Berechenbarkeit und Komplexität - WS17/18

Lukas Glänzer

Inhaltsverzeichnis

1 Grundlagen

Probleme/Berechnungsprobleme/Algorithmen sind Relationen

- $R \subset \Sigma^* \times \Sigma'^*$ mit $(x, y) \in R$, wenn y eine richtige Ausgabe zur Eingabe x ist
- Gibt es zu einer Eingabe eine eindeutige Ausgabe, dann lässt sich das Problem durch die Funktion $f : \Sigma^* \to \Sigma'^*, x \mapsto f(x)$ beschreiben
- Probleme der Form $f : \Sigma^* \to \{0, 1\}$ heißen Entscheidungsprobleme
- Sprachen sind Teilmengen von Σ^*

1.1 Turingmaschinen

Eine Turingmaschine ist ein mathematisches Modell der Algorithmik und ist definiert als 7-Tupel:

- $(Q, \Sigma, \Gamma, B, q_0, \overline{q}, \delta)$
  - Q ist die endliche Zustandsmenge
  - Σ ist das endliche Eingabealphabet
  - Γ ist das endliche Bandalphabet
  - $B \in \Gamma \setminus \Sigma$ ist das Bandzeichen/Leerzeichen
  - $q_0 \in Q$ ist der Startzustand
  - $\overline{q} \in Q$ ist der Endzustand
  - $\delta : (Q \setminus \{\overline{q}\}) \times \Gamma \to Q \times \Gamma \times \{R, L, N\}$ ist die endliche **deterministische** Übergangsfunktion
- Bei einem Rechenschritt wird aus aktuellem Zustand und dem Symbol unter dem Kopf ein neuer Zustand mittels δ ermittelt:
  - $\delta(q, a) = (q', a', d)$, wo bei q' der Folgezustand, a' das geschriebene Symbol und d die Richtung des Kopfes sind
  - Die verschiedenen Gesamtzustände einer TM heißen Konfigurationen $\alpha q \beta$ mit $\alpha, \beta \in \Gamma^*$ und $q \in Q$:
    - Der Bandinhalt ist $\alpha\beta$ und der Kopf ist über β_1
    - $\alpha q \beta \vdash \alpha' q' \beta'$, wenn $\alpha' q' \beta'$ eine direkte Nachfolgekonfiguration ist, sonst $\vdash^*$
- Die TM terminiert, wenn sie $\overline{q}$ erreicht. Die Ausgabe ist das Wort auf dem Band
  - Bei Entscheidungsproblemen nur das erste Zeichen: 1 für Ja, 0 für Nein
- Die Laufzeit einer TM ist die Anzahl der Zustandübergänge bis zur Terminierung
- Der Speicherbedarf einer TM ist die Anzahl der besuchten Bandzellen während der Berechnung

Alle entscheidbaren Probleme und Funktionen lassen sich durch eine TM berechnen, daher gilt:

- $f : \Sigma^* \to \Sigma^*$ heißt rekursiv, wenn eine TM existiert, die zu x den Funktionswert $f(x)$ berechnet
- $L \subset \Sigma^*$ heißt rekursiv, wenn eine TM existiert, die w genau dann akzeptiert, wenn $w \in L$

K-Spurige Turingmaschinen sind TM, deren Band in mehrere Spuren geteilt ist. Der Kopf kann an jeder Position nun die k Werte gleichzeitig auslesen

- K-Spur TM sind gleichmächtig zu Einspurigen TM und lassen sich in einander transformieren

K-Band Turingmaschinen sind TM, die mehrere Bänder mit jeweils eigenem Lese-/Schreibkopf besitzen

- Die Übergangsfunktion ist nun: $\delta : (Q \setminus \{\overline{q}\}) \times \Gamma^k \to Q \times \Gamma^k \times \{L, R, N\}$

- k-Band TM sind wiederum gleichmächtig zu 1-Band Turingmaschinen und lassen sich ineinander überführen
  - Hat die k-Band-TM M Laufzeit $t(n)$ und Platzbedarf $s(n)$, so hat 1-Band-TM M' eine Laufzeit von $\mathcal{O}(t^2(n))$ und einen Platzbedarf von $\mathcal{O}(s(n))$

Eine universelle Turingmaschine ist eine general purpose Turingmaschine, die andere Turingmaschinen simulieren kann. Es wird also nicht mehr für jedes Problem eine eigene TM benötigt.

