Inhaltsverzeichnis

3 Komplexitätsbetrachtungen

3.1 Einführung

Beobachtung Es gibt Probleme, für die Polynomialzeit-Algorithmen existieren, z.B:

- Bestimmung des Maximalgrads eines Graphen $G = (V, E)$
- Test, ob ein Graph $G = (V, E)$ bipartit ist.
- Bestimmung des Komplements $\overline{G}$ eines Graphen.

Frage Gibt es für alle Probleme einen Algorithmus, der

(i) das Problem löst?

(ii) das Problem in polynomieller Zeit löst?

Antwort

(i) Nein. Betrachte dazu das sogenannte HALTE-Problem (Turing, 1936):

Frage: Gibt es einen Algorithmus, der entscheidet, ob ein beliebiger anderer Algorithmus terminiert?

Antwort: Einen solcher Algorithmus existiert nicht.

Beweisidee. Angenommen es gäbe einen solchen Algorithmus. Dann hätte er folgende Struktur:

```
HALT (Algorithmus A)
  If(A terminiert) then return true
  Else return false
```

Sei nun Algorithmus A der Form

```
test()
  while(HALT(test())
```

Dann gibt es zwei Fälle:

a) `test()` terminiert $\Rightarrow$ `HALT` liefert true $\Rightarrow$ while-Schleife läuft für immer $\Rightarrow$ `test()` terminiert nicht $\lightning$

b) `test()` terminiert nicht $\Rightarrow$ `HALT` liefert false $\Rightarrow$ while-Schleife terminiert $\Rightarrow$ `test()` terminiert $\lightning$

$\Rightarrow$ `HALT ()` existiert nicht. Das HALTE-Problem ist algorithmisch nicht entscheidbar. □

(ii) offen; führt auf $\mathcal{NP}$-Vollständigkeit (s.u.)

3.2 Komplexitätsklassen

Die Komplexität von Entscheidungsproblemen lässt sich im Wesentlichen in drei Klassen einteilen:

- Es existiert kein Algorithmus für das Problem, d.h. das Problem ist nicht entscheidbar.
- Es existiert ein Polynomialzeit-Algorithmus für das Problem.
- Es sind keine Polynomialzeit-Algorithmen für das Problem bekannt (z.B. nur Exponentialzeit-Algorithmen bekannt), aber es gibt keinen Beweis, dass kein Polynomialzeit-Algorithmus existiert.

Für die Klassifikation der Komplexität von Optimierungsproblemen werden deren Entscheidungsprobleme betrachtet (dies geht auf die Theorie von Turingmaschinen & Sprachen zurück und wird in anderen Vorlesungen behandelt).

Beispiel (VCP)**.**

- VC-Optimierungsproblem:

  Gegeben: Graph $G = (V, E)$

  Gesucht: VC minimaler Kardinalität

- VC-Entscheidungsproblem:

  Gegeben: Graph $G = (V, E)$, natürliche Zahl $k \in \{1, \ldots, |V|\}$

  Frage: Gibt es ein VC C mit $|C| \leq k$?

Wenn das Optimierungsproblem „einfach“, d.h. effizient lösbar ist, so ist auch das Entscheidungsproblem „einfach“. Umgekehrt gilt, wenn das Entscheidungsproblem „schwer“ ist, dann ist das Optimierungsproblem „mindestens genauso schwer“.

Definition (Komplexitätsklassen)**.**

Mit $\mathcal{P}$ bezeichnet man eine Klasse von Entscheidungsproblemen, die in polynomieller Zeit von einer deterministischen Turingmaschine gelöst werden können.

Mit $\mathcal{NP}$ bezeichnet man eine Klasse von Entscheidungsproblemen, die in polynomieller Zeit von einer nicht-deterministischen Turingmaschine gelöst werden können.

