

NP-Vollständigkeit wichtiger Probleme

Sebastian Bernauer

22.03.2019

Inhaltsverzeichnis

1	Komplexitätsklassen	3
1.1	NP-vollständig	3
1.2	NP-schwierig	3
2	Satisfiability Problem (SAT)	3
2.1	Problembeschreibung	3
2.2	3-SAT	4
2.3	Beweis: $\text{SAT} \leq_p \text{3-SAT}$	4
3	Clique Problem	5
3.1	Problemstellung	5
3.2	Fragestellungen	6
3.3	Beweis: Clique ist NP-vollständig	6
3.3.1	Beweis: Clique ist in NP enthalten	7
3.3.2	Beweis: $\text{SAT} \leq_p \text{Clique}$	7
4	Knapsack Problem (KP)	8
4.1	Problemstellung	8
4.2	Fragestellungen	9
4.3	KP ist NP-vollständig	9
5	Partition Problem	9
5.1	Problemstellung	9
5.2	Beweis: Partition ist NP-vollständig	9
6	Zusammenfassung	10

Abbildungsverzeichnis

1	Ein Graph mit einer Clique der Größe 3	5
2	Ein Graph mit zwei Cliques der Größe 4	6
3	Beispielhafte Darstellung SAT-Problem als Clique-Problem . . .	8
4	Grafische Darstellung eines Knapsack Problems	9

Tabellenverzeichnis

1	Häufige Problemklassen	3
2	Die 3 Fragestellungen bei Problemen	6

1 Komplexitätsklassen

Probleme in der Informatik werden in verschiedene Komplexitätsklassen aufgeteilt.

Der Operator $L_1 \leq_p L_2$ bedeutet, dass das L_1 polynomiell auf L_2 reduzierbar ist. Dies ist der Fall, wenn es eine polynomielle Transformation von L_1 nach L_2 gibt, so das heißt, wenn es eine von einer DTM in polynomieller Zeit berechenbare Funktion $f : \Sigma_1^* \rightarrow \Sigma_2^*$ gibt, so dass für alle $w \in \Sigma_1^*$ gilt:

$$w \in L_1 \leftrightarrow f(w) \in L_2$$

Dieser Operator \leq_p kann nun Probleme in diverse Problemklassen einteilen. In der Tabelle 1 sind häufige Problemklassen dargestellt.

Komplexitätsklasse	Beschreibung
P	Polynomiell lösbar
NP	Nichtdeterministisch polynomiell lösbar
NP-vollständig	Schwierigste Probleme in NP
NP-schwierig	Genauso schwer wie NP-vollständig, das Problem muss aber nicht in NP enthalten sein
nicht rekursiv	Sehr, sehr schwere Probleme

Tabelle 1: Häufige Problemklassen

1.1 NP-vollständig

Ein Problem/Sprache L ist NP-vollständig, wenn folgendes gilt:

- $L \in NP$
- $\forall L' \in NP : L' \leq_p L$

1.2 NP-schwierig

Ein Problem/Sprache L ist NP-schwierig, wenn folgendes gilt:

- $\forall L' \in NP : L' \leq_p L$

2 Satisfiability Problem (SAT)

2.1 Problembeschreibung

Für natürliche Zahlen n und m seien m Klauseln über n Variablen gegeben. Eine Klausel ist die Disjunktion [Veroderung] von einigen Literalen x_i bzw. \bar{x}_i mit $i, j \in \{1, \dots, n\}$. Es soll entschieden werden, ob es eine Belegung $a =$

$\{a_1, \dots, a_n\} \in \{0, 1\}^n$ der Variablen x_1, \dots, x_n gibt, so dass alle Klauseln erfüllt sind.

Eine mögliche Fragestellung für das SAT-Problem ist: Existiert eine Wahrheitsbelegung der Variablen x_1, \dots, x_n , so dass alle Klauseln erfüllt sind?

Der Beweis, dass das SAT-Problem NP-vollständig ist, wird in dem Satz von Cook gezeigt, der in der vorherigen Präsentation gezeigt wurde. Er ist unter [4] im Kapitel 3.3.7 zu finden.

