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).

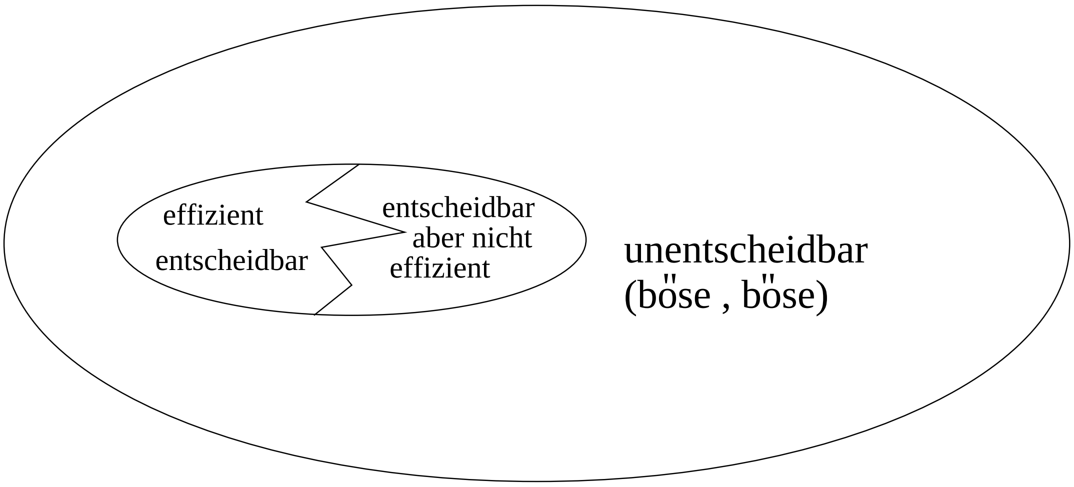

Abbildung 1: Grobunterteilung aller Probleme.

Der Ressourcenverbrauch kann und soll i.A. nicht exakt, sondern nur größenordnungsmäßig erfasst werden. [1] Wir werden aus diesem Grund von der Landau'schen O-Notation Gebrauch machen. Für eine Funktion f von $\mathbb{N}$ nach $\mathbb{N}$ sei die Funktionsmenge $O(f)$ definiert wie folgt:

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

Funktionen g aus $O(f)$ heißen *größenordnungsmäßig durch f beschränkt*. Aus historischen Gründen schreibt man $g = O(f)$ anstelle von $g \in O(f)$. Zum Beispiel gilt $10n^2 + 1000n = O(n^2)$. 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 $10n^2 + 1000n = O(n^3)$, aber $O(n^2)$ ist die genauere Abschätzung der Größenordnung. Die O-Notation sollte aus der

[1] Eine präzise Erfassung würde durch jede Technologieverbesserung obsolet. Nach jeder neuen Computergeneration müsste man die Theorie umschreiben.

Vorlesung „Diskrete Mathematik“ oder einer anderen mathematischen Grundveranstaltung bekannt sein.

Die wichtigsten Ressourcen sind Platz (= Speicherplatz) und Zeit (= Rechenzeit). Seien S, T Funktionen von $\mathbb{N}$ nach $\mathbb{N}$ 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 an den König, der versprach, auf ein Schachbrett mit 64 Feldern Reiskörner der Anzahlen

$$1, 2, 4, 8, 16, \ldots, 2^{63}$$

zu legen und zu verschenken. Er musste bestürzt erkennen, dass alle Kornspeicher seines Reiches zum Einlösen dieses Versprechens nicht ausreichen.

Aus diesen Gründen hat sich die Sichtweise durchgesetzt, polynomielles Wachstum noch als „effizient“ oder zumindest „praktikabel“ durchgehen zu lassen, aber kein superpolynomielles Wachstum zu tolerieren.

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.*

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.*

Diese Definitionen lassen 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.

Bemerkung 1.3 *Falls T, S bestimmte Konstruierbarkeitsaxiome erfüllen, kann man ebensogut verlangen, dass* **alle** *Rechnungen, die Ressourcenschranken $S(n)$, $T(n)$ einhalten, da man die anderen Rechnungen abbrechen kann, ohne dabei L_M zu verändern.*

Definition 1.4 (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 $\overset{?}{\neq}$ NP)-Problems nachgehen. Viele fundamentale Anwendungsprobleme (s. Liste im Anhang) gehören zur Klasse NP. Diese 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. Ein Nachweis von

$$P = NP$$

würde eine Revolution in Theorie und Praxis auslösen. In dieser Situation ist die Theorie der NP-Vollständigkeit entstanden. Salopp gesprochen ist ein NP-vollständiges Problem ein „härtester Vertreter“ der Klasse NP. Wir werden in Kürze zeigen, dass das Erfüllbarkeitsproblem der Aussagenlogik (Satisfiability-Problem oder kurz SAT) NP-vollständig ist. Wenn Sie einen deterministischen Polynomialzeitalgorithmus für SAT entwerfen könnten, hätten sie P = NP bewiesen (und vermutlich Lehrstuhlangebote nach Princeton und Harvard).

