Algorithmische Spieltheorie

Komplexitätstheorie

Paul Nüsken

14. April 2026; 16. April 2026

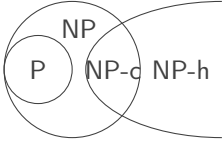
MNF, Institut für Informatik
Komplexitätstheorie und Kryptologie

Komplexitätstheorie

Komplexitätstheorie (informal)

Ein (großes) Teilgebiet der Komplexitätstheorie beschäftigt sich mit der Klassifizierung von Entscheidungsproblemen bezüglich Komplexitätsklassen.

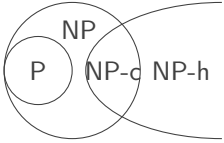
- Entscheidungsproblem: Ausgabe ist Ja oder Nein.



Komplexitätstheorie (informal)

Ein (großes) Teilgebiet der Komplexitätstheorie beschäftigt sich mit der Klassifizierung von Entscheidungsproblemen bezüglich Komplexitätsklassen.

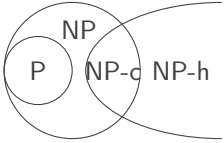
- Entscheidungsproblem: Ausgabe ist Ja oder Nein.
- Instanz: Eine Eingabe bezüglich eines gegebenen Problems



Komplexitätstheorie (informal)

Ein (großes) Teilgebiet der Komplexitätstheorie beschäftigt sich mit der Klassifizierung von Entscheidungsproblemen bezüglich Komplexitätsklassen.

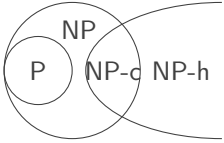
- Entscheidungsproblem: Ausgabe ist Ja oder Nein.
- Instanz: Eine Eingabe bezüglich eines gegebenen Problems
- Die Komplexitätsklasse P: Entscheidungsprobleme die (deterministisch) in Polynomialzeit entschieden werden können.



Komplexitätstheorie (informal)

Ein (großes) Teilgebiet der Komplexitätstheorie beschäftigt sich mit der Klassifizierung von Entscheidungsproblemen bezüglich Komplexitätsklassen.

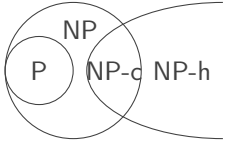
- Entscheidungsproblem: Ausgabe ist Ja oder Nein.
- Instanz: Eine Eingabe bezüglich eines gegebenen Problems
- Die Komplexitätsklasse P: Entscheidungsprobleme die (deterministisch) in Polynomialzeit entschieden werden können.
- Die Komplexitätsklasse NP: Entscheidungsprobleme deren Lösung (deterministisch) in Polynomialzeit überprüft werden können.



Komplexitätstheorie (informal)

Ein (großes) Teilgebiet der Komplexitätstheorie beschäftigt sich mit der Klassifizierung von Entscheidungsproblemen bezüglich Komplexitätsklassen.

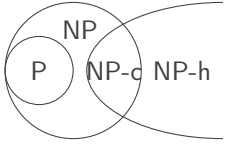
- Entscheidungsproblem: Ausgabe ist Ja oder Nein.
- Instanz: Eine Eingabe bezüglich eines gegebenen Problems
- Die Komplexitätsklasse P: Entscheidungsprobleme die (deterministisch) in Polynomialzeit entschieden werden können.
- Die Komplexitätsklasse NP: Entscheidungsprobleme deren Lösung (deterministisch) in Polynomialzeit überprüft werden können.
- NP-schweres Problem (NP-h): Mindestens so schwer zu lösen (bis auf polynomiellen Zeitfaktor) wie alle Probleme in NP.



Komplexitätstheorie (informal)

Ein (großes) Teilgebiet der Komplexitätstheorie beschäftigt sich mit der Klassifizierung von Entscheidungsproblemen bezüglich Komplexitätsklassen.

- Entscheidungsproblem: Ausgabe ist Ja oder Nein.
- Instanz: Eine Eingabe bezüglich eines gegebenen Problems
- Die Komplexitätsklasse P: Entscheidungsprobleme die (deterministisch) in Polynomialzeit entschieden werden können.
- Die Komplexitätsklasse NP: Entscheidungsprobleme deren Lösung (deterministisch) in Polynomialzeit überprüft werden können.
- NP-schweres Problem (NP-h): Mindestens so schwer zu lösen (bis auf polynomiellen Zeitfaktor) wie alle Probleme in NP.
- NP-vollständiges Problem (NP-c): Liegt in NP und ist NP-schwer.



Polynomialzeit-many-one-Reduktion

Wir wollen nun zeigen, dass ein Entscheidungsproblem A NP-schwer ist.

- Dazu zeigen wir, dass ein NP-schweres Problem B Polynomialzeit-many-one-reduzierbar auf A ist, geschrieben $B \leq_m^p A$.

Polynomialzeit-many-one-Reduktion

Wir wollen nun zeigen, dass ein Entscheidungsproblem A NP-schwer ist.

