- in s zu starten,
- die x_j-Ketten in der Reihenfolge $j = 1, \ldots, n$ zu durchlaufen, wobei man jedesmal die freie Wahl zwischen der Durchlaufrichtung „links-rechts“ und „rechts-links“ hat,

- unterwegs jeden Klauselknoten C_i genau einmal zu besuchen (durch Wahl einer geeigneten Abzweigung aus einer Kette),
- und schließlich in t zu enden.

Wenn wir die x_j-Kette von rechts nach links (bzw. von links nach rechts) durchlaufen, können wir zu einer beliebigen Teilmenge der Klauseln abzweigen, in denen x_j negiert (bzw. unnegiert) vorkommt. Wenn wir eine Belegung $x_j = 0$ (bzw. $x_j = 1$) mit der rechts-links Durchlaufrichtung (bzw. mit der links-rechts Durchlaufrichtung) der x_j-Kette identifizieren, dann wird offensichtlich: es existiert eine Belegung der x_j, welche alle Klauseln erfüllt, genau dann, wenn in G ein Hamiltonscher Pfad existiert. **qed**

Im Königreich der *Anordnungsprobleme* angekommen, haben wir es nun wieder etwas leichter.

Satz 2.9 *DHP*$\leq_{pol}$*HP.*

Beweis Sei $G = (V, E)$ ein Digraph, der die Eingabe für DHP repräsentiert. Wir verwandeln G durch eine *lokale Ersetzung* in einen Graphen $G' = (V', E')$:

- $V' = \{v_{in}, v', v_{out} |\ v \in V\}$.
- E' besteht
  - aus den (ungerichteten) Verbindungskanten zwischen v_{in} und v' bzw. zwischen v' und v_{out} für alle $v \in V$
  - sowie den Kanten e' für alle $e \in E$. Wenn e eine (gerichtete) Kante von u nach v ist, so ist e' die (ungerichtete) Kante zwischen u_{out} und v_{in}.

Hinter dieser Konstruktion steckt die einfache lokale Ersetzungsregel, die in Abbildung 5 illustriert ist. Die Reduktionsabbildung $G \mapsto G'$ ist sicher in Polynomialzeit berechenbar. Der Beweis wird vervollständigt durch die

Behauptung: Es existiert genau dann ein gerichteter Hamiltonscher Pfad (DHP) in G, wenn ein (ungerichteter) Hamiltonscher Pfad (HP) in G' existiert.

Es sollte klar sein, wie wir einen DHP in G in einen HP in G' transformieren können. Die Knoten v' sind dabei im HP genauso angeordnet wie die Knoten v im DHP (gleiche Durchlaufstrategie).

Gehen wir umgekehrt von einem HP in G' aus. Obwohl die Kanten in G ungerichtet sind, stellen wir sie uns (im Geiste) als gerichtete Kanten der Form (u_{out}, v_{in}), (v_{in}, v'), (v', v_{out}) vor. Die erste Beobachtung ist: die Kanten auf dem HP werden entweder **alle** entlang dieser Orientierung oder **alle** entlang der umgekehrten Orientierung durchlaufen. Denn würden wir etwa bei v_{in} die Orientierung „umpolen“, also v_{in} über e'_1 (mit Eingangskante e_1 von

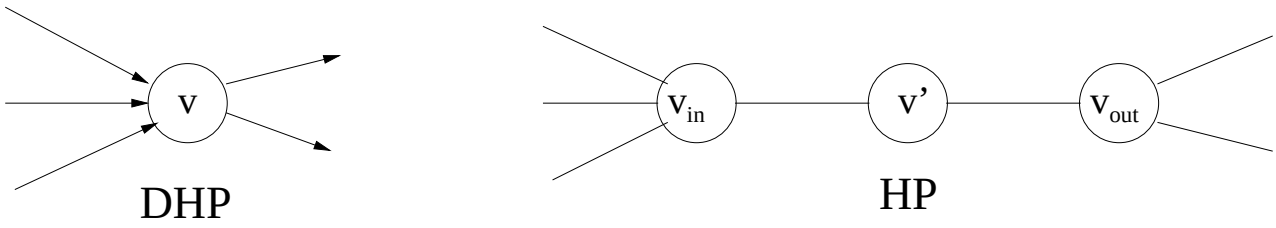

Abbildung 5: Die lokale Ersetzung bei der Reduktion von DHP auf HP.

v) betreten und über e_2' (mit Eingangskante e_2 von v) gleich wieder verlassen, dann könnte HP den Knoten v' nicht durchlaufen, ohne v_{in} oder v_{out} mehrmals zu durchlaufen.[4] Wir können daher oBdA annehmen, dass alle Kanten von HP gemäß unserer imaginierten Orientierung durchlaufen werden. Die gleiche Durchlaufstrategie können wir dann aber auch in G anwenden. G besitzt folglich einen DHP. **qed**

Satz 2.10 *HP$\leq_{pol}$START-ZIEL HP.*

Der Beweis ergibt sich aus Abbildung 6 durch scharfes Hinsehen.