2.2 3-SAT

3-SAT ist ein Spezialfall von SAT, bei denen jede Klausel die Länge 3 hat, also 2 Literale besitzt. Im Folgenden wird bewiesen, dass das allgemeine SAT-Problem durch das 3-SAT Problem abbildbar ist. Durch den Beweis wird gezeigt, dass die zwei Probleme gleich komplex und somit beide NP-vollständig sind (SAT wurde ja durch den Satz von Cook als NP-vollständig bewiesen).

2.3 Beweis: $\text{SAT} \leq_p \text{3-SAT}$

Das grundsätzliche Vorgehen, um SAT-Probleme in 3-SAT-Probleme abzubilden ist, alle Klauseln in neue Klauseln der Länge 3 zu überführen. Haben die ursprüngliche, allgemeinen Klauseln die Länge ≤ 3 , sind die Umformungen trivial:

- Länge 1: Die Klausel besteht aus einem Literal z
Neue Klausel der Länge 3: $z \vee z \vee z$
Durch die Umformung wird die semantische Bedeutung der Klausel nicht verändert, dies könnte z.B. durch eine Wahrheitstabelle bewiesen werden.
- Länge 2: Die Klausel besteht aus 2 Literalen $z \vee y$
Neue Klausel: $z \vee z \vee y$
Hier gilt das Gleiche wie bei der Umformung einer Klausel der Länge 1.
- Länge 3: Die Klausel besteht aus 3 Literalen $z \vee y \vee z$
Neue Klausel: $z \vee y \vee z$
An dieser Stelle besteht die Klausel bereits aus 3 Literalen, es muss keine Umformung durchgeführt werden.

Hat die ursprüngliche Klausel eine Länge ≥ 4 , wird die Umformung schwieriger. Hier wird für das Verständnis zunächst eine beispielhafte Umformung gezeigt, anschließend wird die allgemeine Umformung definiert.

Beispiel: Klausel der Länge $k = 7$ mit den Literalen $z_1 \vee \dots \vee z_k$ wird umgeformt zu folgenden 5 Klauseln der Länge 3:

1. $z_1 \vee z_2 \vee y_1$
2. $\overline{y_1} \vee z_3 \vee y_2$
3. $\overline{y_2} \vee z_4 \vee y_3$

$$4. \overline{y_3} \vee z_5 \vee y_4$$

$$5. \overline{y_4} \vee z_6 \vee z_7$$

Die Literale y_1, \dots, y_4 sind eingeführte "Hilfsvariablen" und werden benötigt, um die Klausel aufzutrennen ohne die Bedeutung zu verändern. Betrachten wir die entstandene Klausel 3: Es ist zu erkennen, dass durch das Literal $\overline{y_2}$ entweder die vorherige Klausel (2) oder die aktuelle Klausel (3) erfüllt werden muss. Dies entspricht dem Operator, der in der ursprünglichen Klausel zwischen den Literalen stand, dem logischen ODER. Das zweite Literal in der neuen Klausel - z_5 - wurde von der ursprünglichen Klausel übernommen. Mit dem dritten und letztem Literal y_4 wurde die ODER-Verknüpfung zu der nächsten entstandenen Klausel geschaffen.

Das war ein konkretes Beispiel, um eine Klausel der Länge $k = 7$ in Klauseln der Länge 3 zu überführen. Die allgemeine Umformung für Klauseln der Länge $k \geq 4$ sieht wie folgt aus: c ist die Nummer der Klausel und wurde eingeführt, um die Hilfsvariablen der Klauseln formal zu trennen.

$$1. z_1 \vee z_2 \vee y_{c,1}$$

$$2. \overline{y_{c,l}} \vee z_{l+2} \vee y_{c,l+1} \text{ für } 1 \leq l \leq k-4$$

$$3. \overline{y_{c,k-3}} \vee z_{k-1} \vee z_k$$

Das Prinzip bleibt das Gleiche, wie bei dem Beispiel mit $k = 7$, die Umformung ist nur allgemein definiert.

Durch die Transformation eines SAT-Problem in ein 3-SAT-Problem wurde bewiesen, dass das 3-SAT-Problem NP-vollständig ist.