- Dazu zeigen wir, dass ein NP-schweres Problem B Polynomialzeit-many-one-reduzierbar auf A ist, geschrieben $B \leq_m^p A$.
- Wir geben eine effizient berechenbare Funktion f (Konstruktion) an, welche eine beliebige Instanz $I_B \in B$ in eine Instanz $I_A \in A$ überführt.

Polynomialzeit-many-one-Reduktion

Wir wollen nun zeigen, dass ein Entscheidungsproblem A NP-schwer ist.

- Dazu zeigen wir, dass ein NP-schweres Problem B Polynomialzeit-many-one-reduzierbar auf A ist, geschrieben $B \leq_m^p A$.
- Wir geben eine effizient berechenbare Funktion f (Konstruktion) an, welche eine beliebige Instanz $I_B \in B$ in eine Instanz $I_A \in A$ überführt.
- Für alle I_B und $I_A = f(I_B)$ muss nun gelten, dass I_A eine JA-Instanz ist, genau dann wenn I_B eine JA-Instanz ist (Korrektheitsbeweis).

Polynomialzeit-many-one-Reduktion

Wir wollen nun zeigen, dass ein Entscheidungsproblem A NP-schwer ist.

- Dazu zeigen wir, dass ein NP-schweres Problem B Polynomialzeit-many-one-reduzierbar auf A ist, geschrieben $B \leq_m^p A$.
- Wir geben eine effizient berechenbare Funktion f (Konstruktion) an, welche eine beliebige Instanz $I_B \in B$ in eine Instanz $I_A \in A$ überführt.
- Für alle I_B und $I_A = f(I_B)$ muss nun gelten, dass I_A eine JA-Instanz ist, genau dann wenn I_B eine JA-Instanz ist (Korrektheitsbeweis).
- $\leq_m^p$ ist transitiv, d.h. alle Probleme aus NP sind auf A reduzierbar.

Polynomialzeit-many-one-Reduktion

Was haben wir damit gezeigt?

Polynomialzeit-many-one-Reduktion

Was haben wir damit gezeigt?

- Falls A in P liegt, liegt B auch in P ($\mathrm{P} = \mathrm{NP}$).

Polynomialzeit-many-one-Reduktion

Was haben wir damit gezeigt?

- Falls A in P liegt, liegt B auch in P ($\mathrm{P} = \mathrm{NP}$).
- Falls B nicht in P liegt, liegt A auch nicht in P ($\mathrm{P} \neq \mathrm{NP}$).

Polynomialzeit-many-one-Reduktion

Was haben wir damit gezeigt?

- Falls A in P liegt, liegt B auch in P ($\mathrm{P} = \mathrm{NP}$).
- Falls B nicht in P liegt, liegt A auch nicht in P ($\mathrm{P} \neq \mathrm{NP}$).

$\Rightarrow$ Kein bekannter Algorithmus kann NP-schwere Probleme effizient lösen.

Beispiel 3-Sat

Ein Beispiel für ein NP-vollständiges Problem:

3-SAT	
Gegeben:	Eine Aussagenlogische Formel in konjunktiver Normalform $\phi = C_1 \wedge C_2 \ldots \wedge C_m$ mit jeweils drei literalen pro Klausel.
Frage:	Gibt es eine Variablenbelegung, welche ϕ erfüllt?

Beispiel 3-Sat

Ein Beispiel für ein NP-vollständiges Problem:

3-SAT	
Gegeben:	Eine Aussagenlogische Formel in konjunktiver Normalform $\phi = C_1 \wedge C_2 \ldots \wedge C_m$ mit jeweils drei literalen pro Klausel.
Frage:	Gibt es eine Variablenbelegung, welche ϕ erfüllt?

- Beispiel: $\phi = (x_1 \vee \neg x_3 \vee x_4) \wedge (\neg x_2 \vee x_3 \vee \neg x_4) \wedge (\neg x_1 \vee x_2 \vee x_4)$

Beispiel 3-Sat

Ein Beispiel für ein NP-vollständiges Problem:

3-SAT	
Gegeben:	Eine Aussagenlogische Formel in konjunktiver Normalform $\phi = C_1 \wedge C_2 \ldots \wedge C_m$ mit jeweils drei literalen pro Klausel.
Frage:	Gibt es eine Variablenbelegung, welche ϕ erfüllt?

- Beispiel: $\phi = (x_1 \vee \neg x_3 \vee x_4) \wedge (\neg x_2 \vee x_3 \vee \neg x_4) \wedge (\neg x_1 \vee x_2 \vee x_4)$
- Eine mögliche erfüllende Belegung ist $(x_1, x_2, x_3, x_4) = (1, 0, 0, 1)$.

Beispiel NP-Vollständigkeitsbeweis

Wir führen ein neues Problem ein.

Clique	
Gegeben:	Ein ungerichteter Graph $G = (V, E)$ und eine Zahl $k \in \mathbb{N}$.
Frage:	Gibt es eine Clique, also eine Teilmenge an paarweise durch eine Kante verbundene Knoten, der Größe mindestens k?

Aufgabe:

a) Geben Sie eine "Ja"-Instanz für Clique mit $k = 5$ an.

b) Geben Sie eine "Nein"-Instanz für Clique mit $k = 4$ und einem Graphen mit mindestens 6 Knoten an.