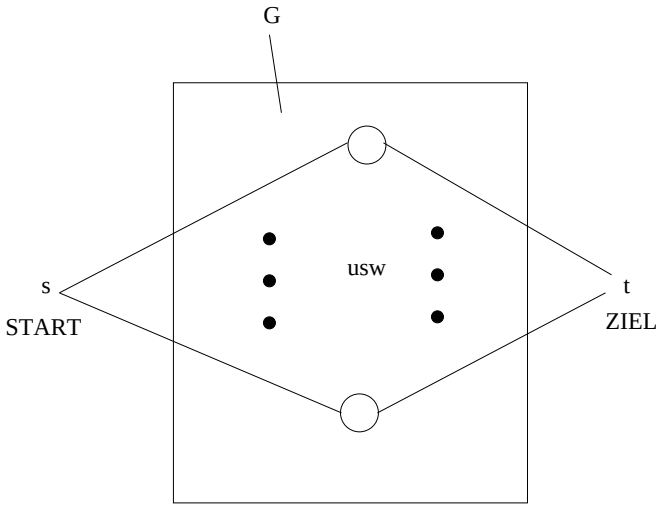

Abbildung 6: Die Reduktionsabbildung für HP$\leq_{pol}$START-ZIEL HP: Knoten s und t sind jeweils mit jedem Knoten aus G durch eine Kante verbunden.

Satz 2.11 *START-ZIEL HP$\leq_{pol}$HC.*

Der Beweis ergibt sich aus Abbildung 7 durch scharfes Hinsehen.

[4]Wir erinnern uns, dass ein Hamiltonscher Pfad jeden Knoten genau einmal durchlaufen muss.

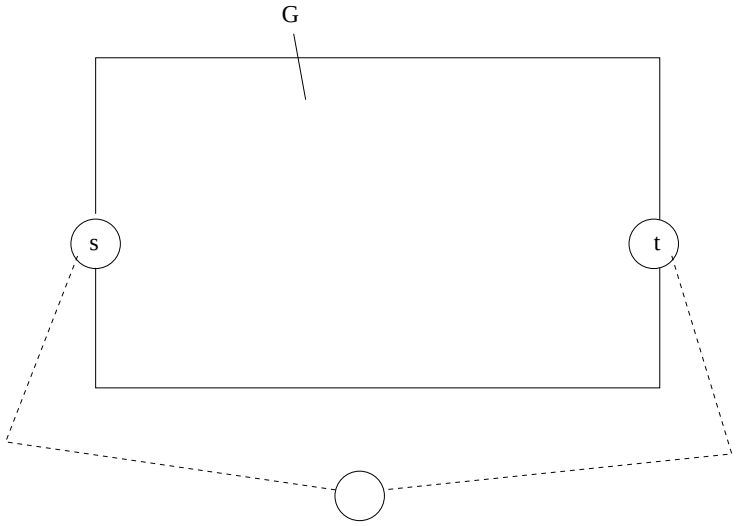

Abbildung 7: Die Reduktionsabbildung für START-ZIEL HP$\leq_{pol}$HC: G' ergibt sich aus G durch Hinzufügen der zwei gestrichelt gezeichneten Kanten, die den Zielknoten t mit dem Startknoten s verbinden.

Die Reduktionskette DHP$\leq_{pol}$START-ZIEL DHP$\leq_{pol}$DHC ist ähnlich zu bewerkstelligen wie die Reduktionskette HP$\leq_{pol}$START-ZIEL HP$\leq_{pol}$HC. (Alternativ lässt sich der Beweis von Satz 2.8 so modifizieren, dass wir anstelle der Reduktion SAT$\leq_{pol}$DHP, die Reduktionen SAT$\leq_{pol}$START-ZIEL DHP bzw. SAT$\leq_{pol}$DHC erhalten.)

Wir kommen zum Finale mit einer *Spezialisierung*.