Vermutlich aber sind NP-vollständige Probleme nicht (deterministisch) in Polynomialzeit zu lösen. Die Theorie der NP-Vollständigkeit liefert die Legitimation, es dann mit Heuristiken[2] zu probieren.

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

Definition 1.5 (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 1.6 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 $f(w) \in L_0$.

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

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

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.)

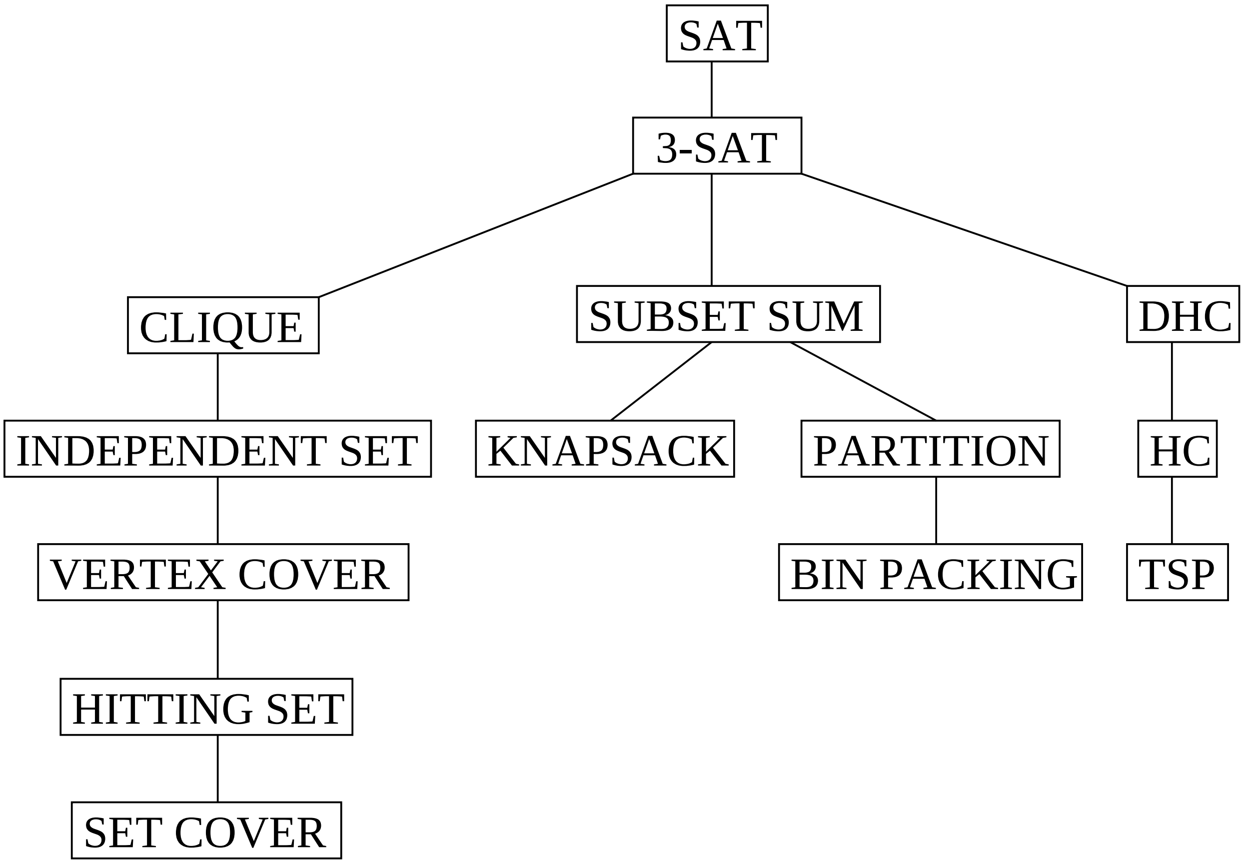

Abbildung 2: Stammbaum NP-vollständiger Probleme.

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 1.7

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

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

Beweis

1. SAT $\in$ NP.
   Gegeben die CNF-Formel F, rate (nichtdeterministisch) die erfüllende Belegung und werte danach F (deterministisch und in Polynomialzeit) aus.

2. Für alle $L \in$ NP: $L \leq_{\text{pol}}$ SAT. Sei $L \in$ NP und M eine polynomiell zeitbeschränkte NTM für L. Es ist relativ leicht einzusehen, dass M sich so normalisieren läßt, dass für geeignete Konstanten $c_1 \leq c_2$ gilt:

   - Sei $R = R(n) = n^{c_1}$. Bei Eingaben der Länge n rät M zunächst einen Binärstring $r \in \{0,1\}^R$ und schreibt diesen in die Zellen $-1, \ldots, -R$.
   - Auf Zelle 0 wird Trennsymbol \$ geschrieben. Zellen $1, \ldots, n$ enthalten weiterhin die Eingabe $w = w_1 \ldots w_n$. 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 Stoppzustand ACC.