Beispiel NP-Vollständigkeitsbeweis

Nun wollen wir NP-Vollständigkeit zeigen.

CLIQUE	
Gegeben:	Ein ungerichteter Graph $G = (V, E)$ und eine Zahl $k \in \mathbb{N}$.
Frage:	Gibt es eine Clique, also eine Teilmenge an paarweise durch eine Kante verbundene Knoten, der Größe mindestens k?

- Zu zeigen: CLIQUE liegt in NP.

Beispiel NP-Vollständigkeitsbeweis

Nun wollen wir NP-Vollständigkeit zeigen.

CLIQUE	
Gegeben:	Ein ungerichteter Graph $G = (V, E)$ und eine Zahl $k \in \mathbb{N}$.
Frage:	Gibt es eine Clique, also eine Teilmenge an paarweise durch eine Kante verbundene Knoten, der Größe mindestens k?

- Zu zeigen: CLIQUE liegt in NP.
  - Wir können in quadratischer Zeit verifizieren, ob eine gegebene Teilmenge an Knoten eine Clique bildet oder nicht.

Beispiel NP-Vollständigkeitsbeweis

Nun wollen wir NP-Vollständigkeit zeigen.

- Zu zeigen: CLIQUE ist NP-schwer.

$$\text{3-Sat} \leq_m^p \text{Clique}$$

Beispiel NP-Vollständigkeitsbeweis

Nun wollen wir NP-Vollständigkeit zeigen.

- Zu zeigen: CLIQUE ist NP-schwer.

$$\text{3-Sat} \leq_m^p \text{Clique}$$

- **Konstruktion:**
  - Konstruiere einen Graphen G.
  - Füge für jedes Literal in jeder Klausel einen Knoten ein.

Beispiel NP-Vollständigkeitsbeweis

Nun wollen wir NP-Vollständigkeit zeigen.

- Zu zeigen: CLIQUE ist NP-schwer.

$$\text{3-Sat} \leq_m^p \text{Clique}$$

- **Konstruktion:**
  - Konstruiere einen Graphen G.
  - Füge für jedes Literal in jeder Klausel einen Knoten ein.
  - Verbinde alle Knoten durch Kanten untereinander.

Beispiel NP-Vollständigkeitsbeweis

Nun wollen wir NP-Vollständigkeit zeigen.

- Zu zeigen: CLIQUE ist NP-schwer.

$$\text{3-Sat} \leq_m^p \text{Clique}$$

- **Konstruktion:**
  - Konstruiere einen Graphen G.
  - Füge für jedes Literal in jeder Klausel einen Knoten ein.
  - Verbinde alle Knoten durch Kanten untereinander.
  - Entferne die Kanten zwischen Knoten innerhalb einer Klausel

Beispiel NP-Vollständigkeitsbeweis

Nun wollen wir NP-Vollständigkeit zeigen.

- Zu zeigen: CLIQUE ist NP-schwer.

$$\text{3-Sat} \leq_m^p \text{Clique}$$

- **Konstruktion:**
  - Konstruiere einen Graphen G.
  - Füge für jedes Literal in jeder Klausel einen Knoten ein.
  - Verbinde alle Knoten durch Kanten untereinander.
  - Entferne die Kanten zwischen Knoten innerhalb einer Klausel
  - Entferne die Kanten zwischen Knoten, welche ein Literal und seine Negation repräsentieren.

Beispiel NP-Vollständigkeitsbeweis

Nun wollen wir NP-Vollständigkeit zeigen.

- Zu zeigen: CLIQUE ist NP-schwer.

$$\text{3-Sat} \leq_m^p \text{Clique}$$

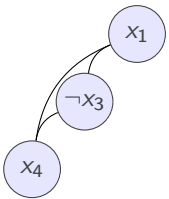
- **Konstruktion:**
  - Konstruiere einen Graphen G.
  - Füge für jedes Literal in jeder Klausel einen Knoten ein.
  - Verbinde alle Knoten durch Kanten untereinander.
  - Entferne die Kanten zwischen Knoten innerhalb einer Klausel
  - Entferne die Kanten zwischen Knoten, welche ein Literal und seine Negation repräsentieren.
  - Setze $k = m$, also auf die Zahl der Klauseln der gegebenen Formel ϕ.

Beispiel Konstruktion

$$\phi = (x_1 \vee \neg x_3 \vee x_4) \wedge (\neg x_2 \vee x_3 \vee \neg x_4) \wedge (\neg x_1 \vee x_2 \vee x_4)$$

Beispiel Konstruktion

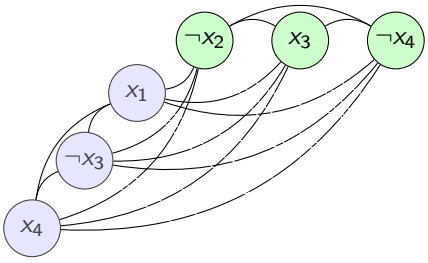
$$\phi = (x_1 \vee \neg x_3 \vee x_4) \wedge (\neg x_2 \vee x_3 \vee \neg x_4) \wedge (\neg x_1 \vee x_2 \vee x_4)$$



- Konstruiere einen Graphen G.
- Füge für jedes Literal in jeder Klausel einen Knoten ein.
- Verbinde alle Knoten durch Kanten untereinander.

