2 Komplexitätstheorie

Die Komplexitätstheorie untersucht den Ressourcenbedarf (Zeitbedarf, Speicherplatz, Programmlänge,...) von Algorithmen.

Die Komplexitätstheorie ist interessiert an:

oberen Schranken:	Garantie, daß das untersuchte Problem im Rahmen der vorgegebenen Ressourcen gelöst werden kann. Ein solcher Algorithmus heißt obere Schranke.
unteren Schranken:	Aussagen über den Mindestbedarf an Ressourcen für die Lösung eines Problems.

Ein Algorithmus ist (asymptotisch) *optimal* für ein Problem, wenn er bei der Lösung des Problems mit den durch die untere Schranke beschriebenen Ressourcen auskommt.

2.1 Komplexitätsmaße und Komplexitätsklassen

Beschreibe Problem als formale Sprache "Entscheidungsproblem":

$$A \subseteq \Sigma^*$$

$x \in A \Leftrightarrow x$ "löst" Problem

Da Algorithmen untersucht werden, kann man sich auf die Klasse der entscheidbaren Sprachen beschränken.

Sei M eine Mehrband–TM, die bei allen Eingaben $x \in \Sigma^*$ hält.

$time_M : \Sigma^* \rightarrow I\!N$ "Zeitkomplexität von M".
$time_M(x)$ = Anzahl der Rechenschritte von M bei Eingabe von x.
$space_M : \Sigma^* \rightarrow I\!N$ "Speicherkomplexität von M".
$time_M(x)$ = Anzahl der bei Eingabe von x von M benutzten Felder der Bänder.

Sei $A \subseteq \Sigma^*$ entscheidbar.
$time_A(x) = min\{time_M(x) : M$ ist Mehrband–TM für $A\}$ ist Zeitkomplexität von A
$space_A(x) = min\{space_M(x) : M$ ist Mehrband–TM für $A\}$ ist Speicherkomplexität von A

Sei $f : I\!N \rightarrow I\!N$
$TIME(f(n)) = \{A \subseteq \Sigma^* : time_A(x) \leq f(|x|)\}$
$SPACE(f(n)) = \{A \subseteq \Sigma^* : space_A(x) \leq f(|x|)\}$

Satz.
Die Komplexitätsklasse $TIME(f(n))$, wobei $f(n)$ nach oben durch LOOP–Programme beschränkt werden kann, ist enthalten in der Klasse der primitiv rekursiven Sprachen (bzw. der LOOP–berechenbaren Sprachen).

Beweis.
Sei M TM mit $time_M(x) \leq f(|x|)$.
Simuliere M durch ein GOTO–Programm , wobei jeder TM–Schritt durch eine endliche Zahl von Wertzuweisungen bzw. GOTOs simuliert wird. Forme das GOTO–Programm in ein äquivalentes WHILE–Programm mit nur einer WHILE–Schleife um.
Die Anzahl der Durchläufe der WHILE–Schleife ist durch die Zahl $f(n)$ beschränkt.
Ersetze "WHILE count$\neq 0$ DO" durch "$y := f(n)$;LOOP y DO".
Das Ergebnis ist ein LOOP–Programm, das M simuliert. ■

Korollar
$TIME(n^k)$ $(k \in I\!N), TIME(2^n), TIME(2^{2^{\cdot^{\cdot^{2}}}})$ enthalten nur primitiv rekursive Mengen.

Komplexitätsmaße können auch mit Hilfe von WHILE–Programmen eingeführt werden.
Vorsicht
$x_i := x_j$ muß so groß angesetzt werden, wie die Anzahl der Bits, die bei dieser Aktion übertragen werden (also etwa $\log x_j$).
("logarithmisches Kostenmaß")
Dieses Kostenmaß ist zwar genau, aber schwierig zu analysieren.
Setzt man dagegen die Kosten für elementare Anweisungen 1, spricht man vom "uniformen Kostenmaß".

Beispiel
INPUT (n);
$x := 2$;
LOOP n DO $x := x * x$ END;
OUTPUT(x);

Der Algorithmus berechnet 2^{2^n}.
Kosten bei: "uniformem Kostenmaß": $O(n)$
"logarithmischem Kostenmaß": $O(2^n)$

Üblich
Die Komplexität eines Algorithmus wird unter dem uniformen Kostenmaß angegeben.

2.2 Die Komplexitätsklassen P und NP

Es ist sinnvoll, anstelle von Funktionen Funktionsklassen zu betrachten:

Beispiel
Problem ist mit Einband–TM M lösbar $\Leftrightarrow$
Problem ist mit Mehrband–TM N lösbar,
aber
$time_M = (time_N)^2$

Es werden deshalb Klassen betrachtet, die gegenüber solchen Modifikationen abgeschlossen sind.