   Der wesentliche Trick bei dieser Normalisierung ist, dass alle benötigten „Ratebits“ in der Anfangsphase ermittelt werden und danach deterministisch weitergerechnet wird (Schema des *Ratens und Verifizierens*). 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 Stoppzustand 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)
$S(0,j,B)$ 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. $\Box$

2 Grundlegende NP–vollständige Probleme

Ausgehend von dem Erfüllbarkeitsproblem der Aussagenlogik (SAT) werden wir in diesem Abschnitt mit Hilfe geeigneter polynomiellen 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* $\le_{pol}$ 3-*SAT.*

Beweis Der Beweis benutzt die Methode der *lokalen Ersetzung*: zu einer Klausel

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

suchen wir eine „äquivalente Kollektion“ K_C von 3-Klauseln. K_C benutzt neben den Booleschen Variablen $x_1,\ldots,x_n$ weitere Hilfsvariable $h_{C,1},h_{C,2},\ldots$ und soll zu C in folgendem Sinn äquivalent sein:

Eigenschaft 1 Jede C erfüllende Belegung ist fortsetzbar zu einer K_C erfüllenden Belegung.

Eigenschaft 2 Die Einschränkung einer K_C erfüllenden Belegung auf die Variablen $x_1,\ldots,x_n$ ist eine C erfüllende Belegung.

Falls wir eine polynomiell berechenbare Reduktionsabbildung $C \mapsto K_C$ mit diesen Eigenschaften finden, dann liefert offensichtlich die Reduktionsabbildung

$$(C_1,\ldots,C_m) \mapsto \cup_{j=1}^m K_{C_j}$$

eine polynomielle Reduktion von SAT auf 3-SAT.[3] Begeben wir uns also auf die Suche nach einer geeigneten lokalen Ersetzung $C \mapsto K_C$.

Bei der Reduktionsabbildung $C \mapsto K_C$ unterscheiden wir die folgenden Fälle:

[3] Beachte dabei, dass die Variablen der Klauseln $C_1,\ldots,C_m$ aus dem gemeinsamen Vorrat $x_1,\ldots,x_n$ stammen, wohingegen jede Kollektion K_{C_j} von 3-Klauseln ihren privaten Hilfsvariablenvorrat hat.

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 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 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 diese lokalen Ersetzungen die Eigenschaften 1 und 2 besitzen.

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 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 $h_{k-2} \vee z_{k-1} \vee z_k$.
Wir verifizieren Eigenschaft 1. 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 Eigenschaft 2. 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!**

$\square$

Die polynomielle Reduktion von SAT auf 3-SAT verläuft im Bereich der Booleschen Logik. Die folgende polynomielle Reduktion führt von einem Problem der Booleschen Logik zu einem graphentheoretischen Problem.

Satz 2.2 *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. $\square$

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.

Die nächste polynomielle Reduktion, die wir durchführen, führt von einem Problem der Booleschen Logik (3-SAT) zu einem Zahlenproblem (SUBSET SUM). Diese Reduktion steht vor der technischen Schwierigkeit, dass Zahlenprobleme von Natur aus keinen rein kombinatorischen Charakter haben. Zum Beispiel hat die Übertragsbildung bei arithmetischen Operationen auf Zahlen in kombinatorischen Strukturen keine unmittelbare Entsprechung. Um die Reduktion erfolgreich zu entwerfen, verwenden wir die Technik der *Kombinatorialisierung von Zahlen*. Es wird eine Art „Zahlenpuzzle“ generiert, dass durch Vermeidung von Übertragsbildung eine klare Beziehung zum 3-SAT Problem aufweist.

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

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. Die korrespondierende Eingabe $\mathcal{I}$ für SUBSET SUM soll aus folgenden Zahlen bestehen:

Literalzahlen die Zahlen $A_1, \ldots, A_n$, die zu $x_1, \ldots, x_n$ korrespondieren, und die Zahlen $B_1, \ldots, B_n$, die zu $\bar{x}_1, \ldots, \bar{x}_n$ korrespondieren

Klauselzahlen die Zahlen $E_1, \ldots, E_m$ und $D_1, \ldots, D_m$, die jeweils zu $C_1, \ldots, C_m$ korrespondieren

Teilsummenzahl die angestrebte Teilsumme S

$\mathcal{I}$ soll so entworfen werden, dass C genau dann erfüllbar ist, wenn eine Auswahl der Literal- und Klauselzahlen mit Summe S existiert. Die Zahlen in $\mathcal{I}$ sind in Dezimaldarstellung durch

$$
\begin{aligned}
a_{i,k} &= \begin{cases} 1 & \text{falls } x_k \in C_i \\ 0 & \text{falls } x_k \notin C_i \end{cases} \\
b_{i,k} &= \begin{cases} 1 & \text{falls } \bar{x}_k \in C_i \\ 0 & \text{falls } \bar{x}_k \notin C_i \end{cases} ,
\end{aligned}
$$