Beispiel Konstruktion

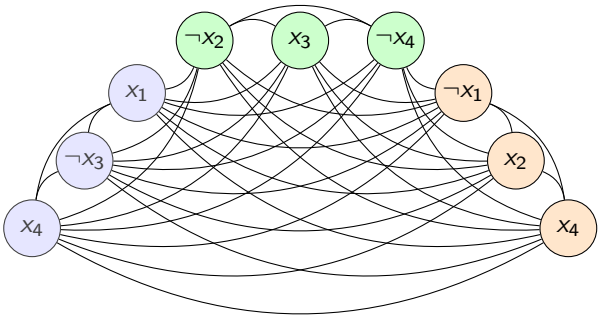
$$\phi = (x_1 \vee \neg x_3 \vee x_4) \wedge (\neg x_2 \vee x_3 \vee \neg x_4) \wedge (\neg x_1 \vee x_2 \vee x_4)$$



- Konstruiere einen Graphen G.
- Füge für jedes Literal in jeder Klausel einen Knoten ein.
- Verbinde alle Knoten durch Kanten untereinander.

Beispiel Konstruktion

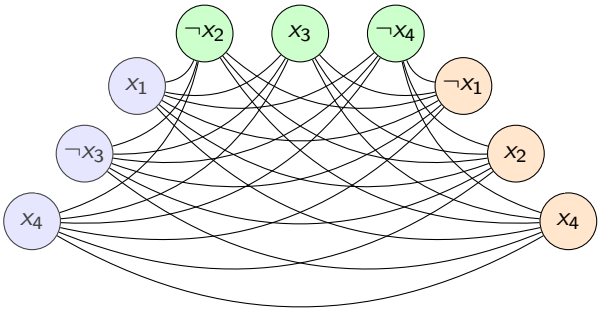
$$\phi = (x_1 \vee \neg x_3 \vee x_4) \wedge (\neg x_2 \vee x_3 \vee \neg x_4) \wedge (\neg x_1 \vee x_2 \vee x_4)$$



- Konstruiere einen Graphen G.
- Füge für jedes Literal in jeder Klausel einen Knoten ein.
- Verbinde alle Knoten durch Kanten untereinander.

Beispiel Konstruktion

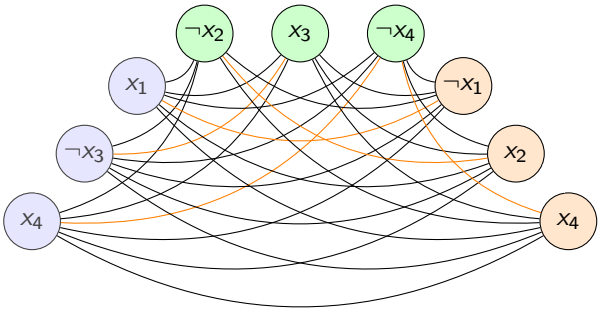
$$\phi = (x_1 \vee \neg x_3 \vee x_4) \wedge (\neg x_2 \vee x_3 \vee \neg x_4) \wedge (\neg x_1 \vee x_2 \vee x_4)$$



- Konstruiere einen Graphen G.
- Füge für jedes Literal in jeder Klausel einen Knoten ein.
- Verbinde alle Knoten durch Kanten untereinander.
- Entferne die Kanten zwischen Knoten innerhalb einer Klausel

Beispiel Konstruktion

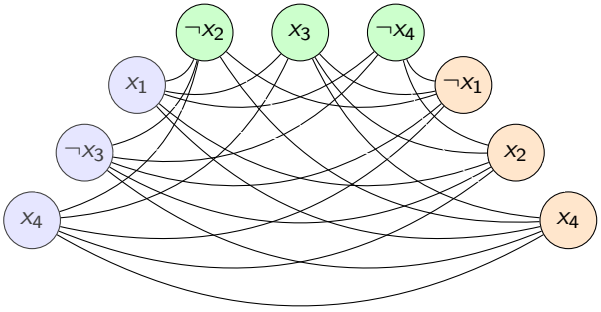
$$\phi = (x_1 \vee \neg x_3 \vee x_4) \wedge (\neg x_2 \vee x_3 \vee \neg x_4) \wedge (\neg x_1 \vee x_2 \vee x_4)$$



- Konstruiere einen Graphen G.
- Füge für jedes Literal in jeder Klausel einen Knoten ein.
- Verbinde alle Knoten durch Kanten untereinander.
- Entferne die Kanten zwischen Knoten innerhalb einer Klausel
- Entferne die Kanten zwischen Knoten, welche ein Literal und seine Negation repräsentieren.