Definition.
Ein *Polynom* $p : I\!N \to I\!N$ ist eine Funktion der Form
$p(n) = a_k n^k + \ldots + a_1 n + a_0 \quad a_i \in I\!N, k \in I\!N$

Komplexitätsklasse P

$$\begin{aligned} P &= \bigcup_{P\ Polynom} TIME(p(n)) \\ &= \{A : \exists \text{ TM } M \text{ mit } L(M) = A \text{ und } time_M(x) \leq p(|x|) \text{ für ein Polynom } p\} \\ \Rightarrow P &= \bigcup_{k \geq 1} TIME(O(n^k)) \end{aligned}$$

Bermerkung
Auch Algorithmen der Komplexität $n \log n$ sind polynomial, da
$n \log n = O(n^2)$.

$n^{\log n}, 2^n$ sind nicht polynomial ("exponentiell").

Allgemein akzeptiert

P = Klasse der effizienten Algorithmen(d.h. der praktisch realisierbaren Algorithmen) .

Algorithmen mit exponentieller Laufzeit sind nicht effizient.

- P könnte auch als WHILE–Programm mit logarithmischen Kostenmaß definiert werden.
- $P \subseteq TIME(2^n) \subseteq$ {Primitiv rekursive Sprache}
- P enthält alle Probleme, für die sich in polynomieller Zeit ein Beweis finden läßt.

Betrachte daneben die Klasse NP der Probleme, für die sich in polynomieller Zeit ein vorgegebener Beweis überprüfen läßt:

Definition.
Eine *nichtdeterministische TM* ist diejenige TM, bei der die Übergangsfunktion eine Übergangsrelation ist (eine Konfiguration hat i.a. mehrere mögliche Nachfolgekonfigurationen). Eine *akzeptierende Berechnung* besteht aus einer zulässigen Folge von Konfigurationen, die mit der Startkonfiguaration beginnt und in einer akzeptierenden Konfiguration endet.

Definition.
Sei M eine nichtdeterministische Mehrband–TM.

$$time_M(x) = \begin{cases} \min\{\text{Länge einer akzeptierenden Berechnung von } M \text{ auf } x\}, & x \in L(M) \\ 0, & x \notin L(M) \end{cases}$$

$f : I\!N \to I\!N$

$$NTIME(f(n)) = \{A : \exists \text{ NTM } M \text{ mit } L(M) = A \text{ und } time_M(x) \leq f(|x|) \forall x \in \Sigma^*\}$$

$$\begin{aligned} NP &:= \bigcup_{p \ Polynom} NTIME\ p(n) \\ &= \bigcup_{k \geq 1} TIME(O(n^k)) \end{aligned}$$

Offenbar gilt:

$$P \subseteq NP$$

Die Umkehrung ist unklar: "$P - NP$–Problem"

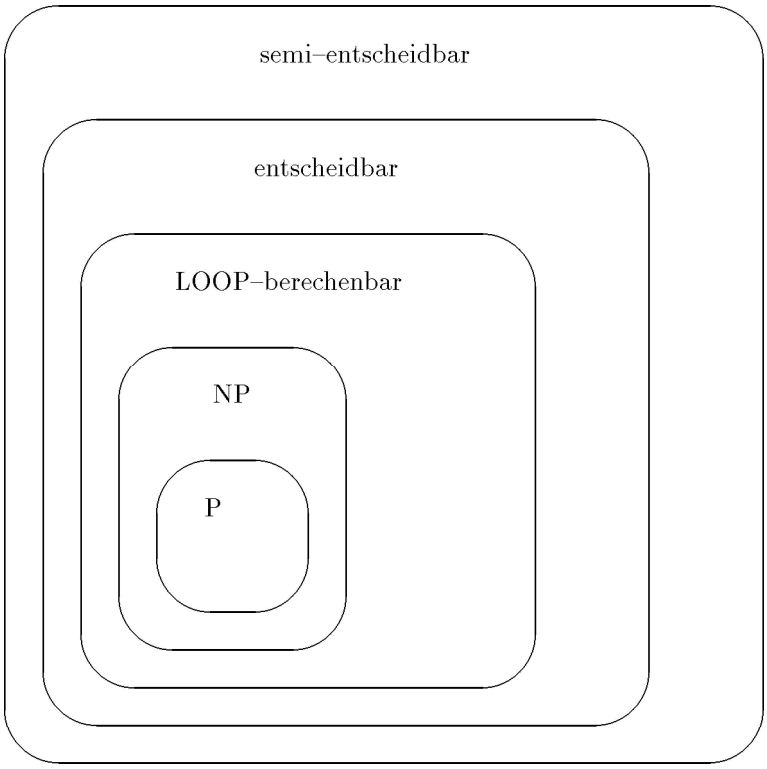
Das $P - NP$–Problem ist wegen seiner Bedeutung für die Findung effizienter Algorithmen von großer Bedeutung (von vielen als das bedeutendste Problem der Theoretischen Informatik angesehen).