3 Clique Problem

3.1 Problemstellung

In einem ungerichteten Graphen $G = (V, E)$ bildet die Knotenmenge $V' \subseteq V$ eine Clique, wenn für alle $v, v' \in V'$ gilt:

$$v, v' \in E$$

In den Abbildungen 1 und 2 auf der nächsten Seite sind 2 Graphen mit Cliquen dargestellt.

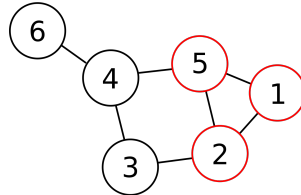


Abbildung 1: Ein Graph mit einer Clique der Größe 3. Bildquelle: [3]

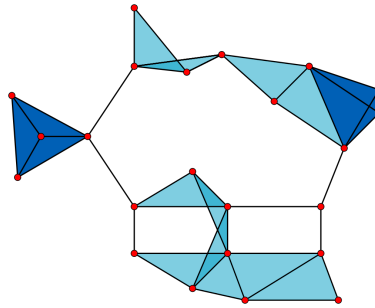


Abbildung 2: Ein Graph mit zwei Cliques der Größe 4 (dunkelblau) und vielen Cliques der Größe 3 (türkis). Bildquelle: [1]

3.2 Fragestellungen

Es gibt für die meisten Probleme 3 Fragestellungen, diese sind in Tabelle 2 aufgeführt.

Fragestellung	Beschreibung
Entscheidungsproblem	Liefert wahr oder falsch
Wertproblem	Liefert eine Zahl
Optimierungsproblem	Liefert die Lösung des Problems, z.B. eine Menge an Knoten

Tabelle 2: Die 3 Fragestellungen bei Problemen

Für das Clique-Problem gibt es die folgenden 3 Fragestellungen:

1. **Entscheidungsproblem:** Gibt es eine Clique der Größe k ?
2. **Wertproblem:** Berechne das größte k , so dass eine Clique der Größe k vorhanden ist.
3. **Optimierungsproblem:** Berechne eine Clique mit dem größten k .

3.3 Beweis: Clique ist NP-vollständig

Der Beweis, dass Clique NP-vollständig ist, teilt sich in 2 Teile auf:

1. Clique ist in NP enthalten
2. $\text{SAT} \leq_p \text{Clique}$

Die Beweise werden in den nächsten 2 Kapiteln durchgeführt.

3.3.1 Beweis: Clique ist in NP enthalten

Das Clique-Problem kann mit einer NTM gelöst werden. Dies geht wie folgt vor:

1. NTM zählt Anzahl n der Knoten im Graphen
2. Rät Wort $w \in \{0, 1\}^n$
3. Das Wort wird als Knotenauswahl interpretiert, V' enthält alle Knoten i mit $w_i = 1$
4. Es wird getestet, ob
 - (a) V' genau k Knoten beinhaltet.
 - (b) G eine Clique auf V' enthält

Der Rechenaufwand hierfür ist polynomiell in der Knotenzahl n . Es wurde bereits bewiesen, dass NTM durch DTM abgebildet werden können. Die polynomielle Laufzeit des Entscheidens, die angegebenen Knoten eine Clique bilden, kombiniert mit dem Nichtdeterminismus ergibt die Problemklasse NP für das Cliquen-Problem.

3.3.2 Beweis: $\text{SAT} \leq_p \text{Clique}$

Konstruiere einen Graphen, der mittels *Clique* ein Problem löst, welches ein *SAT*-Problem ist. Dadurch wird das SAT-Problem in ein Clique-Problem transferiert.

1. Füge für jedes Literal in den Klauseln einen Knoten hinzu.
2. Verbinde alle Literale außer folgende Kanten:
 - Klauselgruppen untereinander
 - Gegensätzliche Literale (z.B. x_1 und \bar{x}_1)
3. Suche eine Clique der Größe k , k ist die Anzahl der Klauseln. Da die Knoten einer Klauselgruppe nicht verbunden sind, muss aus jeder Klausel ein Literal "wahr" sein. Da die Literale in den Klauseln ODER-verknüpft sind, sind alle Klauseln erfüllt.

In Abbildung 3 auf der nächsten Seite ist ein beispielhaftes SAT-Problem als Clique-Graph dargestellt. Der Graph wurde nach den oben genannten Regeln erstellt. Der Beweis wurde [5] entnommen.