Dabei gilt für eine *deterministische Turingmaschine*: Für gegebene Eingabe, sind der momentane Rechenschritt sowie alle folgenden Rechenschritte eindeutig bestimmt, d.h. die Zwischenresultate sind für eine gegebene Eingabe in jedem Durchlauf gleich.
Für eine *nicht-deterministische Turingmaschine* gilt: Anstelle von einer Regel für jeden Rechenschritt bei gegebener Eingabe, sind verschiedene Übergangsregeln möglich (nicht eindeutig bestimmt), um zum nächsten Schritt zu gelangen. In jedem Schritt wird aber der richtige Folgezustand ausgewählt, wenn das Entscheidungsproblem eine Ja-Antwort hat.

Um zu zeigen, dass ein Entscheidungsproblem

- $\in \mathcal{P}$, genügt es, einen Polynomialzeit-Algorithmus anzugeben, der das Problem löst.
- $\in \mathcal{NP}$, genügt es zu zeigen, dass eine korrekte, vorgegebene Lösung in polynomieller Zeit als richtig klassifiziert werden kann.

Es ist klar, dass $\mathcal{P} \subseteq \mathcal{NP}$, aber die Frage, ob $\mathcal{P} = \mathcal{NP}$ oder $\mathcal{P} \neq \mathcal{NP}$ stellt eines der sogenannten Millennium-Probleme dar. Es wird im Allgemeinen angenommen, dass $\mathcal{P} \neq \mathcal{NP}$.

3.3 Prinzip der Reduktion und NP-Vollständigkeit

3.3.1 Polynomialzeitreduktion

Beispiel (Einführendes Beispiel für das Prinzip der Reduktion)**.** Angenommen wir haben ein Problem A unbekannter Komplexität und ein weiteres Problem B, das "einfach" ist, z.B.

Problem A:

Gegeben: Sack voll 1€ Stücke, natürliche Zahl k

Frage: Enthält der Sack genau k Münzen?

Problem B:

Gegeben: Item I, natürliche Zahl L

Frage: Wiegt Item I genau L Gramm?

Sei also eine beliebige Instanz $\alpha \in A$ gegeben, z.B. α = beliebiger Sack mit Münzen und $k = 1223313$. Dann ist die Instanz $\alpha \in A$ äquivalent zu folgender Instanz $\beta \in B$:
Nimm eine Münze und wiege diese (7,5 g).
Die Beantwortung der Frage "Wiegt Item I = (Münzen ohne Sack) genau $L = k \cdot 7{,}5$ g?" beantwortet damit die Frage des Problems A.
Somit kann man die Lösung für jede Instanz von A erhalten, indem man die zugehörige Instanz des Problem B löst. Da Problem B "einfach" ist (mit hinreichend gute Waage), ist auch das Problem A "einfach".

Grundidee Sei Problem $B \in \mathcal{P}$ und Problem A von unbekannter Komplexität. Angenommen es existiert eine „Prozedur“, die jede Instanz α von A in eine Instanz β von B transformiert, sodass

1. die Transformation in polynomieller Zeit erreicht werden kann und

2. die Antworten auf die entsprechenden Entscheidungsprobleme identisch sind, d.h. α hat Ja-Antwort $\Leftrightarrow$ β hat Ja-Antwort,

Als “Bild”:

$$\underbrace{\forall\, \alpha \in A \,\exists\, \beta \in B \text{ sodass } \alpha \leq_p \beta}_{\text{kurz: } A \leq_p B} \xrightarrow{Polynomialzeit-Alg.} \text{Lösung für } \beta = \begin{cases} \text{Ja} \longrightarrow & \text{Ja für } \alpha \\ \text{Nein} \longrightarrow & \text{Nein für } \alpha \end{cases},$$

Somit folgt $A \in \mathcal{P}$. Wir haben die Einfachheit von Problem B genutzt, um die Einfachheit von Problem A zu zeigen, formal:

$$A \leq_p B \; : B \in \mathcal{P} \Rightarrow A \in \mathcal{P}$$

Umgekehrt, falls

$$A \notin \mathcal{P} \text{ und } A \leq_p B \Rightarrow B \notin \mathcal{P},$$

d.h., wenn A nicht in Polynomialzeit gelöst werden kann, kann auch B nicht in Polynomialzeit gelöst werden oder in anderen Worten: Wenn A „schwer“, dann ist B mindestens genauso „schwer“.