- Eingabe der universellen TM ist ein String der Form $\langle M \rangle w$ und simuliert M auf w
  - $\langle M \rangle$ ist die Gödelnummer der TM M und w die Eingabe für M
  - Simulation auf 3 Bändern: Erstes simuliert Band von M, das Zweite speichert die Gödelnummer und das Dritte den aktuellen Zustand
  - Nur konstanter Zeitverlust durch die Simulation

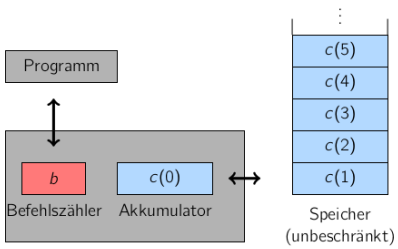
1.1.1 Non-deterministische Turingmaschinen

Eine NTM ist analog zur bisherigen DTM definiert, nur, dass die Zustandsübergänge nicht mehr über einer Funktion, sondern über eine Relation beschrieben werden:

- $\delta \subseteq ((Q \setminus \{\overline{q}\}) \times \Gamma) \times (Q \times \Gamma \times \{L, R, N\})$
- Es können zu einer Konfiguration also mehrere direkte Nachfolgekonfigurationen existieren
- Rechenwege können in Berechnungsbaum dargestellt werden
  - Der maximale Verzweigungsgrad Δ ist größte Anzahl an direkten Nachfolgekonfigurationen
- Die Laufzeit $T_M(w)$ einer NTM M für Eingabe w ist definiert als:
  - $w \in L(M)$: die Länge des kürzesten akzeptierenden Rechenweges
  - $w \notin L(M) : 0$
- Worst-Case-Laufzeit der NTM M auf Eingaben der Länge n ist $t_M(n) := \max\{T_M(x) \mid x \in \Sigma^n\}$

1.2 Registermaschinen

Registermaschinen sind ein alternatives Modell zur Turingmaschine. Eine RAM besitzt einen unbeschränkten Speicher mit einzeln adressierbaren Registern $c(i)$. Mit einem Befehlssatz werden Programme geschrieben, die die Inhalte der Register so manipulieren, dass ein Algorithmus/eine Funktion berechnet werden können. Der Akkumulator $c(0)$ bietet ein Referenzregister, mit dem andere Register verglichen werden können und Werte manipuliert werden können. Die Eingabe einer RAM ist eine initiale Registerbelegung. Die Ausgabe ist die Registerbelegung nach Terminierung.



Befehlssatz:
LOAD, STORE, ADD, SUB, MULT, DIV
INDLOAD, INDSTORE, INDADD, INDSUB, INDMULT, INDDIV
CLOAD, CADD, CSUB, CMULT, CDIV
GOTO, IF c(0)?x THEN GOTO j (mit ? aus {=,<,<=,>,>=}),
END

Jede RAM, die (im logarithmischen Kostenmaß), zeitlich durch $t(n)$ beschränkt ist, existiert eine TM M mit Laufzeit $\mathcal{O}(q(n + t(n)))$. Andersherum lässt sich jede TM mit Laufzeit $t(n)$ durch eine RAM in $\mathcal{O}(t(n) + n)$ bzw. $\mathcal{O}(t(n) + n) \cdot \log(t(n) + n)$ im logarithmischen Kostenmaß simulieren

1.3 Church-Turing-These

Die Klasse der TM-berechenbaren Funktionen stimmt mit der Klasse der „intuitiv berechenbaren" Funktionen überein.