Beispiel Konstruktion

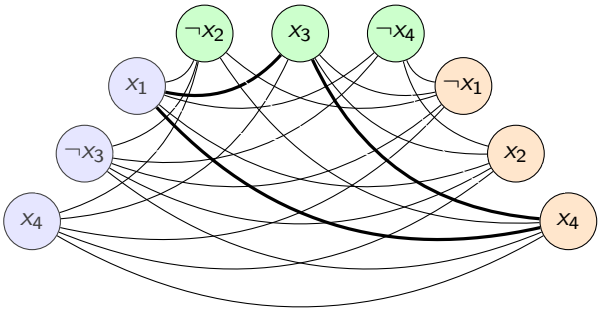
$$\phi = (x_1 \vee \neg x_3 \vee x_4) \wedge (\neg x_2 \vee x_3 \vee \neg x_4) \wedge (\neg x_1 \vee x_2 \vee x_4)$$



- Konstruiere einen Graphen G.
- Füge für jedes Literal in jeder Klausel einen Knoten ein.
- Verbinde alle Knoten durch Kanten untereinander.
- Entferne die Kanten zwischen Knoten innerhalb einer Klausel
- Entferne die Kanten zwischen Knoten, welche ein Literal und seine Negation repräsentieren.
- Setze $k = m = 3$.

Beispiel Konstruktion

$$\phi = (x_1 \vee \neg x_3 \vee x_4) \wedge (\neg x_2 \vee x_3 \vee \neg x_4) \wedge (\neg x_1 \vee x_2 \vee x_4)$$



- Konstruiere einen Graphen G.
- Füge für jedes Literal in jeder Klausel einen Knoten ein.
- Verbinde alle Knoten durch Kanten untereinander.
- Entferne die Kanten zwischen Knoten innerhalb einer Klausel
- Entferne die Kanten zwischen Knoten, welche ein Literal und seine Negation repräsentieren.
- Setze $k = m = 3$.

Beispiel NP-Vollständigkeitsbeweis

Korrektheitsbeweis: Zu zeigen: ϕ ist erfüllbar, genau dann wenn G enthält eine Clique der Größe (mindestens) k.

$\Rightarrow$ Aufgabe!

$\Leftarrow$ Aufgabe!

TWO-Clique

Wir führen ein weiteres Problem ein.

TWO-Clique	
Gegeben:	Ein ungerichteter Graph $G = (V, E)$.
Frage:	Besitzt G zwei Cliquen $V_1, V_2 \subset V$, sodass V_1 und V_2 eine Partition von G bilden ($V = V1 \cup V2$)?

Aufgabe:

a) Zeigen Sie entweder, dass TWO-Clique in P liegt, oder dass TWO-Clique NP-vollständig ist.

b) Veranschaulichen Sie ihren Beweis an einem selbst gewählten Beispiel.

Co-Klassen

- Sei $\mathscr{C}$ eine beliebige Komplexitätsklasse für ein Entscheidungsproblem.

Co-Klassen

- Sei $\mathscr{C}$ eine beliebige Komplexitätsklasse für ein Entscheidungsproblem.
- Dann enthält co$\mathscr{C}$ genau die Probleme, dessen Komplement in $\mathscr{C}$ liegt.

Co-Klassen

- Sei $\mathscr{C}$ eine beliebige Komplexitätsklasse für ein Entscheidungsproblem.
- Dann enthält $\mathrm{co}\mathscr{C}$ genau die Probleme, dessen Komplement in $\mathscr{C}$ liegt.
- Das bedeutet, genau die Nein-Instanzen für ein Problem $A \in \mathscr{C}$ sind Ja-Instanzen für das Komplement von A.

Co-Klassen

- Sei $\mathscr{C}$ eine beliebige Komplexitätsklasse für ein Entscheidungsproblem.
- Dann enthält $\mathrm{co}\mathscr{C}$ genau die Probleme, dessen Komplement in $\mathscr{C}$ liegt.
- Das bedeutet, genau die Nein-Instanzen für ein Problem $A \in \mathscr{C}$ sind Ja-Instanzen für das Komplement von A.
- Es gilt $\mathrm{P} = \mathrm{coP}$.

Co-Klassen

- Sei $\mathscr{C}$ eine beliebige Komplexitätsklasse für ein Entscheidungsproblem.
- Dann enthält $\mathrm{co}\mathscr{C}$ genau die Probleme, dessen Komplement in $\mathscr{C}$ liegt.
- Das bedeutet, genau die Nein-Instanzen für ein Problem $A \in \mathscr{C}$ sind Ja-Instanzen für das Komplement von A.
- Es gilt $\mathrm{P} = \mathrm{coP}$.
- Es ist ein offenes Problem, ob $\mathrm{NP} = \mathrm{coNP}$ gilt.

Beispiel coNP

UNSAT	
Gegeben:	Eine Aussagenlogische Formel ϕ.
Frage:	Ist ϕ unerfüllbar?

Anders formuliert

Für alle Variablenbelegungen gilt: ϕ is nicht erfüllt.

Beispiel coNP

UNSAT	
Gegeben:	Eine Aussagenlogische Formel ϕ.
Frage:	Ist ϕ unerfüllbar?

Anders formuliert
Für alle Variablenbelegungen gilt: ϕ is nicht erfüllt.

UNSAT ist coNP-vollständig.

Orakel-Turingmaschinen

Sei $\mathscr{C}$ eine beliebige Komplexitätsklasse für ein Entscheidungsproblem.
Ein $\mathscr{C}$-Orakel kann ein beliebiges Problem in $\mathscr{C}$ in einem Schritt entscheiden.