Anmerkung. Letztere Grundidee und das Beispiel sollen nur eine “Gedankenstütze” sein. Im Folgenden betrachten wir “$\mathcal{NP}$-Vollständigkeit”, nehmen aber *nicht* an, dass kein Polynomialzeit-Algorithmus existiert.

Die Methodik ist aber ähnlich, d.h. wir werden das Prinzip der Reduktion nutzen, um die „Schwere“ von Problemen zu zeigen: Wenn die Komplexität eines Problems B unbekannt ist, es aber ein „schweres“ Problem A gibt, das sich auf B reduzieren lässt, so muss B mindestens genauso „schwer“ sein wie A.

3.3.2 NP-Vollständigkeit

Definition ($\mathcal{NP}$-vollständig)**.** Ein Entscheidungsproblem D ist *$\mathcal{NP}$-vollständig* $\Leftrightarrow$

1. D in der Klasse $\mathcal{NP}$ liegt, d.h. $D \in \mathcal{NP}$ und

2. D ist $\mathcal{NP}$-schwer, d.h. $\forall\, D' \in \mathcal{NP} : D' \leq_p D$.
   (Jedes Problem D' in $\mathcal{NP}$ kann durch Polynomialzeitreduktion auf D reduziert werden. Das bedeutet auch, dass ein Algorithmus zur Lösung von D dazu genutzt werden kann, um alle anderen Probleme in $\mathcal{NP}$ zu lösen.)

Nachweis der $\mathcal{NP}$-Vollständigkeit von Problemen:

1. Um zu zeigen, dass $D \in \mathcal{NP}$, gibt man eine Lösung an (z.B. durch Raten) und zeigt, dass in polynomieller Zeit verifiziert werden kann, ob die Lösung richtig ist.

2. Um die $\mathcal{NP}$-Schwere von D zu zeigen, nimmt man ein beliebiges $\mathcal{NP}$-vollständiges Problem D' und zeigt, dass es auf das betrachtete Problem D reduziert werden kann, d.h. $D' \leq_p D$. Aus der Transitivität der Polynomialzeitreduktion folgt dann, dass alle anderen Probleme D'' aus $\mathcal{NP}$ ebenfalls auf das betrachtete Problem D reduzierbar sind:
$$\forall\, D'' \in \mathcal{NP} : D'' \leq_p D' \text{gilt: wenn } D' \leq_p D \text{ dann auch } D'' \leq_p D$$

Klar, es muss ein erstes Problem gegeben haben, dessen $\mathcal{NP}$-Schwere bewiesen wurde, das sogenannte SAT-Problem (COOK-LEVIN-THEOREM (1971): SAT ist $\mathcal{NP}$-vollständig). Der Beweis dieses Theorems ist aber außerhalb des Fokus dieser Vorlesung. Das Standartbuch zu diesem Thema ist "Computers and Intractability: A Guide to the Theory of NP-Completeness" von M.R. Garey und D.S. Johnson.

3.4 Das SAT-Problem

Erfüllbarkeitsproblem der Aussagenlogik (SAT, von englisch *satisfiability* = Erfüllbarkeit)

Gegeben: Boolescher Ausdruck φ (oBdA. in konjunktiver Normalform) mit

- AND($\wedge$)-, OR($\vee$)- und NOT-Operationen
- Literalen (Negation oder Nicht-Negation von booleschen Variablen)
- Klammern „ (“, „) “

z.B.
$$\varphi = \underbrace{(x_1 \vee \overline{x_2} \vee x_3)}_{\text{Klausel 1}} \wedge \underbrace{(x_1 \vee \overline{x_4} \vee x_3 \vee x_5)}_{\text{Klausel 2}}$$

Frage: Gibt es eine Wahrheitsbelegung der booleschen Variablen, sodass φ wahr (erfüllbar) ist (dazu muss jede OR-Verknüpfung von Literalen, genannt Klausel, wahr sein.)