$1 \leq i \leq m, 1 \leq k \leq n$, und durch folgende Tabelle gegeben:

a_{11}	$\dots$	a_{1i}	$\dots$	a_{1m}	1	0	0	$\dots$	0	0	A_1
a_{21}	$\dots$	a_{2i}	$\dots$	a_{2m}	0	1	0	$\dots$	0	0	A_2
.		.		.	.	.	.		.	.	.
.		.		.	.	.	.		.	.	.
.		.		.	.	.	.		.	.	.
a_{n1}	$\dots$	a_{ni}	$\dots$	a_{nm}	0	0	0	$\dots$	0	1	A_n
b_{11}	$\dots$	b_{1i}	$\dots$	b_{1m}	1	0	0	$\dots$	0	0	B_1
b_{21}	$\dots$	b_{2i}	$\dots$	b_{2m}	0	1	0	$\dots$	0	0	B_2
.		.		.	.	.	.		.	.	.
.		.		.	.	.	.		.	.	.
.		.		.	.	.	.		.	.	.
b_{n1}	$\dots$	b_{ni}	$\dots$	b_{nm}	0	0	0	$\dots$	0	1	B_n
1	$\dots$	0	$\dots$	0	0	0	0	$\dots$	0	0	E_1
.		.		.	.	.	.		.	.	.
.		.		.	.	.	.		.	.	.
0	$\dots$	1	$\dots$	0	0	0	0	$\dots$	0	0	E_i
.		.		.	.	.	.		.	.	.
.		.		.	.	.	.		.	.	.
0	$\dots$	0	$\dots$	1	0	0	0	$\dots$	0	0	E_m
2	$\dots$	0	$\dots$	0	0	0	0	$\dots$	0	0	D_1
.		.		.	.	.	.		.	.	.
.		.		.	.	.	.		.	.	.
0	$\dots$	2	$\dots$	0	0	0	0	$\dots$	0	0	D_i
.		.		.	.	.	.		.	.	.
.		.		.	.	.	.		.	.	.
0	$\dots$	0	$\dots$	2	0	0	0	$\dots$	0	0	D_m
4	$\dots$	4	$\dots$	4	1	1	1	$\dots$	1	1	S

Es sollte klar sein, dass sich die Reduktionsabbildung $C \mapsto \mathcal{I}$ in Polynomialzeit berechnen lässt. Beschäftigen wir uns nun mit dem kunstvollen Design von $\mathcal{I}$. Jede Zahl in $\mathcal{I}$ besteht aus $m+n$ Dezimalziffern. Die ersten m Dezimalziffern der Literalzahlen sind die $0,1$-wertigen *Inzidenzvariablen*, die anzeigen ob Literal x_k bzw. $\bar{x}_k$ in Klausel C_i vorkommt. Beachte, dass genau drei der Bits $a_{1i}, \dots, a_{ni}, b_{1i}, \dots, b_{ni}$ auf 1 gesetzt sind, da jede Klausel C_i eine 3-Klausel ist. Die letzten n Dezimalziffern sind $0,1$-wertige *Auswahlvariablen*: um die Teilsumme S zu erzeugen, deren letzte n Dezimalziffern Einsen sind, müssen wir ganz offensichtlich für jedes k entweder die Zahl A_k oder die Zahl B_k auswählen (Auswahlregel für Literalzahlen).

Zu $X \subseteq \mathcal{I} \setminus \{S\}$ bezeichne S_X die Summe der Zahlen in X. Beachte, dass bei der Bildung der Summe S_X **keine Überträge** entstehen, egal wieviele Literal- oder Klauselzahlen in X aufgenommen werden.

Die Klauselzahlen dienen der Inkrementierung der ersten m Dezimalziffern in S_X. Aufnahme von E_k (bzw. D_k, bzw. E_k und D_k) in X erhöht die k-te Dezimalziffer von S_X um 1 (bzw. 2, bzw. 3).

Der Beweis von Theorem 2.3 wird abgeschlossen durch die

Behauptung C ist genau dann erfüllbar, wenn eine Auswahl X der Zahlen aus $\mathcal{I}\backslash S$ existiert mit $S_X = S$.

Nehmen wir zunächst an, dass C erfüllbar ist. Dann bestimmen wir zu einer gegebenen erfüllenden Belegung eine geeignete Auswahl X von Zahlen nach der folgenden Strategie:

Regel 1 Falls x_k mit 1 belegt wird, nehmen wir A_k in X auf. Falls x_k mit 0 belegt wird, nehmen wir B_k in X auf.

Regel 2 Falls die i-te Dezimalziffer von S_X den Wert 3 (bzw. 2, bzw. 1) hat, dann nehmen wir zusätzlich E_i (bzw. D_i, bzw. E_i und D_i) in X auf.

Nach Anwendung von Regel 1 stimmt S_X auf den letzten n Dezimalziffern bereits mit S überein. Da die ersten m Dezimalziffern der Literalzahlen die Inzidenzstruktur von C angeben, gilt zusätzlich: wenn C_i durch die erfüllende Belegung mit $r_i \in \{1, 2, 3\}$ Literalen erfüllt wird, dann hat die i-te Dezimalziffer von S_X den Wert r_i (und somit im Vergleich zur Dezimalziffer 4 von S den *Defekt* $4 - r_i$. Anwendung von Regel 2 bewirkt ganz offensichtlich, dass der Defekt auf den ersten m Dezimalziffern exakt ausgeglichen wird. Somit gilt nach Anwendung beider Regeln $S_X = S$.