Orakel-Turingmaschinen

Sei $\mathscr{C}$ eine beliebige Komplexitätsklasse für ein Entscheidungsproblem.
Ein $\mathscr{C}$-Orakel kann ein beliebiges Problem in $\mathscr{C}$ in einem Schritt entscheiden.

- Die Klasse $\Delta_2^p = \mathrm{P}^{\mathrm{NP}}$ enthält genau die Probleme, welche mit einer **deterministischen** Turingmaschine in Polynomialzeit entschieden werden kann, welche **beliebig oft** auf ein NP-**Orakel** zugreift.

Orakel-Turingmaschinen

Sei $\mathscr{C}$ eine beliebige Komplexitätsklasse für ein Entscheidungsproblem.
Ein $\mathscr{C}$-Orakel kann ein beliebiges Problem in $\mathscr{C}$ in einem Schritt entscheiden.

- Die Klasse $\Delta_2^p = \mathrm{P}^{\mathrm{NP}}$ enthält genau die Probleme, welche mit einer **deterministischen** Turingmaschine in Polynomialzeit entschieden werden kann, welche **beliebig oft** auf ein NP-**Orakel** zugreift.
- Die Klasse $\Theta_2^p = \mathrm{P}^{\mathrm{NP[log]}}$ enthält genau die Probleme, welche mit einer **deterministischen** Turingmaschine in Polynomialzeit entschieden werden kann, welche **höchstens logarithmisch oft** auf ein NP-**Orakel** zugreift.

Orakel-Turingmaschinen

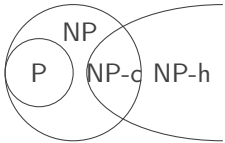
Sei $\mathscr{C}$ eine beliebige Komplexitätsklasse für ein Entscheidungsproblem.
Ein $\mathscr{C}$-Orakel kann ein beliebiges Problem in $\mathscr{C}$ in einem Schritt entscheiden.

- Die Klasse $\Delta_2^p = \mathrm{P}^{\mathrm{NP}}$ enthält genau die Probleme, welche mit einer **deterministischen** Turingmaschine in Polynomialzeit entschieden werden kann, welche **beliebig oft** auf ein NP-**Orakel** zugreift.
- Die Klasse $\Theta_2^p = \mathrm{P}^{\mathrm{NP[log]}}$ enthält genau die Probleme, welche mit einer **deterministischen** Turingmaschine in Polynomialzeit entschieden werden kann, welche **höchstens logarithmisch oft** auf ein NP-**Orakel** zugreift.
- Die Klasse $\Sigma_2^p = \mathrm{NP}^{\mathrm{NP}}$ enthält genau die Probleme, welche mit einer **nichtdeterministischen** Turingmaschine in Polynomialzeit entschieden werden kann, welche **beliebig oft** auf ein NP-**Orakel** zugreift.

Entscheidungsprobleme und Klasse NP - Wiederholung

Ein (großes) Teilgebiet der Komplexitätstheorie beschäftigt sich mit der Klassifizierung von Entscheidungsproblemen bezüglich Komplexitätsklassen.

- Entscheidungsproblem: Ausgabe ist Ja oder Nein.
- Instanz: Eine Eingabe bezüglich eines gegebenen Problems.
- Die Komplexitätsklasse NP: Entscheidungsprobleme deren Lösung (deterministisch) in Polynomialzeit überprüft werden können.



Weitere Entscheidungsprobleme

Wir führen zwei neue Entscheidungsprobleme ein.

VERTEX-COVER	
Gegeben:	Ein ungerichteter Graph $G = (V, E)$ und eine positive ganze Zahl k.
Frage:	Gibt es eine Knotenüberdeckung V' von G mit $\lvert V' \rvert \leq k$, d.h. $V' \subseteq V$ und für alle $u, v \in E$ gilt $u \in V'$ oder $v \in V'$?

Weitere Entscheidungsprobleme

Wir führen zwei neue Entscheidungsprobleme ein.

Vertex-Cover	
Gegeben:	Ein ungerichteter Graph $G = (V, E)$ und eine positive ganze Zahl k.
Frage:	Gibt es eine Knotenüberdeckung V' von G mit $\lvert V' \rvert \leq k$, d.h. $V' \subseteq V$ und für alle $u, v \in E$ gilt $u \in V'$ oder $v \in V'$?

Subset-Sum	
Gegeben:	Positive ganze Zahlen $(a_1, \ldots, a_n)$ und eine positive ganze Zahl K.
Frage:	Existiert eine Teilmenge $A \subseteq \{1, \ldots, n\}$, für die $\sum_{i \in A} a_i = K$ gilt?

Komplexität von Vertex-Cover und Subset-Sum

Theorem

$$\textit{Clique} \leq_m^p \textit{Vertex-Cover} \tag{1}$$

$$\textit{Vertex-Cover} \leq_m^p \textit{Subset-Sum} \tag{2}$$

Was können wir damit über die Komplexität von Vertex-Cover und Subset-Sum aussagen?

Zählprobleme

Neben Entscheidungsproblemen betrachten wir Zählprobleme, die in der Komplexitätstheorie ebenfalls bezüglich Komplexitätsklassen klassifiziert werden.