Allgemeine Annahme

$$P \neq NP$$

Bei der Untersuchung des $P - NP$–Problems wurde die Theorie der NP–Vollständigkeit entwickelt. (Cook '71, Karp '72)

Es gilt:



2.3 NP–Vollständigkeit

Definition.
Seien $A, B \subseteq \Sigma^*$.
A heißt *polynomiell auf B reduzierbar* ($A \leq_p B$) , falls es eine totale und mit polynomialer Komplexität berechenbare Funktion $f : \Sigma^* \to \Sigma^*$ gibt, so daß für alle $x \in \Sigma^*$ gilt:

$$x \in A \Leftrightarrow f(x) \in B$$

Lemma
$\leq_p$ ist transitiv.

Beweis.
Übungsaufgabe.

Lemma
i) $A \leq_p B, B \in P \Rightarrow A \in P$
ii) $A \leq_p B, B \in NP \Rightarrow A \in NP$
Beweis.
i) Sei $A \leq_p B$ mittels f.
TM M_f berechne f in Polynomialzeit p.
Sei $B \in P$ mittels TM M in Rechenzeit q.
$\Rightarrow (M_f; M)$ berechnet A in Rechenzeit
$p(|x|) + q(|f(x)|) \leq p(|x|) + q(p(|x|))$ ist Polynom. ■

ii) analog

Definition.
Eine Sprache A heißt *NP–hart*, falls für alle Sprachen $L \in NP$ gilt: $L \leq_p A$.
Eine Sprache A heißt *NP–vollständig*, falls A NP–hart ist und $A \in NP$ gilt.

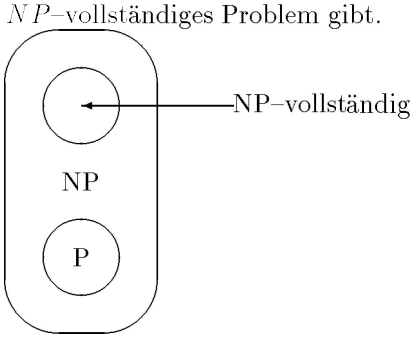
Bemerkung
NP-vollständige Sprachen sind "schwere" Sprachen in NP.

Satz.
Sei A NP–vollständig.
Dann gilt:

$$A \in P \Leftrightarrow P = NP$$

Beweis.
(—): Sei $A \in P, L \in NP$ beliebig.
da A NP–hart gilt: $L \leq_p A$
$\Rightarrow_{Lemma} L \in P$ ■
(—): $P = NP \Rightarrow A \in P$

$\Rightarrow$ Zum Nachweis von $P = NP$ genügt die Angabe eines polynomialen Algorithmus für ein NP–vollständiges Problem:
Die Annahme $P \neq NP$ bedeutet, daß es keinen effizienten Algorithmus für ein NP–vollständiges Problem gibt.



Definition.
Das folgende Problem heißt "Erfüllbarkeitsproblem der Aussagenlogik" SAT.

Gegeben:	Eine Formel F der Aussagenlogik mit n Variablen, $n \in I\!N$
Gefragt:	Ist F erfüllbar?
	(d.h. $\exists$ Belegung $a \in \{0,1\}^n$ mit $F(a) = 1$?)

Formal
$SAT = \{code(F) \in \Sigma^* : F \text{ ist erfüllbare Formel der Aussagenlogik}\}$

Theorem von Cook
Das Erfüllbarkeitsproblem der Aussagenlogik SAT ist NP–vollständig.

Beweis.
1.) Zu zeigen: $SAT \in NP$.
Angabe einer polynomial zeitbeschränkten NTM für SAT:
M stellt in einem Durchlauf über der Eingabe fest, welche Variablen in F vorkommen.
Seien dies $x_1, \ldots, x_k$.
M rät die Werte $a_1, \ldots, a_k \in \{0,1\}$ für $x_1, \ldots, x_k$ und setzt diese in F ein.
(es existieren 2^k mögliche unabhängige Berechnungen – für jede Belegung eine)
Für jede Belegung rechnet M jeweils deterministisch den Wert von F aus und akzeptiert, falls dieser 1 ist.
$F \in SAT \Leftrightarrow M$ akzeptiert F.
Wegen $k \leq |F|$ ist M polynomialzeit beschränkt.
$\Rightarrow SAT \in NP$.