2 Berechenbarkeit

2.1 Abzählbarkeit

Eine Menge M heißt abzählbar, wenn $M = \emptyset$ oder ein surjektives $c : \mathbb{N} \to M$ existiert

- Endliche Mengen sind abzählbar
- Abzählbar unendliche Mengen haben die gleiche Mächtigkeit wie $\mathbb{N}$
- Sprachen über einem endlichen Alphabet sind abzählbar
- Die Potenzmenge $\mathcal{P}(M)$ einer abzählbar unendlichen Menge M ist überabzählbar

Ein Entscheidungsproblem ist eine Teilmenge von $\{0,1\}^*$, also ist die Menge aller Entscheidungsprobleme $\mathcal{P}(\{0,1\}^*)$ überabzählbar. Da die Menge aller Turingmaschinen abzählbar ist, muss es Probleme geben, die nicht TM-berechenbar, also nicht rekursiv/entscheidbar sind.

2.2 Entscheidbarkeit und Semi-Entscheidbarkeit

Ist L ein entscheidbares/rekursives Problem, dann gilt:

- Es existiert eine TM M: M hält auf jeder Eingabe w und akzeptiert g.d.w. $w \in L$
- M entscheidet L

Ist L ein semi-entscheidbares/rekursiv aufzählbares Problem, dann gilt:

- Es existiert eine TM M: M akzeptiert die Eingabe w g.d.w. $w \in L$ und verhält sich ansonsten beliebig ($\to$ Verwerfen oder nicht terminieren)
- M erkennt L

Jede Sprache L wird genau einer der folgenden 4 Familen zugeordnet:

- L ist entscheidbar $\to$ L und $\overline{L}$ sind semi-entscheidbar
- L ist semi-entscheidbar, aber $\overline{L}$ nicht
- L ist nicht semi-entscheidbar, aber $\overline{L}$
- Weder L noch $\overline{L}$ sind semi-entscheidbar

2.3 Verschiedene Probleme

2.3.1 Die Diagonalsprache

$D = \{w \in \{0,1\}^* \mid w = w_j \text{ und } M_j \text{ akzeptiert } w \text{ nicht}\}$

- D ist nicht entscheidbar (Beweis über Diagonalisierung)

2.3.2 Das Halteproblem

$H = \{\langle M \rangle w \mid M \text{ hält auf } w\}$

- H soll entscheiden, ob M gestartet auf der Eingabe w terminiert
- H ist nicht entscheidbar (Beweis über Diagonalisierung)
- H ist semi-entscheidbar, aber $\overline{H}$ ist nicht semi-entscheidbar

2.3.3 Das spezielle Halteproblem

$H_\varepsilon = \{\langle M \rangle \mid M \text{ hält auf } \varepsilon\}$

- H_ε soll entscheiden, ob M gestartet auf der leeren Eingabe terminiert
- H_ε ist nicht entscheidbar
- H_ε ist semi-entschiedbar, aber $\overline{H_\varepsilon}$ ist nicht semi-entscheidbar

2.3.4 Das totale Halteproblem

$H_{tot} = \{\langle M \rangle \mid M \text{ hält auf jeder Eingabe } w \in \{0,1\}^*\}$

- H_{tot} soll entscheiden, ob M immer terminiert
- H_{tot} ist nicht entscheidbar
- H_{tot} und $\overline{H_{tot}}$ sind nicht semi-entscheidbar

2.3.5 Das Collatzproblem

Berechne die durch die Iterationsvorschrift $x \leftarrow \begin{cases} \frac{x}{2} & \text{,wenn x gerade} \\ 3x+1 & \text{,wenn x ungerade} \end{cases}$ definierte Folge

- Es ist nicht bekannt, ob das Problem auf allen Eingaben hält (1 erreicht)

2.3.6 Das Postsche Korrespondenzproblem

Gegeben sei eine Menge K an Dominosteinen. Jeder Dominostein hat zwei nicht leere Wörter, oben x und unten y, über einem Alphabet Σ. Aufgabe ist es eine nicht leere Folge von Dominos zu finden, sodass oben und unten jeweils das gleiche Wort steht.