Zählprobleme

Neben Entscheidungsproblemen betrachten wir Zählprobleme, die in der Komplexitätstheorie ebenfalls bezüglich Komplexitätsklassen klassifiziert werden.

- Zählproblem: Ausgabe ist eine nicht negative ganze Zahl.

Zählprobleme

Neben Entscheidungsproblemen betrachten wir Zählprobleme, die in der Komplexitätstheorie ebenfalls bezüglich Komplexitätsklassen klassifiziert werden.

- Zählproblem: Ausgabe ist eine nicht negative ganze Zahl.
- Instanz: Eine Eingabe bezüglich eines gegebenen Problems.

Zählprobleme

Neben Entscheidungsproblemen betrachten wir Zählprobleme, die in der Komplexitätstheorie ebenfalls bezüglich Komplexitätsklassen klassifiziert werden.

- Zählproblem: Ausgabe ist eine nicht negative ganze Zahl.
- Instanz: Eine Eingabe bezüglich eines gegebenen Problems.
- Die Komplexitätsklasse #P: Die Klasse der Funktionen, welche die Anzahl der Lösungen von NP Problemen berechnen.

Zählprobleme

Neben Entscheidungsproblemen betrachten wir Zählprobleme, die in der Komplexitätstheorie ebenfalls bezüglich Komplexitätsklassen klassifiziert werden.

- Zählproblem: Ausgabe ist eine nicht negative ganze Zahl.
- Instanz: Eine Eingabe bezüglich eines gegebenen Problems.
- Die Komplexitätsklasse #P: Die Klasse der Funktionen, welche die Anzahl der Lösungen von NP Problemen berechnen.
- Ein Problem ist #P-vollständig, wenn es in #P liegt und #P-schwer ist (d.h. mindestens so schwer zu lösen wie alle Probleme in #P).

Beispiel: Subset-Sum und #Subset-Sum

Subset-Sum	
Gegeben:	Positive ganze Zahlen $(a_1, \ldots, a_n)$ und eine positive ganze Zahl K.
Frage:	**Existiert eine Teilmenge** $A \subseteq \{1, \ldots, n\}$, für die $\sum_{i \in A} a_i = K$ gilt?

Beispiel: Subset-Sum und #Subset-Sum

Subset-Sum	
Gegeben:	Positive ganze Zahlen $(a_1, \ldots, a_n)$ und eine positive ganze Zahl K.
Frage:	**Existiert eine Teilmenge** $A \subseteq \{1, \ldots, n\}$, für die $\sum_{i \in A} a_i = K$ gilt?

Das entsprechende #P-vollständige Zählproblem, kann wie folgt definiert werden.

#Subset-Sum	
Eingabe:	Positive ganze Zahlen $(a_1, \ldots, a_n)$ und eine positive ganze Zahl K.
Ausagbe:	**Für wie viele Teilmengen** gilt $A \subseteq \{1, \ldots, n\}$ $\sum_{i \in A} a_i = K$?

Beispiel: Subset-Sum und #Subset-Sum

Betrachte die folgenden Instanzen für SUBSET-SUM:

1. $((5,6,5,4,2,2),10)$
2. $((5,6,5,4,2,2),3)$

Beispiel: Subset-Sum und #Subset-Sum

Betrachte die folgenden Instanzen für SUBSET-SUM:

1. $((5,6,5,4,2,2),10)$
2. $((5,6,5,4,2,2),3)$

- Handelt es sich um JA- oder NEIN-Instanzen für SUBSET-SUM?

Beispiel: Subset-Sum und #Subset-Sum

Betrachte die folgenden Instanzen für SUBSET-SUM:

1. $((5,6,5,4,2,2),10)$, JA-Instanz
2. $((5,6,5,4,2,2),3)$

- Handelt es sich um JA- oder NEIN-Instanzen für SUBSET-SUM?

Beispiel: Subset-Sum und #Subset-Sum

Betrachte die folgenden Instanzen für SUBSET-SUM:

1. $((5,6,5,4,2,2),10)$, JA-Instanz
2. $((5,6,5,4,2,2),3)$, NEIN-Instanz

- Handelt es sich um JA- oder NEIN-Instanzen für SUBSET-SUM?

Beispiel: Subset-Sum und #Subset-Sum

Betrachte die folgenden Instanzen für SUBSET-SUM:

1. $((5,6,5,4,2,2),10)$, JA-Instanz
2. $((5,6,5,4,2,2),3)$, NEIN-Instanz

- Handelt es sich um JA- oder NEIN-Instanzen für SUBSET-SUM?
- Welches Ergebnis berechnet die Funktion #SUBSET-SUM für die Instanzen?

Beispiel: Subset-Sum und #Subset-Sum

Betrachte die folgenden Instanzen für SUBSET-SUM:

1. $((5,6,5,4,2,2),10)$, JA-Instanz, 3.
2. $((5,6,5,4,2,2),3)$, NEIN-Instanz

- Handelt es sich um JA- oder NEIN-Instanzen für SUBSET-SUM?
- Welches Ergebnis berechnet die Funktion #SUBSET-SUM für die Instanzen?