2.) Zu zeigen: SAT ist NP–hart.
Sei $L \in NP$ beliebig.
M NTM für L der Rechenzeit p.
OBdA gelte: $\delta(z_e, a) \ni (z_e, a, N)$ (d.h. erreichte Endzustände werden nicht mehr

verlassen).
Sei $x = x_1, \ldots, x_n \in \Sigma^*$ die Eingabe von M.
Konstruktion einer Fromel F mit

$x \in L \Leftrightarrow F$ ist erfüllbar.

Sei $\Gamma = \{a_1, \ldots, a_l\}$ das Anfangsalphabet
$Z = \{z_1, \ldots, z_k\}$ die Zustandsmenge von M
F enthält folgende Variable:

Variable	Indizes	intendierte Bedeutung
$zust_{t,z}$	$t = 0, \ldots, p(n)$ $z \in Z$	$zust_{t,z} = 1 \Leftrightarrow$ nach t Schritten befindet sich M im Zustand z
$pos_{t,i}$	$t = 0, \ldots, p(n)$ $i = -p(n), \ldots, p(n)$	$pos_{i,j} = 1 \Leftrightarrow$ M's Schreib–Lesekopf befindet sich nach t Schritten auf Position i
$band_{t,i,a}$	$t = 0, \ldots, p(n)$ $i = -p(n), \ldots, p(n)$ $a \in \Gamma$	$band_{t,i,a} = 1 \Leftrightarrow$ nach t Schritten befindet sich auf Bandposition i das Zeichen a

F besteht aus mehreren Bauteilen:

$G(x_1, \ldots, x_m) = 1 \Leftrightarrow$ für geanu ein i ist $x_i = 1$.

Behauptung
G existiert und es gilt $size(G) = O(m^2)$.

Beweis.

$$G = (\bigvee_{i=1}^{k} x_i) \wedge (\bigwedge_{j=1}^{m-1} \bigwedge_{l=j+1}^{m} (\neg(x_j \wedge x_l))$$

Die erste Teilformel wird genau dann wahr, wenn mindestens eine Variable wahr ist.
Die zweite Teilformel wird genau dann wahr, wenn höchstens eine Variable wahr wird.

$F = R \wedge A \wedge \ddot{U}_1 \wedge \ddot{U}_2 \wedge E$, wobei

R	für Randbedingung
A	für Anfangsbedingung
$\ddot{U}_1, U_2$	für Übergangsbedingung und
E	für Endbedingung steht.

R drückt aus:

- Zu jedem Zeitpunkt t ergibt sich für genau ein z $zust_{t,z} = 1$.
- Zu jedem Zeitpunkt t gibt es genau eine Bandposition i mit $pos_{t,i} = 1$.
- Zu jedem Zeitpunkt t und jeder Bandposition i gibt es genau ein a mit $band_{t,i,a} = 1$

$$R = \bigwedge_t [G(zust_{t,z_1}, \ldots, zust_{t,z_k}) \wedge G(pos_{t,-p(n)}, \ldots, pos_{t,p(n)}) \wedge \bigwedge_i G(band_{t,i,a_1}, \ldots, band_{t,i,a_l})]$$

A beschreibt den Status der Variablen für den Fall $t = 0$:

$$A = zust_{0,z_0} \wedge pos_{0,1} \wedge \bigwedge_{j=1}^{n} band_{0,j,x_j} \wedge \bigwedge_{j=-p(n)}^{0} band_{0,j,\square} \wedge \bigwedge_{j=n+1}^{p(n)} band_{0,j,\square}$$

$\ddot{U}_1$ beschreibt den Übergang von Zeitpunkt t nach $t+1$ an der Kopfposition ($y \in \{-1, 0, +1\}$):

$\ddot{U}_1 = \bigwedge_{t,z,i,a} [(zust_{t,z} \wedge pos_{t,i} \wedge band_{t,i,a}) \rightarrow \bigvee_{z',a',y \text{ mit } \delta(z,a) \ni (z',a',y)} (zust_{t+1,z'} \wedge pos_{t+1,i+y} \wedge band_{t+1,i,a'}]$

$\ddot{U}_2$ besagt, daß auf den übrigen Bandfeldern nichts passiert:

$$\ddot{U}_2 = \bigwedge_{t,i,a} ((\neg pos_{t,i} \wedge band_{t,i,a}) \rightarrow band_{t+1,i,a})$$

E überprüft, ob der Endzustand erreicht ist (wird auf jeden Fall im Zeitpunkt $p(n)$ erreicht):

$$E = \bigvee_{z \in E} zust_{p(n),z}$$

($\rightarrow$): Sei $x \in L$
$\Rightarrow \exists$ nichtdeterministische Rechnung der Länge $p(n)$, die in einen Endzustand führt.
$\Rightarrow$ Alle Teilformeln von F erhalten den Wert 1.
$\Rightarrow F(x)$ erhält den Wert 1.
$\Rightarrow F(x)$ ist erfüllbar.