Nehmen wir umgekehrt an, dass X eine Auswahl von Zahlen aus $\mathcal{I}\backslash S$ ist, so dass $S_X = S$. Notwendigerweise muss X dann die Auswahlregel für Literalzahlen beherzigen. Wir wenden folgende *Belegungsregel* an:

Belege x_k mit 1, falls $A_k \in X$, und mit 0, falls $B_k \in X$.

Sei $i \in \{1, \ldots, m\}$ beliebig aber fest. Da die i-te Dezimalziffer von S_X Wert 4 hat, die Ziffern der Klauselzahlen dazu aber maximal den Beitrag 3 leisten, muss ein Beitrag von mindestens 1 von einer Literalzahl geleistet werden, sagen wir von A_k (der Fall B_k ist symmetrisch). D.h., $A_k \in X$ und $a_{ik} = 1$. Gemäß der Belegungsregel haben wir x_k mit 1 belegt. Da $a_{ik} = 1$, kommt x_k in C_i vor. Die Belegung erfüllt jede Klausel C_i. Damit ist der Beweis abgeschlossen. $\square$

Die in diesem Beweis verwendete Reduktion benutzt erneut *verbundene Komponenten.* Auf den ersten Blick sieht es zwar nach einer *lokalen Ersetzung* aus, bei der Literale durch Literalzahlen und Klauseln durch Klauselzahlen ersetzt werden. Die ersten m Dezimalziffern der Literalzahlen kodieren jedoch die Inzidenzstruktur der Klauselmenge C und schafft implizit eine Verbindung zwischen allen Komponenten. Die Reduktion enthält auch ein neues Werkzeug, nämlich die *Ergänzung.* Oft hinterlässt eine Reduktion (im ersten Versuch) einen *Defekt*, der durch geeignete *Ergänzungskomponenten* beseitigt werden kann. Bei der Reduktion von 3-SAT nach SUBSET SUM bestand dieser Defekt in zu kleinen Dezimalziffern auf den ersten m Zifferpositionen. Wir illustrieren die Reduktion von 3-SAT nach SUBSET SUM abschließend mit einem

Beispiel 2.4 *Sei* $C = (C_1, C_2, C_3)$ *mit*

$$\begin{aligned} C_1 &= x_1 \vee \bar{x}_2 \vee x_3 \\ C_2 &= \bar{x}_1 \vee x_2 \vee \bar{x}_4 \\ C_3 &= \bar{x}_1 \vee \bar{x}_2 \vee \bar{x}_3 \end{aligned}$$

d.h. $m = 3, n = 4$.

S =444 1111			
A_1=100 1000	B_1=011 1000	E_1=100 0000	D_1=200 0000
A_2=010 0100	B_2=101 0100	E_2=010 0000	D_2=020 0000
A_3=100 0010	B_3=001 0010	E_3=001 0000	D_1=002 0000
A_4=000 0001	B_4=010 0001		

Es ist $(1, 1, 0, 0)$ *eine erfüllende Belegung, und es gilt:*

$$A_1 + A_2 + B_3 + B_4 + E_1 + E_3 + D_1 + D_2 + D_3 = S.$$

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 $\mathcal{I} \in \text{SUBSET SUM} \Leftrightarrow \mathcal{I}' \in \mathit{KNAPSACK}$.

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!** □

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. $\square$

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.) □

In unserem angekündigten Stammbaum fehlen noch die Probleme DHC, HC und TSP. Auf geht's!

Satz 2.8 3-*SAT*$\leq_{pol}$*DHC.*

Beweis Sei $C = (C_1, \ldots, C_m)$ mit $C_i = z_{i1} \vee z_{i2} \vee z_{i3}$ und $z_{ij} \in \{x_0, \bar{x}_0, \ldots, x_{n-1}, \bar{x}_{n-1}\}$ eine Eingabe für 3-SAT. Unser Ziel ist C effizient in einen Digraphen G zu transformieren, so dass C genau dann erfüllbar ist, wenn es in G einen gerichteten Hamiltonschen Kreis gibt.

Wir beginnen mit einer groben Modellierung der CNF-Formel C durch einen Digraphen $G' = (V', E')$. In G' ist jede Boolesche Variable x_k, $k = 0, \ldots, n-1$, durch einen Knoten repräsentiert, den wir ebenfalls mit x_k notieren. Analog repräsentieren wir jede Klausel C_i, $i = 1, \ldots, m$, durch einen Klauselknoten C_i.[4] Aus dem Kontext wird stets hervorgehen, ob mit x_k (bzw. C_i) die Boolelsche Variable (bzw. Klausel) oder der betreffende Knoten in G' gemeint ist. Die Knotenmenge von G' ist somit gegeben durch

$$V' = \{x_0, \ldots, x_{n-1}, C_1, \ldots, C_m\}.$$

Kantenmenge E' wird so entworfen, dass sie eine Zerlegung der Form

$$E' = \bigcup_{k=0}^{n-1} (E(k,0) \cup E(k,1))$$

besitzt. Für alle $k = 0, \ldots, n-1$ seien

$$i_1(k) < \cdots < i_{r(k)}(k) \text{ bzw. } \bar{i}_1(k) < \cdots < \bar{i}_{\bar{r}(k)}(k)$$

die Indizes aller Klauseln, in denen x_k bzw. $\bar{x}_k$ vorkommt. Mit diesen Bezeichnungen bestehe $E(k,1)$ aus allen Kanten des Pfades

$$P(k,1): \; x_k \to C_{i_1(k)} \to C_{i_2(k)} \to \cdots C_{i_{r(k)}(k)} \to x_{k+1 \bmod n}.$$