Variante: k-SAT: SAT, sodass jede Klausel genau k Literale besitzt, d.h.
$$\varphi = C_1 \wedge C_2 \wedge \ldots \wedge C_n \text{ gilt } |C_i| = k, \text{ also } C_i = l_1^i \vee \ldots \vee l_k^i$$

Theorem. *3-SAT ist $\mathcal{NP}$-vollständig.*

Beweis.

1. 3-SAT ist in $\mathcal{NP}$:
   Sei φ ein Boolescher Ausdruck und die X eine Menge der zugehörigen Booleschen Variablen. Gegeben eine Wahrheitsbelegung $f : X \to \{0, 1\}$, so lässt sich in polynomieller Zeit überprüfen, ob φ wahr.

2. 3-SAT ist $\mathcal{NP}$-schwer:
   Dazu zeigen wir SAT $\leq_p$ 3-SAT (SAT ist $\mathcal{NP}$-vollständig und aus SAT $\leq_p$ 3-SAT folgt damit, dass 3-SAT $\mathcal{NP}$-schwer).
   Konkret zeigen wir: Jede Instanz $\varphi \in$ 3-SAT lässt sich in polynomieller Zeit auf eine Instanz $\varphi' \in$ SAT reduzieren, wobei φ erfüllbar $\Leftrightarrow$ φ' erfüllbar.
   Sei also $\varphi \in$ SAT, $\varphi = C_1 \wedge C_2 \wedge \ldots \wedge C_n$, $|C_i| \in \{1, 2, 3, \ldots, n\}$.
   Nun betrachten wir die folgenden möglichen Fälle:

   - Wenn $|C_i| = 3 \, \forall i \Rightarrow \varphi \in$ 3-SAT, d.h. es ist nichts zu zeigen.
   - Es gibt eine Klausel C_i mit $|C_i| = 2$, z.B. $C_i = (x \vee y)$. Dann führen wir eine neue Variable u und zwei neue Klauseln $C_i' = (x \vee y \vee u)$ und $C_i'' = (x \vee y \vee \overline{u})$ ein, die C_i ersetzen. Damit gilt:
     - C_i wahr $\Rightarrow x = 1$ oder $y = 1 \Rightarrow C_i'$ und C_i'' erfüllbar.
     - C_i' und C_i'' wahr $\Rightarrow \begin{cases} u = 1 \Rightarrow \overline{u} = 0 \\ u = 0 \Rightarrow \overline{u} = 1 \end{cases}$ $\Rightarrow$ in jedem Fall muss $x = 1$ oder $y = 1$ sein $\Rightarrow C_i$ erfüllbar
   - Es gibt eine Klausel C_i mit $|C_i| = 1$, z.B. $C_i = (x)$. Dann führen wir eine neue Variable u ein und ersetzen C_i durch zwei neue Klauseln $C_i' = (x \vee u)$ und $C_i'' = (x \vee \overline{u})$. Es ist klar, dass C_i erfüllbar $\Leftrightarrow$ C_i' und C_i'' erfüllbar. Analog zum Fall $|C_i| = 2$, führen wir nun noch eine Variable v ein und ersetzen

     C_i' durch $\widetilde{C_i} = (x \vee u \vee v)$ und $\widetilde{\widetilde{C_i}} = (x \vee u \vee \overline{v})$

     C_i'' durch $\widehat{C_i} = (x \vee \overline{u} \vee v)$ und $\widehat{\widehat{C_i}} = (x \vee \overline{u} \vee \overline{v})$