($\leftarrow$) : Sei $F(x)$ erfüllbar.
$\Rightarrow \exists$ Belegung, die F und alle Teilformeln den Wert 1 annehmen läßt.

Insbesondere ist R erfüllt:
$\Rightarrow zust_{t,z}, pos_{t,i}, band_{t,i,a}$ können $\forall t$ als Konfiguration von M interpretiert werden.
Insbesondere ist A erfüllt:

für $t = 0$ kann aus den Variablenwerten die Startkonfiguration von M abgelesen werden.

Insbesondere sind $\ddot{U}_1, \ddot{U}_2$ erfüllt:
$\Rightarrow$ zwischten t und $t+1$ ist die Nachfolgekonfigurationsbedingung erfüllt.
$\Rightarrow \forall t = 0, 1, 2, \ldots$ ist eine mögliche nichtdeterministische Rechnung beschrieben.

Insbesondere ist E erfüllt:
$\Rightarrow$ Rechnung erreicht Endzustand.

Insgesamt gilt also :

$$x \in L(M)$$

Noch zu zeigen: F ist in polynomieller Zeit berechenbar:
Offenbar ist der Aufwand zur Erzeugung von F linear in der Länge von F.

$$\begin{aligned} \text{Wegen } |R| &= O(n^2) \\ |A| &= O(n) \\ |\ddot{U}_1| &= O(n^2) \\ |\ddot{U}_2| &= O(n^2) \\ |E| &= O(1) \\ \\ \text{gilt } |F| &= O(n^2) \end{aligned}$$

■

NP–Berechnungen: "guess and check"
Alle deterministischen Algorithmen zur Berechnung von SAT haben Komplexität $2^{O(n)}$.
(Z.B. systematisches Durchprobieren aller Eingabeformeln).
Da SAT NP–hart ist folgt:

$$NP \subseteq \bigcup_{p \ Polynom} TIME(2^{p(n)})$$

2.4 Weitere NP–vollständige Probleme

2.4.1 $3SAT$

Definition. ($3SAT$)
Gegeben: Boolesche Formel F in KNF mit höchstens 3 Literalen pro Klausel.
Gefragt: Ist F erfüllbar?

Satz.
$3SAT$ ist NP–vollständig.

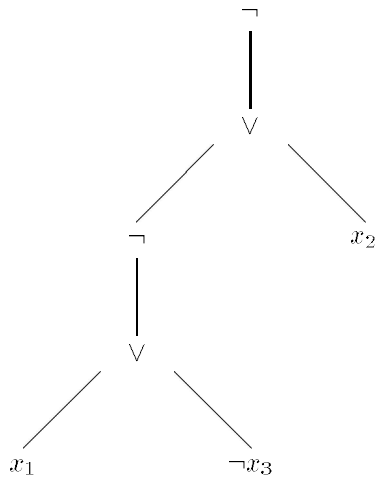
Beweis. 1.) $3SAT \in NP$, klar mit "guess and check"– Argument.
2.) $3SAT$ NP–hart.
Es reicht zu zeigen : $SAT \leq_p 3SAT$.
Angabe eines polynomiellen Verfahrens, das eine beliebige Formel F in eine Formel F' in KNF mit höchstens 3 Literalen pro Klausel umformt mit:

F erfüllbar $\Leftrightarrow$ F' erfüllbar.

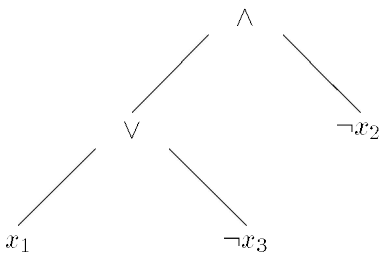
(Dabei genügt "Erfüllbarkeitsäquivalenz". "Äquivalenz ist nicht notwendig.)

Allgemeine Verfahren zur Überführung von F in äquivalente KNF benötigt i.a. exponentielle Zeit und garantiert nicht, daß alle Klauseln höchstens 3 Literale enthalten.