Analog bestehe $E(k,0)$ aus allen Kanten des Pfades

$$P(k,0): \; x_k \to C_{\bar{i}_1(k)} \to C_{\bar{i}_2(k)} \to \cdots C_{\bar{i}_{\bar{r}(k)}(k)} \to x_{k+1 \bmod n}.$$

[4]Die mehrdeutige Verwendung von x_k und C_i ist etwas schlampig, unterstützt aber im Folgenden das intuitive Verständnis der Konstruktion.

Pfad $P(k,1)$ repräsentiert einen Spaziergang von x_k nach $x_{k+1 \bmod n}$, bei dem der Reihe nach alle Klauseln (sprich: Klauselknoten) C_i mit $x_k \in C_i$ besucht werden. Pfad $P(k,0)$ repräsentiert einen Spaziergang von x_k nach $x_{k+1 \bmod n}$, bei dem der Reihe nach alle Klauseln (sprich: Klauselknoten) $C_{\bar{i}}$ mit $\bar{x}_k \in C_{\bar{i}}$ besucht werden.

Mit einer Belegung $a = (a_0, \ldots, a_{n-1}) \in \{0,1\}^n$ der Variablen $x_0, \ldots, x_{n-1}$ verbinden wir die folgende Rundtour durch G':

$$R'(a): \; x_0 \stackrel{P(0,a_0)}{\longrightarrow} x_1 \stackrel{P(1,a_1)}{\longrightarrow} \cdots \stackrel{P(n-2,a_{n-2})}{\longrightarrow} x_{n-1} \stackrel{P(n-1,a_{n-1})}{\longrightarrow} x_0$$

Beachte, dass $P(k, a_k)$ alle Klauseln (sprich: Klauselknoten) besucht, die durch die Belegung $x_k = a_k$ erfüllt werden. Dies führt zu folgender

Beobachtung: a ist eine Belegung, die alle Klauseln $C_1, \ldots, C_m$ erfüllt **gdw** $R'(a)$ eine Rundtour ist, die alle Klauselknoten $C_1, \ldots, C_m$ (und somit alle Knoten) in G' besucht.

Da sich verschiedene Pfade $P(k, a_k)$ und $P(k', a_{k'})$ an demselben (dann mehrfach besuchten) Klauselknoten kreuzen können, ist $R'(a)$ aber i.A. noch kein gerichteter Hamiltonscher Kreis. Um eine Entflechtung der Pfade zu erreichen, müssen wir unsere Modellierung verfeinern. Details folgen.

Wir ersetzen jeden Klauselknoten C_i durch einen Untergraphen H_i, der isomorph zu dem Hilfsgraphen H in Abbildung 3 ist. H hat drei Eingangsknoten $1, 2, 3$ und drei Ausgangsknoten $1', 2', 3'$. Es ist nicht schwer zu sehen, dass H die folgenden Eigenschaften hat[5]:

Eigenschaft 1 Sei $p \in \{1,2,3\}$. p Pfade, die H über die Eingangsknoten $j_1, \ldots, j_p$ (in dieser Reihenfolge) betreten, **können** so durch H geroutet werden, dass H an den Ausgangsknoten $j'_1, \ldots, j'_p$ (in dieser Reihenfolge) wieder verlassen wird. Dabei wird jeder Knoten von H genau einmal durchlaufen.

Eigenschaft 2 Sei $p \in \{1,2,3\}$. Wenn p Pfade, die H über die Eingangsknoten $j_1, \ldots, j_p$ (in dieser Reihenfolge) betreten, so durch H geroutet werden, dass jeder Knoten von H genau einmal durchlaufen wird, dann **müssen** diese Pfade H über die Ausgangsknoten $j'_1, \ldots, j'_p$ (in dieser Reihenfolge) wieder verlassen.

Ein Pfad $P(k,1)$ mit $x_k = z_{ij}$ für ein $j \in \{1,2,3\}$, der vorher durch Klauselknoten C_i geroutet wurde, wird nunmehr so geroutet, dass er H_i über den j–ten Eingangsknoten betritt und über den j–ten Ausgangsknoten verlässt. Eine analoge Bemerkung gilt für die Pfade der Form $P(k,0)$. [6] Den Graphen, der auf diese Weise entsteht, bezeichnen wir als $G = (V, E)$. Die Reduktionsabbildung $C \mapsto G$ ist offensichtlich in Polynomialzeit berechenbar.

Wir können nun eine Rundtour der Form $R'(a)$ durch G' in eine Rundtour $R(a)$ durch G transformieren wie folgt:

[5] Hilfsgraph H ist gerade so kunstvoll entworfen, dass diese Eigenschaften gelten. Konstruktionen dieser Art werden im Amerikanischen *gadget (Spielzeug)* genannt. Die Eigenschaften eines Gadget kann man verifizieren, indem man eine Weile damit herumspielt.