     Wie oben folgt C_i' und C_i'' erfüllbar $\Leftrightarrow$ $\widetilde{C_i}, \widetilde{\widetilde{C_i}}, \widehat{C_i}$ und $\widehat{\widehat{C_i}}$ erfüllbar.
   - Betrachte nun noch den Fall $|C_i| > 3$, d.h. $C_i = (x_1 \vee x_2 \vee \ldots \vee v_k)$. Dann führen wir neue Variablen $u_1, \ldots u_{k-3}$ ein und ersetzen C_i durch Klauseln der Form

$$
\begin{aligned}
C_i^1 &= (x_1 \vee x_2 \vee u_1) \\
C_i^2 &= (x_3 \vee \overline{u_1} \vee u_2) \\
C_i^3 &= (x_4 \vee \overline{u_2} \vee u_3) \\
&\vdots \\
C_i^{l-2} &= (x_{l-1} \vee \overline{u_{l-3}} \vee u_{l-2}) \\
C_i^{l-1} &= (x_l \vee \overline{u_{l-2}} \vee u_{l-1}) \\
C_i^l &= (x_{l+1} \vee \overline{u_{l-1}} \vee u_l) \\
&\vdots \\
C_i^{k-3} &= (x_{k-2} \vee \overline{u_{k-4}} \vee u_{k-3}) \\
C_i^{k-2} &= (x_{k-1} \vee x_k \vee \overline{u_{k-3}}).
\end{aligned}
$$

Zeigen nun C_i erfüllbar $\Leftrightarrow$ C_i^j erfüllbar, $j = 1, \ldots, k-2$:

„$\Rightarrow$“ Sei C_i erfüllbar $\Rightarrow$ mindestens ein Literal x_l in C_i ist wahr, d.h. $x_l = 1$.

$$\Rightarrow \text{Setze } \begin{cases} u_j = \text{ wahr }, j < l-2 \\ u_j = \text{ falsch, sonst} \end{cases}$$

$\Rightarrow$ Alle C_i^j erfüllbar, $j = 1, \ldots, k-2$. (siehe Tafel oder nachrechnen)

„$\Leftarrow$“ Seien alle C_i^j erfüllbar, $j = 1, \ldots, k-2$. Angenommen C_i ist nicht erfüllbar.
$\Rightarrow x_i = 0$ für alle $i = 1, \ldots, k-2$
$\Rightarrow$ insbesondere $x_{k-1} = 0$ und $x_k = 0$
Da aber alle C_i^j erfüllbar, $j = 1, \ldots, k-2$, folgt $\overline{u_{k-3}} = 1$, also $u_{k-3} = 0$
$\Rightarrow u_{k-4} = 0$ usw.
$\Rightarrow$ alle $u_i = 0$, $i = 1, \ldots, k-3$
$\Rightarrow C_i^1$ nicht erfüllbar $\lightning$
Also war die Annahme falsch, d.h. C_i muss erfüllbar sein.

□

3.5 Weitere NP-Vollständige Probleme

3.5.1 Cliquen-Problem

CLIQUE-Problem

Gegeben: Graph $G = (V, E)$ natürliche Zahl $L \in \{1, \ldots, |V|\}$
Frage: Existiert ein Teilgraph $H \subseteq G$, sodass $H \cong K_j$, $j \geq L$
(d.h. H ist vollständiger Teilgraph mit $j \geq L$ Knoten)

Theorem. *CLIQUE ist $\mathcal{NP}$-vollständig.*

Beweis.

1. CLIQUE $\in \mathcal{NP}$ (Prüfe bei k gegebenen Knoten in polynomieller Zeit, ob zwischen je zwei paarweise verschiedenen Knoten eine Kante existiert.)

2. CLIQUE ist $\mathcal{NP}$-schwer. Dazu zeigen wir 3-SAT $\leq_p$ CLIQUE:
   Konkret transformieren wir eine 3-SAT Formel φ in einen Graphen $G_\varphi = (V, E)$ und eine Zahl $k \in \mathbb{N}$, sodass gilt: φ erfüllbar $\Leftrightarrow$ G hat eine k-Clique. Dazu

   - Seien $C_1, \ldots, C_k$ die Klauseln von φ.
   - Seien l_1^i, l_2^i und l_3^i die Literale in Klausel C_i.
   - Identifiziere Literale und Knoten. d.h. setze

   $$V = \{l_i^j \mid 1 \leq j \leq k,\ 1 \leq i \leq 3\}.$$

   - Jedes Knotenpaar l_r^i und l_s^j wird durch eine Kante verbunden, außer

a) die assoziierten Literale gehören zur selben Klausel (d.h. es muss gelten $i \neq j$)

b) eines der beiden Literale ist die Negation des anderen Literals (d.h. l_r^i und l_s^j sind nicht durch eine Kante verbunden, wenn z.B. $l_r^i = x$ und $l_s^j = \overline{x}$).