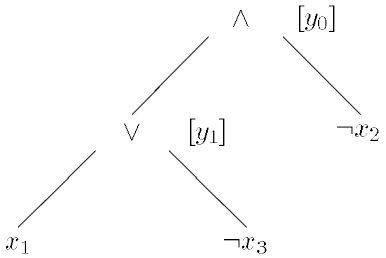
Erläuterung des Verfahrens am Beispiel:



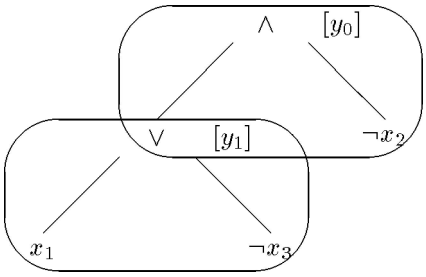
1. Schritt
Mit den DeMorgan'schen Regeln werden alle Negationszeichen zu den Variablen gebracht.



2. Schritt
Jedem inneren Knoten wird eine Variable $\{y_0, y_1, \ldots\}$ zugeordnet, wobei der Baumwurzel y_0 zugeordnet wird.



3. Schritt
Jedem inneren Knoten wird eine Teilformel der Form $(v \leftrightharpoons (y \circ z))$, $\circ \in \{\wedge, \vee\}$ zugeordnet. Man erhält eine neue Formel F_1, indem man alle Teilformeln durch $\wedge$ verknüpft und für die Wurzel y_0 die Teilformel $[y_0]$ hinzunimmt.



Es gilt:

F erfüllbar $\Leftrightarrow F_1$ erfüllbar

(—) : Eine erfüllende Belegung von F liefert eine erfüllende Belegung von F_1.
(—) : Die Belegung der x–Variablen einer erfüllenden Belegung von F_1 liefert eine erfüllende belegung für F.

4. Schritt
Umformung jeder Teilformel in KNF.
Es enstehen Klauseln mit höchstens 3 Literalen.
Das Verfahren ist polynomial, da jede Teilformel in konstanter Länge umgeformt werden kann.

Beispiel $[a \leftrightarrow (b \vee c)] \mapsto (a \vee \neg b) \wedge (\neg a \vee b \vee c) \wedge (a \vee \neg c)$
$[a \leftrightarrow (b \wedge c)] \mapsto (\neg a \vee b) \wedge (\neg a \vee c) \wedge (a \vee \neg b \vee \neg c)$

$\Rightarrow F_1 = y_0 \wedge (\neg y_0 \vee y_1) \wedge (\neg y_0 \vee \neg x_2) \wedge (y_0 \vee \neg y_1 \vee x_2) \wedge (y_1 \vee \neg x_1) \wedge (\neg y_1 \vee x_1 \vee \neg x_3) \wedge (y_1 \vee x_3)$
mit
F_1 ist erfüllbarkeitsäquivalent mit F.

Umformung von F in F_1 ist in polynomieller Zeit möglich. ■

Bemerkung
Für ein analoges Problem gilt:
$2SAT \in P$, da es nur polynomiell viele verschiedene Klauseln mit höchstens 2 Literalen über $\{x_1, .., x_n\}$ gibt.

2.4.2 CLIQUE

Definition. (CLIQUE)
Gegeben: Ein ungerichteter Graph $G = (V, E)$, $k \in I\!N$.
Gefragt: Besitzt G eine "Clique" der Größe k?

Wobei Clique ein vollständiger Teilgraph $G' = (V', E')$ ist, mit

$$(u, v) \in E \quad \forall u, v \in V', \ u \neq v$$

Satz.
CLIQUE ist NP–vollständig.

Beweis.
1.) CLIQUE$\in NP$ mit "guess and check".
2.) CLIQUE ist NP–hart.
Sei F Formel in KNF mit (genau) 3 Literalen pro Klausel.
$F = (z_{1,1} \vee z_{1,2} \vee z_{1,3}) \wedge \ldots \wedge (z_{m,1} \vee z_{m,2} \vee z_{m,3})$ mit $z_{i,j} \in \{x_1, x_2, \ldots\} \wedge \{\neg x_1, \neg x_2, \ldots\}$

Ordne F Graph $G = (V, E)$ und eine Zahl k zu gemäß:

$$\begin{aligned} V &= \{(1,1),(1,2),\ldots,(m,1),(m,2),(m,3)\} \\ E &= \{\{(i,j),(p,q) : i \neq p \text{ und } z_{i,j} \neq \neg z_{p,q}\} \\ k &= m \end{aligned}$$

Es gilt: F ist erfüllbar durch Belegung B.
$\Leftrightarrow$ Jede Klausel hat ein Literal, das unter der Belegung B den Wert 1 annimmt, z.B.: $z_{1,j_1}, z_{2,j_2}, \ldots, z_{m,j_m}$.
$\Leftrightarrow$ Es gibt Literale $z_{1,j_1}, z_{2,j_2}, \ldots, z_{m,j_m}$, die paarweise nicht komplementär sind.
$\Leftrightarrow$ Es gibt Knoten $(1,j_1),(2,j_2),\ldots,(m,j_m)$, die paarweise verbunden sind.
$\Leftrightarrow$ G hat CLIQUE der Größe k. ■

2.4.3 HAMILTON–KREIS

Definition. (GERICHTETER HAMILTON–KREIS)
Gegeben: Ein gerichteter Graph $G = (V, E)$.
Gefragt: Besitzt G einen Hamilton–Kreis?