- Das MPCP (modifizierte PCP) ist eine Modifikation, in der der erste Stein der Menge K auch der Startdomino der Folge sein muss
- Das PCP und das MPCP i.A. sind unentscheidbar
- Jede Turingmaschine lässt sich in ein PCP/MPCP überführen
- Wenn $|K| \leq 2$ oder $|x_i| = |y_i| = 1\ \forall i$ ist das PCP/MPCP entscheidbar, sonst nicht

2.3.7 Kontextfreie Grammatiken

CFGs sind 4-Tupel $G = (N, \Sigma, P, S)$. $L(G)$ sind alle von S mit Produktionsregeln aus P ableitbaren Wörter

- Entscheidbare Probleme sind: Ist $w \in G$?, Ist $L(G) = \emptyset$?, Ist $L(G)$ endlich?
- Unentscheidbar sind: Ist G eindeutig?, $L(G) = \Sigma^*$?, Ist $L(G)$ regulär?, Ist $L(G_1) \subset L(G_2)$?, Ist $L(G_1) \cap L(G_2) = \emptyset$?

2.3.8 Hilberts 10. Problem: Diophantische Gleichungen

Diophantische Gleichungen sind ganzzahlige Polynome in einer oder mehreren Variablen, zu denen ganzzahlige Nullstellen gesucht werden

- Dioph ist unentscheidbar
- Dioph ist semi-entscheidbar

2.4 Beweistechniken zur (Semi-)Entscheidbarkeit

Soll für eine Sprache L gezeigt werden, dass sie entscheidbar ist, so kann man einfach einen entsprechender Algorithmus angegeben, oder:

- Ist L entscheidbar, dann auch $\overline{L}$
- Sind L und $\overline{L}$ semi-entscheidbar, dann ist L entscheidbar
- Sind L_1 und L_2 (semi-)entscheidbar, dann ist auch $L_1 \cap L_2$ entscheidbar
- Sind L_1 und L_2 (semi-)entscheidbar, dann ist auch $L_1 \cup L_2$ entscheidbar
- Finde Zertifikat und Verifizierer, sodass gezeigt ist $L \in NP$ (siehe 3.1)

Ist L jedoch unentscheidbar, so kann dies wie folgt gezeigt werden:

- Ist L unentscheidbar, dann auch $\overline{L}$
- Ist L oder $\overline{L}$ nicht semi-entscheidbar, dann kann L nicht entscheidbar sein
- Unterprogrammtechnik/Reduktion/Satz von Rice

Für eine Sprache L soll gezeigt werden, dass sie semi-entscheidbar ist:

- Gebe einen Aufzähler von L an
- Gebe eine nichtdeterministische TM an, die L entscheidet
- Reduktion

2.4.1 Unterprogrammtechnik

Für eine Sprache L soll gezeigt werden, dass sie unentscheidbar ist:

- Wähle bereits bekannte unentscheidbare Sprache L'
- Nehme an, dass L entscheidbar ist und führe dies zum Widerspruch
- Verwende L und beliebige **berechenbare** Operationen um L' zu lösen
  - Beliebige berechenbare Manipulation der Eingaben und Ausgaben
  - Mehrfache Anwendung von L
  - ...
- Gelingt dies, muss L ebenfalls unentscheidbar sein

Die Unterprogrammtechnik eignet sich nicht für Semi-Entscheidbarkeit!

2.4.2 Reduktion

Für L_1 und L_2 über Σ heißt L_1 auf L_2 reduzierbar ($L_1 \leq L_2$), wenn eine berechenbare Funktion $f : \Sigma^* \to \Sigma^*$ mit $x \in L_1 \Leftrightarrow f(x) \in L_2$ existiert

- Falls $L_1 \leq L_2$ und L_2 entscheidbar/semi-entscheidbar ist, dann ist auch L_1 entscheidbar/semi-entscheidbar
- Falls $L_1 \leq L_2$ und L_1 unentscheidbar/nicht semi-entscheidbar ist, dann ist auch L_2 unentscheidbar/nicht semi-entscheidbar
- Falls $L_1 \leq L_2$ und $L_2 \leq L_3$ gilt auch $L_1 \leq L_3$
- Falls $L_1 \leq L_2$ gilt auch $\overline{L_1} \leq \overline{L_2}$