Satz 2.12 *HC$\leq_{pol}$ TSP.*

Beweis Sei $G = (V, E)$ mit $V = \{1, \ldots, n\}$ eine Eingabeinstanz von HC. Wir assoziieren zu G die Kostenschranke $K_G = n$ und die Distanzmatrix $D_G = D = (d_{ij})_{1 \leq i,j \leq n}$, wobei

$$d_{i,j} = \begin{cases} 0 & \text{falls } i = j, \\ 1 & \text{falls } i \neq j \text{ und } \{i, j\} \in E, \\ 2 & \text{falls } i \neq j \text{ und } \{i, j\} \notin E. \end{cases}$$

Die Reduktionsabbildung $G \mapsto (D_G, K_G)$ ist sicherlich in Polynomialzeit berechenbar. Offensichtlich kann man bezüglich D die Kostenschranke n genau dann einhalten, wenn die Rundreise nur durch Kanten aus E führt, also genau dann, wenn ein Hamiltonscher Kreis in G existiert. **qed**

Die im Beweis von Satz 2.12 vorgeführte Reduktion zeigt, dass HC als folgender Spezialfall von TSP aufgefasst werden kann:

- Die Distanzmatrix $D \in \mathbb{N}_0^{n \times n}$ ist symmetrisch und hat außerhalb der Hauptdiagonalen nur Einträge aus $\{1, 2\}$.

- Die Kostenschranke ist n.

Wir fassen das Hauptresultat dieses Abschnitts zusammen in der

Folgerung 2.13 *Die Probleme SAT, 3-SAT, EXACT-ONE 3-SAT, CLIQUE, INDEPENDENT SET, VERTEX COVER, HITTING SET, SET COVER, SUBSET SUM, KNAPSACK, PARTITION, BIN PACKING, DHP, START-ZIEL DHP, DHC, HP, START-ZIEL HP, HC und TSP sind NP-vollständig.*

Beweis Die Mitgliedschaft dieser Sprachen in der Klasse NP ist jeweils leicht nachzuweisen. Die NP-Härte dieser Sprachen ergibt sich aus dem Cookschen Theorem und der Tatsache, dass von SAT zu jedem dieser Probleme eine Kette von polynomiellen Reduktionen existiert.
qed

A Problemliste

SAT: Satisfiability (Erfüllbarkeitsproblem der Aussagenlogik)

Eingabe: Kollektion $C_1, \ldots, C_m$ von Booleschen Klauseln in n Booleschen Variablen $x_1, \ldots, x_n$. (Eine Boolesche Klausel ist eine Disjunktion von Booleschen Literalen. Ein Boolesches Literal ist eine negierte oder unnegierte Boolesche Variable.)

Frage: Existiert eine Belegung von $x_1, \ldots, x_n$ mit 0 oder 1, die alle Klauseln erfüllt, d.h., die dazu führt, dass $C_1, \ldots, C_m$ zu 1 ausgewertet werden ?

3-SAT: Einschränkung von SAT auf Eingaben, deren Boolesche Klauseln aus jeweils 3 Booleschen Literalen bestehen.

EXACT-ONE 3-SAT: Variante von 3-SAT: es soll entschieden werden, ob eine Belegung existiert, welche jede Klausel mit exakt einem Literal erfüllt.

CLIQUE: Cliquenproblem.

Eingabe: Ein ungerichteter Graph $G = (V, E)$ und eine natürliche Zahl $k \leq |V|$.

Frage: Existiert in G eine Clique der Größe mindestens k, d.h., eine Menge $C \subseteq V$ mit $|C| \geq k$, deren Knoten paarweise in G benachbart sind ?

INDEPENDENT SET: Unabhängige Mengen.

Eingabe: Ein ungerichteter Graph $G = (V, E)$ und eine natürliche Zahl $k \leq |V|$.

Frage: Existiert in G eine unabhängige Menge der Größe mindestens k, d.h., eine Menge $U \subseteq V$ mit $|U| \geq k$, deren Knoten paarweise in G nicht benachbart sind ?

VERTEX COVER: Überdeckung mit Knoten.

Eingabe: Ein ungerichteter Graph $G = (V, E)$ und eine natürliche Zahl $k \leq |V|$.

Frage: Existiert in G ein „Vertex Cover (Knotenüberdeckungsmenge)“ der Größe höchstens k, d.h., eine Menge $C \subseteq V$ mit $|C| \leq k$, die von jeder Kante aus E mindestens einen Randknoten enthält ?

HITTING SET: Auffinden eines Repräsentantensystems.

Eingabe: eine Kollektion $M_1, M_2, \ldots, M_m$ endlicher Mengen und eine natürliche Zahl $k \leq m$.

Frage: Gibt es für diese Mengen ein Repräsentantensystem der Größe höchstens k, d.h., eine Menge R mit $|R| \leq k$, die von jeder der Mengen $M_1, M_2, \ldots, M_m$ mindestens ein Element enthält ?

SET COVER: Mengenüberdeckung.

Eingabe: eine Kollektion $M_1, M_2, \ldots, M_m$ endlicher Mengen und eine natürliche Zahl $k \leq m$.

Frage: Gibt es eine Auswahl von höchstens k dieser Mengen, deren Vereinigung mit der Vereinigung aller Mengen übereinstimmt, d.h., existiert eine Indexmenge $I \subseteq \{1, \ldots, m\}$ mit $|I| \leq k$ und

$$\bigcup_{i \in I} M_i = \bigcup_{i=1}^{m} M_i \ ?$$

SUBSET SUM: Erzielung einer vorgeschriebenen Teilsumme.