Wobei ein Hamilton–Kreis eine Permutation der Knotenindizes $(v_{\pi(1)}, \ldots, v_{\pi(n)})$, so daß $(v_{\pi(i)}, v_{\pi(i+1)}) \in E \ \forall i = 1, \ldots, n-1$ und $(v_\pi(n), v_{\pi(1)}) \in E$.

Definition. (UNGERICHTETER HAMILTON–KREIS)
Gegeben: Ein ungerichteter Graph $G = (V, E)$.
Gefragt: Besitzt G einen Hamilton–Kreis?

Satz.
GERICHTETER HAMILTON–KREIS ist NP–vollständig.

Beweis.
GERICHTETER HAMILTON–KREIS $\in NP$ "guess and check".

Noch zu zeigen: GERICHTETER HAMILTON–KREIS NP–hart.
Es wird gezeigt: 3SAT$\leq_p$GERICHTETER HAMILTON–KREIS.

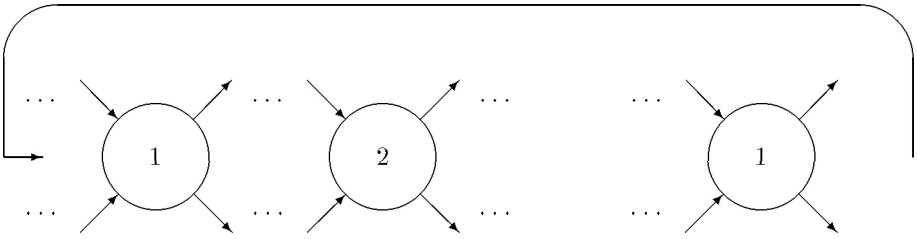
Sei F Formel in KNF mit genau 3 Literalen pro Klausel.

$$\Rightarrow F = (z_{1,1} \vee z_{1,2} \vee z_{1,3}) \wedge \ldots \wedge (z_{m,1} \vee z_{m,2} \vee z_{m,3})$$

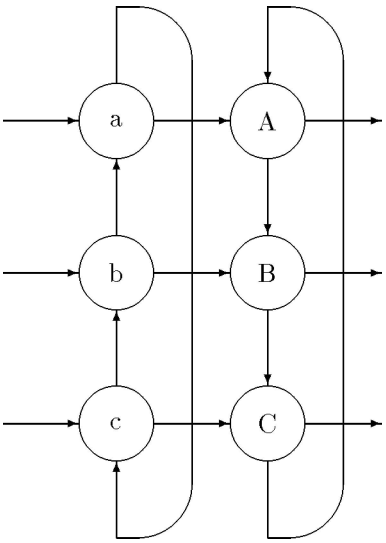
mit

$$z_{i,j} \in \{x_1, \ldots, x_n\} \cup \{\neg x_1, \ldots, \neg x_n\}$$

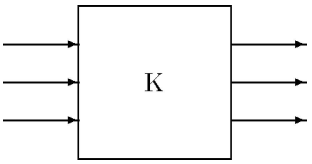
Konstruktion eines gerichteten Graphen:



Vom gleichen Knoten i gehen jeweils 2 Kanten aus.
Die beiden Kanten führen durch folgenden "Klauselgraph" K, von dem m Kopien bereitstehen.



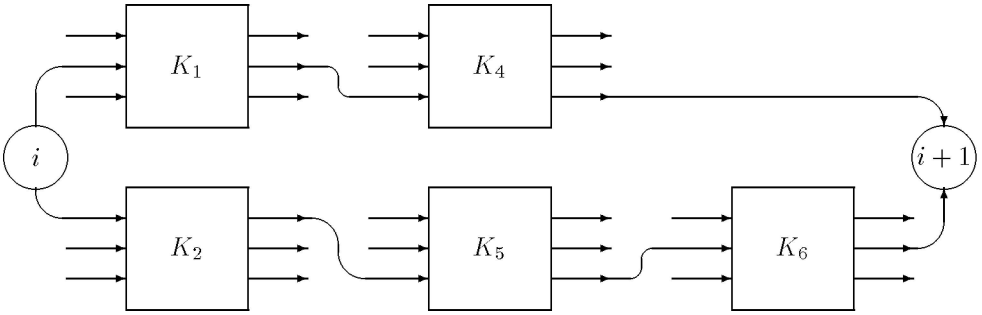
Symbol:



Beispiel.