- Reduktionen sind eine spezielle Art der Unterprogrammtechnik
  - Um die Unentscheidbarkeit von L zu zeigen, wähle unentscheidbare Sprache L'
  - Modifiziere die Eingabe von L' **nur** mittels berechenbarer Funktion f
  - Führe L **genau einmal** auf $f(w)$ aus und übernehme das Ergebnis

2.4.3 Satz von Rice

Es sei $\mathcal{R}$ die Menge der von TM berechenbaren partiellen Funktionen. Es sei $\mathcal{S}$ eine Teilmenge von $\mathcal{R}$ mit $\emptyset \subsetneq \mathcal{S} \subsetneq \mathcal{R}$. Dann ist die Sprache $L(\mathcal{S}) = \{\langle M \rangle \mid M \text{ berechnet eine Funktion aus } \mathcal{S}\}$ nicht entscheidbar.

- TM berechnen partielle Funktionen mit 3 möglichen Ausgängen:
  - $0 \mathrel{\widehat{=}}$ Verwerfen
  - $1 \mathrel{\widehat{=}}$ Akzeptieren
  - $\perp \mathrel{\widehat{=}}$ Nicht Terminieren
- Nicht-triviale Aussagen über das Verhalten einer TM sind nicht entscheidbar
- **Wichtig:** Der Satz von Rice trifft nur Aussagen über die Funktion einer TM
- Es kann nur eine Aussage über Unentscheidbarkeit getroffen werden

2.5 Turing-Mächtigkeit

Ein Rechnermodell wird als Turing-mächtig bezeichnet, wenn sie jede Funktion, die von einer Turingmaschine berechnet werden kann, ebenfalls berechnen kann.

- RAMs sind Turing-mächtig, auch endlicher Anzahl Register und eingeschränktem Befehlssatz:
  - `LOAD, STORE, CLOAD, CADD, CSUB, GOTO, IF c(0)>0 THEN GOTO, END`
- Minesweeper, Game of Life, Programmiersprachen,... sind Turing-mächtig

2.5.1 LOOP-Programme

LOOP ist eine minimale Programmiersprache mit 2 Arten von Befehlen - Zuweisungen und Schleifen:

- $x_i := x_j \pm c$ mit $c \in \mathbb{N}$
- LOOP x_i DO P ENDLOOP
- Jede Hintereinanderausführung $P_1; P_2$ ist ebenfalls ein LOOP-Programm
- Die Eingabe des Programms befindet sich in den Variablen $x_1, ..., x_n$
- Die Ausgabe steht am Ende in x_1
- Besonderheiten:
  - Alle x_i sind in $\mathbb{N}$, d.h. bei Subtraktion werden negative Ergebnisse auf 0 gesetzt
  - Bei LOOP x_i DO P ENDLOOP ist nur der initiale Wert von x_i relevant
  - LOOP-Programme terminieren immer
- In der VL behandelte und verwendbare Macros:
  - ADD, MULT, DIV, MOD, IF $x_i = c$ THEN P_1 ELSE P_2 ENDIF
- LOOP ist nicht Turing-mächtig
- LOOP-berechenbare Funktionen sind gerade die primitiv rekursiven Funktionen

2.5.2 WHILE-Programme

WHILE ist Programmiersprache mit 2 Arten von Befehlen - Zuweisungen und Schleifen:

- $x_i := x_j \pm c$ mit $c \in \mathbb{N}$
- WHILE $x_i \neq 0$ DO P ENDWHILE
- Jede Hintereinanderausführung $P_1; P_2$ ist ebenfalls ein WHILE-Programm
- Die Eingabe des Programms befindet sich in den Variablen $x_1, ..., x_n$
- Die Ausgabe steht am Ende in x_1
- Besonderheiten:
  - Alle x_i sind in $\mathbb{N}$, d.h. bei Subtraktion werden negative Ergebnisse auf 0 gesetzt
  - Bei WHILE $x_i \neq 0$ DO P ENDWHILE kannder Wert von x_i während des durchlaufs verändert werden
  - WHILE kann LOOP-Programme simulieren
  - Es gibt WHILE-Programme, die nicht terminieren
- WHILE ist Turing-mächtig