Eingabe: n Zahlen $a_1, \ldots, a_n \in \mathbb{N}_0$ und eine „Teilsummenzahl“ $S \in \mathbb{N}_0$.

Frage: Gibt es eine Menge $I \subseteq \{1, \ldots, n\}$, so dass $\sum_{i \in I} a_i = S$?

PARTITION: Zerlegung in zwei gleichgroße Teilsummen.

Eingabe: n Zahlen $a_1, \ldots, a_n \in \mathbb{N}_0$.

Frage: Kann man diese Zahlen in zwei gleichgroße Teilsummen zerlegen, d.h., existiert eine Teilmenge $I \subseteq \{1, \ldots, n\}$, so dass $\sum_{i \in I} a_i = \sum_{j \notin I} a_j$?

KNAPSACK: Rucksackproblem.

Eingabe: n Objekte mit Gewichten $w_1, \ldots, w_n \in \mathbb{N}_0$ und Nutzen $p_1, \ldots, p_n \in \mathbb{N}_0$, eine Gewichtsschranke W und eine Nutzenschranke P.

Frage: Kann man einen Rucksack R so packen, dass die Objekte in R einen Gesamtnutzen von mindestens P und ein Gesamtgewicht von höchstens W besitzen, d.h., existiert eine Teilmenge $I \subseteq \{1, \ldots, n\}$, so dass $\sum_{i \in I} p_i \geq P$ und $\sum_{i \in I} w_i \leq W$?

BP: Bin Packing (Behälterpackungsproblem).

Eingabe: n Objekte der Größen $a_1, \ldots, a_n \in \mathbb{N}_0$, m Behälter (=bins) der „Bingröße“ $b \in \mathbb{N}_0$.

Frage: Kann man die n Objekte so in die m Behälter verpacken, dass in jedem Behälter die Größen der in ihm enthaltenen Objekte sich zu höchstens b addieren, d.h., existiert eine Zerlegung von $\{1, \ldots, n\}$ in m disjunkte Teilmengen $I_1, \ldots, I_m$, so dass $\sum_{i \in I_j} a_i \leq b$ für alle $1 \leq j \leq m$ erfüllt ist ?

DHP: Directed Hamiltonian Path (Gerichteter Hamiltonscher Pfad).

Eingabe: Ein gerichteter Graph $G = (V, E)$.

Frage: Gibt es in G einen gerichteten Hamiltonschen Pfad, d.h., können wir mit (gerichteten) Kanten aus E einen Pfad formen, der jeden Knoten aus V genau einmal durchläuft ?

HP: Hamiltonian Path (Hamiltonscher Pfad).
das entsprechende Problem für ungerichtete Graphen

START-ZIEL-DHP Gerichteter Hamiltonscher Pfad mit vorgegebenem Start und Ziel.
Variante von DHP: die Eingabe enthält neben dem gerichteten Graphen $G = (V, E)$ zwei ausgezeichnete Knoten $s, t \in V$ und vom gerichteten Hamiltonschen Pfad wird gefordert, dass er in s beginnt und in t endet.

START-ZIEL-HP Hamiltonscher Pfad mit vorgegebenem Start und Ziel.
die entsprechende Variante von HP

DHC: Directed Hamiltonian Circuit (Gerichteter Hamiltonscher Kreis).

Eingabe: Ein gerichteter Graph $G = (V, E)$.

Frage: Gibt es in G einen gerichteten Hamiltonschen Kreis, d.h., können wir mit (gerichteten) Kanten aus E einen Kreis formen, der (abgesehen von dem identischen Start- und Zielknoten) jeden Knoten aus V genau einmal durchläuft ?

HC: Directed Hamiltonian Circuit (Gerichteter Hamiltonscher Kreis).
das entsprechende Problem für ungerichtete Graphen

TSP: Travelling Salesman Problem (Problem des Handelsreisenden)

Eingabe: Eine Kostenschranke C, n Städte $C_1, \ldots, C_n$ und eine Distanzmatrix $D = (d_{i,j})_{1 \leq i,j \leq n}$, wobei $d_{i,j} \in \mathbb{N}_0$ die Distanz zwischen C_i und C_j angibt.

Frage: Existiert eine Rundreise durch $C_1, \ldots, C_n$, deren Gesamtlänge C nicht überschreitet, d.h., existiert eine Permutation σ von $1, \ldots, n$, so dass

$$\sum_{i=1}^{n-1} d_{\sigma(i)\sigma(i+1)} + d_{\sigma(n)\sigma(1)} \leq C \ ?$$