Literal x_i	kommt in Klausel 1 an Position 2 und in Klausel 4 an Position 3 vor.
Literal $\neg x_i$	kommt in Klausel 2 an Position1 in Klausel 5 an Posiyion 3 und in Klausel 6 an Position 2 vor

Verbindungen von i nach $i+1$:



Beobachtung
Enthält der konstruierte Graph G einen Hamilton–Kreis, so verläßt dieser jedes K wie folgt:
Kommt der Hamilton–Pfad bei $a(b,c)$ an, so verläßt er K in $A(B,C)$.

Beweis.
Annahme: Hamilton–Pfad erreicht K in a und verläßt K nicht in A:
Mögliche Fälle:
$a-A-B$ Sackgasse in b.
$a-A-B-C$ Sackgasse in b.
$a-c-b-B$ A und C sind nicht mehr erreichbar.
$a-c-b-B-C$ A nicht mehr erreichbar.
$a-c-C-A-B$ Sackgasse bei b.

Analog für b und c.

$\Rightarrow$ Ein Hamilton–Pfad kann durch K nur folgende Wege nehmen:
$a-A$
$a-c-C-A$
$a-c-b-B-C-A$

Behauptung.
F erfüllbar $\Leftrightarrow$ G hat Hamilton–Kreis.

Beweis.
(—) : Habe F erfüllende Belegung:
Gilt $x_i = 1$, dann folge dem oberen Pfad von i an.
Gilt $x_i = 0$, dann folge dem unteren Pfad.

Die entsprechenden "Klauselgraphen" werden durchlaufen.

So wird erreicht, daß bei Rückkehr nach 1 alle Konten von G durchlaufen wurden.

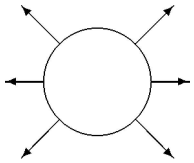
(—) : G besitzt Hamilton–Kreis.
Definiere Variablenbelegung für F:
$x_i = 1$, falls Hamilton–Kreis Knoten i nach "oben" verläßt.
$x_i = 0$, falls Hamilton–Kreis Konten i nach "unten" verläßt.

Die Belegung erfüllt F,da jeder Klauselgraph durchlaufen, entsprechende Klausel also erfüllt wird. ■

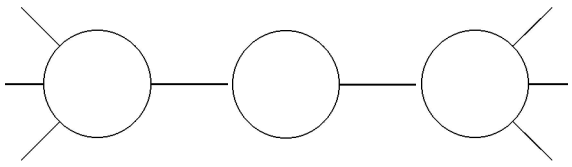
Satz.
UNGERICHTETER HAMILTON–KREIS ist NP–vollständig.

Bewies
Es gilt:
GERICHTETER HAMILTON–KREIS $\leq_p$ UNGERICHTETER HAMILTON–KREIS.

Ersetze in gerichtetem Graphen Knoten der Form



durch ungerichteten Teilgraphen



und erzwinge damit, daß jeder Hamilton–Kreis G_v in Pfeilrichtung durchlaufen wird. ■

Bemerkung
Hamilton–Kreis ("Jeder Knoten wird genau einmal durchlaufen") ist NP–vollständig.
Eulerkreis("Jede Kante wird genau einmal durchlaufen") ist in P ($\rightarrow$ Königsberger Brückenproblem).

Definition. (TRAVELING SALESPERSON Problem)
Gegeben: $n \times n$ Matrix $(M_{i,j})$ von "Entfernungen" zwischen n Städten.
Gefragt: $\exists$ Permutation ("Rundreise") mit

$$\sum_{i=1}^{n-1} M_{\pi(i),\pi(i+1)} + M_{\pi(n),\pi(1)} \leq k?$$

Satz.
Das TRAVELING SALESPERSON Problem ist NP–vollständig.

Beweis.
TRAVELING SALESPERSON Problem$\in NP$ "guess and check".

Noch zu zeigen:
UNGERICHTETER HAMILTON–KREIS $\leq_p$ TRAVELING SALESPERSON Problem:

$$G = (\{1, \ldots, n\}, E) \mapsto \begin{cases} \text{Matrix} M_{i,j} = & \begin{cases} 1 & \{i,j\} \in E \\ 2 & \{i,j\} \notin E \end{cases} \\ \text{Rundreiselänge:} & n \end{cases}$$

■