2.5.3 Die Ackermann Funktion

Die Ackermann Funktion $A : \mathbb{N}^2 \to \mathbb{N}$ ist definiert als:

$$\begin{array}{lcll} A(0,n) & = & n+1 & \text{für } n \geq 0 \\ A(m+1,0) & = & A(m,1) & \text{für } m \geq 0 \\ A(m+1,n+1) & = & A(m,A(m+1,n)) & \text{für } m,n \geq 0 \end{array}$$

- Entspricht wiederholter Potenzierung: Potenzturm mit Symbol $a \uparrow\uparrow b$
- Die Ackermann Funktion ist berechenbar, aber nicht LOOP-berechenbar

3 Komplexität

3.1 P und NP

Definition der Komplexitätsklasse P:

- P ist die Klasse aller Entscheidungsprobleme für die es einen polynomiellen Algorithmus gibt
- Alle Probleme, die in polynomieller Zeit von einer TM oder RAM entschieden werden können
  - Probleme in P werden effizient gelöst

Definition der Komplexitätsklasse NP:

- NP ist die Klasse aller Entscheidungsprobleme, die durch eine NTM M erkannt werden und deren Worst-Case-Laufzeit $t_M(n)$ polynomiell beschränkt ist
- Alternativ: Die Sprache L ist genau dann in NP, wenn es einen polynomiellen deterministischen Algorithmus V und ein Polynom p gibt, mit:
  - $x \in L \Leftrightarrow \exists y \in \{0,1\}^*,\ |y| \leq p(|x|) :\ V$ akzeptiert $y\#x$
  - V heißt Verifizierer und y heißt Zertifikat

3.2 NP Probleme

3.2.1 SAT - Satisfiability

Eingabe: Eine Bool'sche Formel φ in CNF

- Frage: Existiert eine Belegung X, die φ erfüllt?

3.2.2 3-SAT - 3-Satisfiability

Eingabe: Eine Bool'sche Formel φ in CNF in der jede Klausel Länge 3 hat

- Frage: Existiert eine Belegung X, die φ erfüllt?

3.2.3 CLIQUE

Eingabe: Ein ungerichteter Graph $G = (V, E)$ und eine Zahl k

- Frage: Enthält G eine k-Clique?

3.2.4 Independent Set

Eingabe: Ein ungerichteter Graph $G = (V, E)$ und eine Zahl k

- Frage: Enthält G eine unabhängige Menge der Größe k?
- Optimierung: Finde maximales k

3.2.5 VC - Vertex Cover

Eingabe: Ein ungerichteter Graph $G = (V, E)$ und eine Zahl k

- Frage: Enthält G ein Vertex-Cover mit k Knoten?

3.2.6 HAM-CYCLE - Hamiltonkreis

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

- Frage: Besitzt G einen Hamiltonkreis?

3.2.7 HAM-PATH - Hamiltonpfad

Eingabe: Ein ungerichteter Graph $G = (V, E)$ und $u, v \in V$

- Frage: Besitzt G einen Hamiltonpfad von u nach v?

3.2.8 TSP - Travelling Salesman Problem

Eingabe: Städte/Knoten $1, ..., n$, Distanzen $d(i, j)$ und eine Zahl γ

- Frage: Gibt es eine Rundreise/TSP-Tour mit Länge höchstens γ?
- Optimierung: Finde minimales γ

3.2.9 EX-COVER - Exact Cover

Eingabe: Eine endliche Menge X und Teilmengen $S_1, ...S_m$ von X

- Frage: Ex. eine Indexmenge $I \subseteq \{1, ..., m\}$, sodass alle S_i mit $i \in I$ eine Partition von X bilden?