Wir zeigen nun die Korrektheit der Transformation:

(i) φ erfüllbar $\Rightarrow$ G hat eine k-Clique:
Da φ erfüllbar, beinhaltet jede Klausel C_i mindestens ein wahres Literal l_r^i, $r \in \{1, 2, 3\}$. Wähle pro Klausel eines der wahren Literale beliebig aus. Die damit assoziierten k Knoten bilden eine k-Clique, da:

- Alle ausgewählten Literale gehören zu verschiedenen Klauseln. Es kann also keine Kante aufgrund von Regel a) fehlen.
- Alle ausgewählten Literale werden gleichzeitig erfüllt, widersprechen sich also nicht. Es kann somit auch keine Kante wegen Regel b) fehlen.

(ii) G hat eine k-Clique $\Rightarrow$ φ erfüllbar:

- Wenn G eine k-Clique hat, so müssen aufgrund von Regel a), die Knoten in dieser Clique zu verschiedenen Klauseln gehören.
- Die k-Clique wählt somit ein Literal pro Klausel aus.
- Diese Literale können alle gleichzeitig auf "1" gesetzt werden, da sie sich aufgrund von Regel b) nicht widersprechen.
- Also ist φ erfüllbar, da mindestens ein Literal für jede Klausel auf wahr gesetzt wurde.

□

3.5.2 Vertex-Cover-Problem

Theorem. *VCP ist $\mathcal{NP}$-vollständig.*

Beweis.

1. VCP $\in \mathcal{NP}$ (Es lässt sich in polynomieller Zeit überprüfen, ob eine vorgegebene Knotenmenge $C \subseteq V$ tatsächlich ein *vertex cover* bildet, indem man überprüft, ob für alle Kanten aus E mindestens ein Endpunkt in C liegt.)

2. VCP ist $\mathcal{NP}$-schwer. Dazu zeigen wir CLIQUE $\leq_p$ VCP. Insbesondere zeigen wir:
G hat Clique V' der Größe $k \Leftrightarrow \overline{G}$ hat VC $C = V \setminus V'$ der Größe $|V| - k$.

   „$\Rightarrow$" Sei $V' \subseteq V$ eine Clique in G,
   d.h. für den (Knoten)induzierten Teilgraph $\langle V' \rangle$ gilt $\langle V' \rangle \simeq K_k$
   Angenommen $C = V \setminus V'$ ist kein VC.
   $\Rightarrow$ $\exists$ Kante $(uv) \in E(\overline{G})$, sodass weder u noch v in C.
   $\Rightarrow$ $u, v \in V'$. Da aber $(uv) \in E(\overline{G}) \Rightarrow (uv) \notin E(G)$

$\Rightarrow V'$ ist keine Clique ↯.

Klar, wenn V' Clique der Größe k,
dann ist somit $V \setminus V'$ ein VC der Größe $|V| - k$.

„⇐“ Sei $C = V \setminus V'$ ein VC in $\overline{G}$.
$\Rightarrow \forall (uv) \in E(\overline{G})$ gilt $u \in V \setminus V'$ oder $v \in V \setminus V'$
$\Rightarrow$ mindestens einer der Knoten u und v ist nicht in V'
$\Rightarrow$ also gilt für alle Knoten $x, y \in V'$, dass $(xy) \notin E(\overline{G})$
$\Rightarrow (xy) \in E(G)$
$\Rightarrow V'$ ist Clique in G.

Klar, wenn $V \setminus V'$ ein VC der Größe $|V| - k$,
dann ist somit V' eine Clique der Größe k.

□