[6] S. Abbildung 4 zur Illustration. Beachte, dass C_i ein einziger (wenn auch dicker–fetter) Knoten ist; H_i hingegen ist der (hier als „Black Box“ dargestellte) zu H isomorphe Graph.

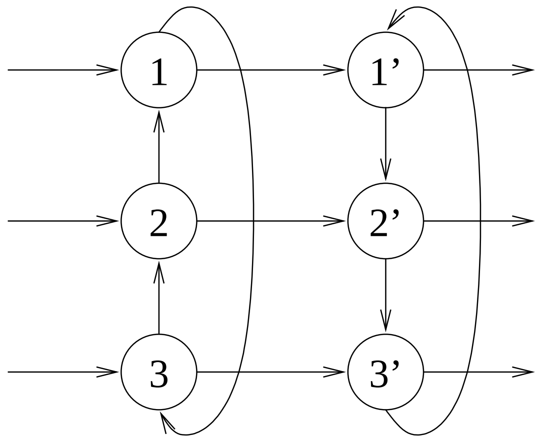

Abbildung 3: Die Klauselkomponente H in der Reduktion von 3-SAT auf DHC

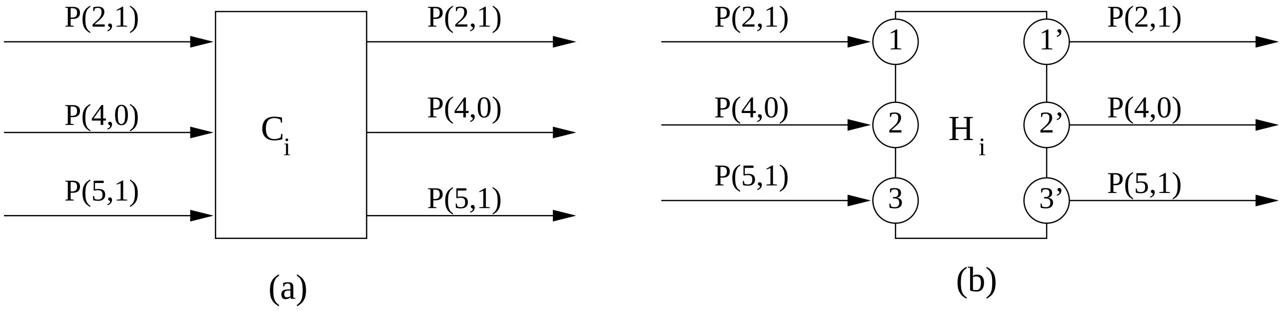

Abbildung 4: (a) Pfade durch C_i mit $C_i = x_2 \vee \bar{x}_4 \vee x_5$. (b) Pfade durch H_i.

- Besuche die Untergraphen H_i mit der gleichen Routing–Strategie wie vorher die Klauselknoten C_i.
- Wenn H_i insgesamt p–mal besucht wird (mit $p \in \{1, 2, 3\}$), dann verwende innerhalb H_i die bei „Eigenschaft 1“ geschilderte Routing–Strategie.

Mit Hilfe von Eigenschaft 1 erkennt man leicht die Äquivalenz der folgenden Aussagen:

(1) a ist eine Belegung, welche die Klauseln $C_1, \ldots, C_m$ erfüllt.

(2) $R'(a)$ besucht alle Klauselknoten $C_1, \ldots, C_m$ (sowie natürlich die Knoten $x_0, \ldots, x_{n-1}$).

(3) $R(a)$ besucht alle Knoten von G genau einmal, d.h., $R(a)$ ist ein gerichteter Hamiltonscher Kreis durch G.

Es folgt direkt, dass es in G einen gerichteten Hamiltonschen Kreis gibt, falls die Klauseln $C_1, \ldots, C_m$ erfüllbar sind.

Der Beweis wird nun abgeschlossen, indem wir umgekehrt nachweisen, dass die Klauseln $C_1, \dots, C_m$ erfüllbar sind, wenn G einen gerichteten Hamiltonschen Kreis K enthält.

Wir konstruieren zunächst eine Belegung $a \in \{0,1\}^n$. Da K jeden Knoten von G genau einmal durchläuft, wird x_k entweder über den Pfad $P(k,0)$ oder über den Pfad $P(k,1)$ verlassen. Wenn K den Teilpfad $P(k,0)$ verwendet, setzen wir $a_k = 0$; andernfalls setzen wir $a_k = 1$.

Mit Hilfe von Eigenschaft 2 erkennt man leicht, dass der Kreis K die Form $R(a)$ haben muss. Wegen der Äquivalenz der obigen Aussagen (1) und (3) ergibt sich nun direkt, dass a eine erfüllende Belegung ist. $\square$

Diese Reduktion benutzte offenkundig wieder eine Reduktionsabbildung mit *verbundenen Komponenten*. Die Knoten x_k sind Variablenkomponenten und die Untergraphen H_i Klauselkomponenten. Die Verbindungsstruktur zwischen diesen Komponenten wurde überaus trickreich gewählt.