3.2.10 SUBSET-SUM

Eingabe: Positive ganze Zahlen $a_1, ..., a_n$ und ganze Zahl b

- Frage: Ex. eine Indexmenge $I \subseteq \{1, ..., n\}$ mit $\sum_{i \in I} a_i = b$?
- Pseudopolynomiell lösbar

3.2.11 PARTITION

Eingabe: Positive ganze Zahlen $a_1, ..., a_n$ mit $\sum_{i=1}^{n} a_i = 2A$

- Frage: Ex. eine Indexmenge $I \subseteq \{1, ..., n\}$ mit $\sum_{i \in I} a_i = A$?
- Pseudopolynomiell lösbar

3.2.12 PARTITION-INTO-THREE-SETS

Eingabe: Positive ganze Zahlen $a_1, ..., a_n$

- Frage: Ex. Indexmengen $I, J, K \subseteq \{1, ..., n\}$ mit $\sum_{i \in I} a_i = \sum_{j \in J} a_j = \sum_{i \in K} a_k$?
- Pseudopolynomiell lösbar

3.2.13 KP - Rucksackproblem

Eingabe: Natürliche Zahlen $w_1, ..., w_n, p_1, ..., p_n$ und b sowie Schranke γ

- Lösungen: Indexmenge $I \in \{1, ..., n\}$ mit $w(I) := \sum_{i \in I} w_i \leq b$
- Frage: Exisitiert eine Lösung mit $p(I) \geq \gamma$?
- Optimierung: Finde größtes γ
- Pseudopolynomiell lösbar

3.2.14 Bin Packing

Eingabe: Natürliche Zahlen b und $w_1, ..., w_n \in \{1, ..., b\}$ sowie Schranke γ

- Lösung: Zahl k und Funktion $f : \{1, ..., n\} \to \{1, ..., k\}$, sodass $\forall i \in \{1, ..., k\} : \sum_{j \in f^{-1}(i)} w_j \leq b$
- Frage: Existiert eine Lösung mit $k \leq \gamma$?
- Optimierung: Finde minimales γ

3.2.15 COLORING

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

- Frage: Gibt es eine Färbung/Funktion $c : V \to \{1, ..., k\}$, sodass $\forall \{u, v\} \in E : c(u) \neq c(v)$?

3.2.16 THREE-PARTITION

Eingabe: Natürliche Zahlen $a_1, ..., a_n, b_1, ..., b_n, c_1, ..., c_n$ mit $\sum_{i=1}^{n}(a_i + b_i + c_i) = n \cdot S$

- Frage: Gibt es Permutationen α, β von $[1, n]$, sodass $a_{\alpha(i)} + b_{\beta(i)} + c_i = S,\ \forall i \in [1, n]$
- Stark NP-schwer

3.2.17 Integer-Programming

Eingabe: Matrix $A \in \Sigma^{m \times n}$ und Vektor $b \in \Sigma^m$

- Frage: Gibt es einen Vektor $x \in \Sigma^n$ mit $Ax = b$?

3.2.18 Makespan-Scheduling

Eingabe: m Maschinen, n Jobs mit Laufzeiten $p_1, ..., p_n$ und Schranke γ

- Frage: Gibt es eine Zuteilung $c : [1, n] \to [1, m]$, sodass die maximale Laufzeit aller Maschinen kleiner als γ ist?
- Optimierung: Finde minimales γ

3.2.19 Dominating Set

Eingabe: Graph $G = (V, E)$ und Zahl k

- Frage: Gibt es eine Kontenmenge $D \subseteq V$ mit $|D| \leq k$, sodass $\forall v \in V \setminus D :\ \exists w \in D :\ (v, w) \in E$

3.3 Polynomielle Reduktion

Es seien L_1 und L_2 Sprachen über Σ_1 und Σ_2. Dann ist L_1 polynomiell reduzierbar auf L_2 (Notation: $L_1 \leq_p L_2$), wenn eine polynomiell berechenbare Funktion $f : \Sigma_1^* \to \Sigma_2^*$ existiert, sodass $\forall x \in \Sigma_1^*$ gilt: $x \in L_1 \ \Leftrightarrow\ f(x) \in L_2$

- Falls $L_1 \leq_p L_2$ und $L_2 \in P$, dann ist auch $L_1 \in P$
- Falls $L_1 \leq_p L_2$ und $L_1 \in NP$, dann ist auch $L_2 \in NP$

3.4 NP-Vollständigkeit

Ein Problem L heißt NP-schwer, wenn gilt $\forall L' \in NP :\ L' \leq_p L$, sich also alle Probleme aus NP auf L reduzieren lassen.