Beispiel: Subset-Sum und #Subset-Sum

Betrachte die folgenden Instanzen für SUBSET-SUM:

1. $((5,6,5,4,2,2),10)$, JA-Instanz, 3.
2. $((5,6,5,4,2,2),3)$, NEIN-Instanz, 0.

- Handelt es sich um JA- oder NEIN-Instanzen für SUBSET-SUM?
- Welches Ergebnis berechnet die Funktion #SUBSET-SUM für die Instanzen?

#SAT

$\#\mathrm{SAT}$	
Eingabe:	Eine Aussagenlogische Formel ϕ.
Ausagbe:	Für wie viele Belegungen der Variablen ist ϕ erfüllt?

$$\phi = x_1 \vee x_2 \tag{3}$$

$$\phi = (x_1 \vee x_2) \wedge x_3 \tag{4}$$

- Handelt es sich um JA- oder NEIN-Instanzen für SAT?
- Welches Ergebnis berechnet die Funktion $\#\mathrm{SAT}$ für die Instanzen?

Reduktion von Funktionen

Seien $f, g : \Sigma^* \to \mathbb{N}$.

Reduktion von Funktionen

Seien $f, g : \Sigma^* \to \mathbb{N}$.

- f ist funktionell reduzierbar auf g, wenn in Polynomialzeit berechenbare Funktionen $\psi : \mathbb{N} \to \mathbb{N}$ und $\rho : \Sigma^* \to \Sigma^*$ existieren, sodass

  $$f(x) = \psi(g(\rho(x)))$$

  für jedes $x \in \Sigma^*$ gilt.

Reduktion von Funktionen

Seien $f, g : \Sigma^* \to \mathbb{N}$.

- f ist funktionell reduzierbar auf g, wenn in Polynomialzeit berechenbare Funktionen $\psi : \mathbb{N} \to \mathbb{N}$ und $\rho : \Sigma^* \to \Sigma^*$ existieren, sodass

  $$f(x) = \psi(g(\rho(x)))$$

  für jedes $x \in \Sigma^*$ gilt.
- f ist sparsam (parsimoniously) reduzierbar auf g, wenn eine in Polynomialzeit berechenbare Funktion $\rho : \Sigma^* \to \Sigma^*$ existiert, sodass

  $$f(x) = g(\rho(x))$$

  für jedes $x \in \Sigma^*$ gilt.

Beispiel

#Partition	
Eingabe:	Eine Sequenz von positiven ganzen Zahlen $(k_1, \ldots, k_n)$ mit $\sum_{i=1}^{n} k_i$ ist gerade.
Ausagbe:	Für wie viele $A \subseteq \{1, \ldots, n\}$ gilt $\sum_{i \in A} k_i = \sum_{\{1,\ldots,n\} \setminus A} k_i$?

#Partition reduces to #Subset-Sum

PP

Eine weitere Klasse der Entscheidungsprobleme ist die Klasse PP.

PP

Eine weitere Klasse der Entscheidungsprobleme ist die Klasse PP.

- Wir betrachten ein Problem A.

PP

Eine weitere Klasse der Entscheidungsprobleme ist die Klasse PP.

- Wir betrachten ein Problem A.
- Wir sagen, dass A in der Komplexitätsklasse PP liegt, wenn

PP

Eine weitere Klasse der Entscheidungsprobleme ist die Klasse PP.

- Wir betrachten ein Problem A.
- Wir sagen, dass A in der Komplexitätsklasse PP liegt, wenn
  eine Funktion f aus der Klasse #P existiert,

PP

Eine weitere Klasse der Entscheidungsprobleme ist die Klasse PP.

- Wir betrachten ein Problem A.
- Wir sagen, dass A in der Komplexitätsklasse PP liegt, wenn

  eine Funktion f aus der Klasse $\#\mathrm{P}$ existiert,
  sodass für alle $x \in \Sigma^*$ gilt:

  $$x \in A \iff f(x) \geq 2^{p(|x|)-1},$$

  wobei p eine polynomiale Funktion ist.

PP

Eine weitere Klasse der Entscheidungsprobleme ist die Klasse PP.

- Wir betrachten ein Problem A.
- Wir sagen, dass A in der Komplexitätsklasse PP liegt, wenn

  eine Funktion f aus der Klasse $\#\mathrm{P}$ existiert,
  sodass für alle $x \in \Sigma^*$ gilt:

  $$x \in A \iff f(x) \geq 2^{p(|x|)-1},$$

  wobei p eine polynomiale Funktion ist.

Alternativ: Für eine Sprache L gilt, $L \in PP$, wenn eine probabilistische Polynomialzeit-Turingmaschine TM existiert, sodass

$$x \in L \iff Pr[TM(x) = 1] \geq 1/2.$$

Beispiel PP-Vollständigkeit

Das folgende Problem ist PP-vollständig.

MAJORITY-SAT	
Gegeben:	Eine Aussagenlogische Formel in konj. Normalform $\phi = C_1 \wedge C_2 \wedge \ldots \wedge C_m$.
Frage:	Ist es wahr, dass mehr als die Hälfte aller Variablenbelegungen ϕ erfüllen?