Eine Reduktion von L_1 nach L_2 ist i.A. technisch umso anspruchsvoller je größer die inhaltliche Kluft zwischen den Problemen L_1 und L_2 ist. 3-SAT ist ein *Auswahlproblem der Booleschen Logik*: zu jeder Booleschen Variable müssen wir einen der Booleschen Wahrheitswerte $0, 1$ auswählen. Die Reduktion von 3-SAT auf CLIQUE führte zu einem *graphentheoretischen Auswahlproblem*: wir müssen paarweise verbundene Knoten auswählen. Obwohl also die Kluft zwischen Boolescher Logik und Graphentheorie überbrückt werden musste, handelte es sich immerhin beidemal um *Auswahlprobleme*. Die Reduktion von 3-SAT auf DHC musste eine größere Kluft überbrücken: sie führt von einem *Auswahlproblem der Booleschen Logik* zu einem *Anordnungsproblem der Graphentheorie*: wir müssen die Knoten eines Digraphen so anordnen, dass sie im Digraphen einen gerichteten Hamiltonschen Kreis bilden. Es ist also kein Zufall, dass eine komplexe Beweisführung notwendig war.

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

Satz 2.9 *DHC*$\leq_{pol}$*HC.*

Beweis Sei $G = (V, E)$ ein Digraph, der die Eingabe für DHC 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 Kreis (DHC) in G, wenn ein (ungerichteter) Hamiltonscher Kreis (HC) in G existiert.

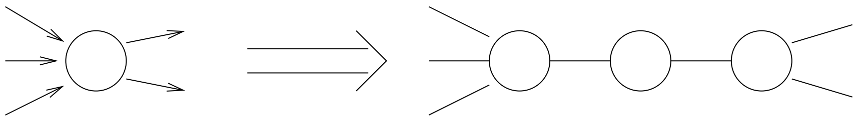

Abbildung 5: Die lokale Ersetzung bei der Reduktion von DHC auf HC.

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

Gehen wir umgekehrt von einem HC 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') oder (v', v_{out}) vor. Die erste Beobachtung ist: die Kanten auf dem HC 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 v) betreten und über e'_2 (mit Eingangskante e_2 von v) gleich wieder verlassen, dann könnte HC den Knoten v' nicht durchlaufen, ohne v_{in} oder v_{out} mehrmals zu durchlaufen.[7] Wir können daher oBdA annehmen, dass alle Kanten von HC gemäß unserer imaginierten Orientierung durchlaufen werden. Die gleiche Durchlaufstrategie können wir dann aber auch in G anwenden. G besitzt folglich einen DHC. □

Wir kommen zum Finale mit einer *Spezialisierung.*

Satz 2.10 *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. □

Die im Beweis von Satz 2.10 vorgeführte Reduktion zeigt, dass HC als folgender Spezialfall von TSP aufgefasst werden kann:

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

- Die Distanzmatrix $D \in \mathbb{N}^{n \times n}$ 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 2.11 *Die Probleme SAT, 3-SAT, CLIQUE, INDEPENDENT SET, VERTEX COVER, HITTING SET, SET COVER, SUBSET SUM, KNAPSACK, PARTITION, BIN PACKING, DHC, 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. □

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.

CLIQUE Cliquenproblem.

> **Eingabe** Ein ungerichteter Graph $G = (V, E)$ und eine Anzahl k.
>
> **Frage** Existiert in G eine Clique der Größe k, d.h., eine Menge $C \subseteq V$ der Mächtigkeit k, deren Knoten paarweise in G benachbart sind ?

INDEPENDENT SET Unabhängige Mengen.

> **Eingabe** Ein ungerichteter Graph $G = (V, E)$ und eine Anzahl k.
>
> **Frage** Existiert in G eine unabhängige Menge der Größe k, d.h., eine Menge $U \subseteq V$ der Mächtigkeit k, deren Knoten paarweise in G nicht benachbart sind ?

VERTEX COVER Überdeckung mit Knoten.

Eingabe Ein ungerichteter Graph $G = (V, E)$ und eine Anzahl k.

Frage Existiert in G ein „Vertex Cover (Knotenüberdeckungsmenge)“ der Größe k, d.h., eine Menge $C \subseteq V$ der Mächtigkeit 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 Zahl $k \in \mathbb{N}$.

Frage Gibt es für diese Mengen ein Repräsentantensystem der Größe k, d.h., eine Menge R der Mächtigkeit 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 Zahl $k \in \mathbb{N}$.

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

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

SUBSET SUM Erzielung einer vorgeschriebenen Teilsumme

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

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 $a_1, \ldots, a_n \in \mathbb{N}$.

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 $w_1, \ldots, w_n \in \mathbb{N}$ und Nutzen $p_1, \ldots, p_n \in \mathbb{N}$, 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 $a_1, \ldots, a_n \in \mathbb{N}$, m Behälter (=bins) der „Bingröße“ b.

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 ?

HC Hamiltonian Circuit (Hamiltonscher Kreis)

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

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

DHC Directed Hamiltonian Circuit (Gerichteter Hamiltonscher Kreis)
Dies ist das entsprechende Problem für gerichtete 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 $d_{i,j} \in \mathbb{N}$ 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

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