- L heißt NP-vollständig, wenn $L \in NP$ und L ist NP-schwer $\to$ Klasse heißt NPC
- $P \cap NPC = \emptyset$ (außer P=NP)

Satz von Cook & Levin:

- SAT ist NP-vollständig

Beweisstruktur für NP-Vollständigkeit:

- Zeige $L \in NP$
- Bestimme f für $L' \leq_p L$ für eine L' ist NP-vollständig
- Zeige f ist polynomiell berechenbar
- Zeige Korrektheit der Reduktion

3.5 Pseudopolynomielle Zeit und starke NP-Schwerheit

Laufzeitverhalten ist immer abhängig von der Eingabelänge. Verschiedene Codierungen machen nur einen polynomiellen Unterschied (außer unäre Codierung!!!)
Definiere $Number(I)$ als den Wert der größten in Instanz I vorkommenden Zahl

- Ein Algorithmus A löst das Problem X in pseudopolinomieller Zeit, falls die Laufzeit von A auf Instanzen I von X polynomiell in der Eingabelänge $|I|$ und in $Number(I)$ beschränkt ist
- Zeige Pseudopolynomialität:
  - Durch $Number(I)$ beschränkte Reduktion von pseudopolynomiellem Problem
  - Dynamische Programmierung des Problems

Ein Entscheidungsproblem X ist stark NP-schwer, wenn es ein Polynom $q : \mathbb{N} \to \mathbb{N}$ gibt, sodass die Restriktion von X auf Instanzen I mit $Number(I) \leq q(|I|)$ NP-schwer ist

- Stark NP-schwere Probleme sind nicht pseudopolynomiell lösbar (außer P=NP)

3.6 Optimierungsprobleme

Optimierungsprobleme besitzen eine Menge $\mathcal{L}$ von Lösungen mit $f : \mathcal{L} \to \mathbb{N}$ als Kosten-/Profitfunktion

- Die Kosten/der Profit sollen minimiert/maximiert werden
- P und NP enthalten nur Entscheidungsprobleme $\to$ Umformulieren in Entscheidungsproblem
- Oft gilt: Wenn das Entscheidungsproblem entscheidbar ist, dann auch das Optimierungsproblem
  - Verwende das entscheidbare Entscheidungsproblem als Unterprogramm zur Lösung

3.7 Weitere Komplexitätsklassen

EXPTIME ist eine weitere Komplexitätsklasse definiert über:

- EXPTIME ist die Klasse aller Entscheidungsprobleme. die durch eine DTM M entschieden werden und deren Worst-Case-Laufzeit durch $2^{q(n)}$ mit polynom q beschränkt ist.
- $NP \subseteq EXPTIME$

coNP liefert eine Art Komplement der Klasse NP

- Ein Entscheidungsproblem X liegt in coNP, wenn für jedes Wort $x \notin X$ ein polynomiell langes Zertifikat y existiert, das in polynomieller Zeit verifiziert werden kann
- X ist coNP-vollständig, alle coNP-vollständigen Probleme auf X reduziert werden können
- Wenn X NP-vollständig ist, dann ist $\overline{X}$ coNP-vollständig

NP-intermediate liegt zwischen P und NPC

- $L \in NP - intermediate$, wenn $L \in NP$ aber $L \notin P$ und $L \notin NPC$
- Graphisomorphismus bisher einzig bekanntes Problem, dass in NP-intermediate vermutet wird

PSPACE

- PSPACE ist die Klasse der Entscheidungsprobleme, deren Speicherplatz polynomiell in der Eingabelänge bechränkt sind
- Satz von Savitch: PSPACE=NPSPACE
- NP$\subseteq$NPSPACE=PSPACE

- PSPACE$\subseteq$EXPTIME
- P$\neq$EXPTIME
- P$\subseteq$NP$\subseteq$PSPACE$\subseteq$EXPTIME
  - Mindestens eine Teilmengenbeziehung muss strikt sein, aber unbekannt welche