Die Transformation von SAT in Clique (Schritte 1 + 2) findet in polynomieller Zeit statt, weshalb gilt: $\text{SAT} \leq_p \text{Clique}$.

Damit das SAT in das Clique-Problem polynomiell abbildbar ist muss folgende Bedingung erfüllt sein:

$$f(w) \in L_2 \leftrightarrow w \in L_1$$

Wenn das Clique-Problem eine Lösung findet, gibt es eine Lösung für das SAT-Problem. Das bedeutet:

$$f(w) \in L_2 \rightarrow w \in L_1$$

Wichtig ist hier der Unterschied, in der oberen Formel die Äquivalenz, in der Unteren die Implikation. Damit die Äquivalenz bewiesen ist, muss zusätzlich zu der Implikation auch folgendes gelten:

$$f(w) \notin L_2 \rightarrow w \notin L_1$$

Da es keine Lösung für das SAT-Problem gibt, wenn das Clique-Problem keine Lösung hat, ist diese Bedingung auch erfüllt.

Damit ist bewiesen: Clique ist NP-vollständig.

Anmerkung: Wenn ein Clique gefunden wird, die über ein Literal keine Aussage (ob dieses Literal wahr oder falsch ist) trifft, so ist der Zustand dieses Literals beliebig wählbar, es gibt also mehrere Lösungen. Würde in der Abbildung 3 weder x_3 noch \bar{x}_3 in der Clique enthalten sein, kann x_3 entweder wahr oder falsch sein. Trotzdem sind alle Klauseln erfüllt.

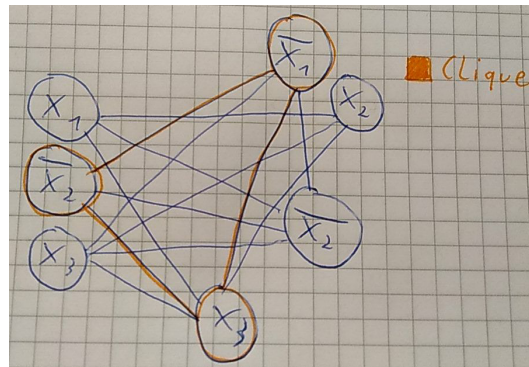


Abbildung 3: Beispielhafte Darstellung SAT-Problem als Clique-Problem

4 Knapsack Problem (KP)

4.1 Problemstellung

Gegeben sind ein Rucksack und n Objekte mit Gewichten $g_1, \dots, g_n \in \mathbb{N}$ sowie eine Gewichtsschranke G . Zusätzlich seien $a_1, \dots, a_n \in \mathbb{N}$ die Nutzenwerte für die Objekte.

Eine grafische Darstellung eines Knapsack Problems ist in Abbildung 4 auf der nächsten Seite abgebildet.

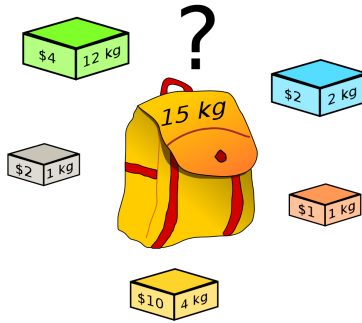


Abbildung 4: Grafische Darstellung eines Knapsack Problems. Bildquelle: [2]

4.2 Fragestellungen

Bei dem KP-Problem gibt es die folgenden 3 Fragestellungen:

1. **Entscheidungsproblem:** Gibt es - unter Beachtung des Gewichtslimits - eine Beladung mit mindestens diesem Nutzwert?
2. **Wertproblem:** Berechne den größtmöglichen Nutzwert x .
3. **Optimierungsproblem:** Berechne die optimale Beladung.

4.3 KP ist NP-vollständig

Der Beweis sei an dieser Stelle vorausgesetzt. Es wird bewiesen, dass $3\text{-SAT} \leq_p \text{KP}$ ist.

Für Interessierte ist er unter [4] im Kapitel 3.4.3 auf Seite 55 zu finden. Der Beweis, dass KP NP-vollständig ist wichtig, da in dem nächsten Beweis darauf aufgebaut wird.

5 Partition Problem

5.1 Problemstellung

Gegeben sind $b_1, \dots, b_n \in \mathbb{N}$. Gibt es eine Teilmenge $I \subseteq \{1, \dots, n\}$, so dass die Summe aller $b_i, i \in I$ gleich der Summe aller $b_i, i \notin I$ ist? Umgangssprachlich wir versucht, eine Menge von Gewichten in 2 gleich schwere Haufen aufzuteilen.

5.2 Beweis: Partition ist NP-vollständig

Es wurde bereits bewiesen, dass ein (spezielles) Knapsack Problem KP^* NP-vollständig ist. (siehe Kapitel 4.3)

(Für a_1, \dots, a_n soll entschieden werden, ob es eine Auswahl gibt, so dass die Summe genau A beträgt).

Nun ist zu beweisen, dass $KP^* \leq_p PARTITION$.

Daraus würde folgen, dass $PARTITION$ NP-vollständig ist.

Sei (a_1, \dots, a_n, A) eine Eingabe für KP^* .

Daraus konstruieren wir in polynomieller Zeit die Eingabe

$(a_1, \dots, a_n, S - A + 1, A + 1)$ für $PARTITION$, wobei S die Summe aller a_i ist.

Falls I eine Lösung für das KP ist, erhalten wir mit $I \cup \{n + 1\}$ eine Lösung für $PARTITION$, da

$$\sum_{i \in I} a_i + S - A + 1 = S + 1 = \sum_{1 \leq i \leq n} a_i + 1 = \sum_{i \notin I} a_i + A + 1$$

Sie Summe aller Zahlen in der Eingabe für $PARTITION$ beträgt $2S + 2$. Ein Lösung für $PARTITION$ muss also so aussehen, dass jeder Teil sich zu $S + 1$ aufsummiert. Damit müssen die Zahlen $S - A + 1$ und $A + 1$ in verschiedenen Teilen sein. $(S - A + 1) + (A + 1) = (S + 2) > (S + 1)$ Die Zahlen, die $S - A + 1$ zu $S + 1$ ergänzen, haben die Summe A und bilden eine Lösung für die Eingabe von KP^* . Der Beweis wurde [4] entnommen.

Damit wurde bewiesen: Partition ist NP-vollständig.

6 Zusammenfassung

Alle behandelten Probleme - SAT, Clique, KP und Partition waren NP-vollständig.

Als ersten Problem wurde mit dem Satz von Cook das SAT-Problem als NP-vollständig bewiesen. Darauf aufbauend wurde das 3-SAT, Clique, KP und Partition-Problem bewiesen. Es ist klar erkennbar, wie die Beweise aufeinander aufbauen, und einem so komplexe Beweise wie der Satz von Cook erspart werden, da auf dem Satz aufgebaut werden kann.

Alle genannten Probleme sind NP-vollständig, somit gelten folgende Bedingungen für sie:

1. $L \in NP$
2. $\forall L' \in NP : L' \leq_p L$

Aus der zweiten Bedingung folgert, dass die Probleme (in polynomieller Zeit) in einander umgewandelt werden können. Besonders anschaulich passiert dies bei der Umwandlung des SAT in das Clique-Problem in Kapitel 3.3.2 auf Seite 7.

Literaturverzeichnis

- [1] David EPPSTEIN. *File:VR complex.svg*. 2007. URL: https://upload.wikimedia.org/wikipedia/commons/d/d0/VR_complex.svg (besucht am 17.03.2019) (siehe S. 6).
- [2] Keenan PEPPER. *File:VR complex.svg*. 2007. URL: https://en.wikipedia.org/wiki/Knapsack_problem#/media/File:Knapsack.svg (besucht am 21.03.2019) (siehe S. 9).
- [3] Erin SILVERSMITH. *Datei:6n-graf-clique.svg*. 2006. URL: <https://upload.wikimedia.org/wikipedia/commons/8/86/6n-graf-clique.svg> (besucht am 17.03.2019) (siehe S. 5).
- [4] Ingo WEGENER. *Theoretische Informatik. Eine algorithmenorientierte Einführung*. Teubner, 2005 (siehe S. 4, 9, 10).
- [5] WEITZ. *Clique ist NP-vollständig*. 2017. URL: <https://www.youtube.com/watch?v=D3gkCTRMckU> (besucht am 25.02.2019) (siehe S. 7).