Theoretische Informatik II

Einheit 8.5

Grenzen überwinden

1. Pseudopolynomielle Algorithmen
2. Approximationsalgorithmen
3. Probabilistische Algorithmen

Wie kann man "unlösbare" Probleme angehen?

- **Künstliche Intelligenz**
  – Heuristische Lösung unentscheidbarer Probleme (ohne Erfolgsgarantie)
  – Theorembeweisen, Programmverifikation und -synthese (unvollständig)

Wie kann man "unlösbare" Probleme angehen?

- **Künstliche Intelligenz**
  - Heuristische Lösung unentscheidbarer Probleme (ohne Erfolgsgarantie)
  - Theorembeweisen, Programmverifikation und -synthese (unvollständig)
- **Approximierende und probabilistische Algorithmen**
  - Effiziente Bestimmung von nahezu optimalen Lösungen
  - Z.B. Primzahltest mit geringem Fehler (logarithmisch statt linear)

Wie kann man "unlösbare" Probleme angehen?

- **Künstliche Intelligenz**
  - Heuristische Lösung unentscheidbarer Probleme (ohne Erfolgsgarantie)
  - Theorembeweisen, Programmverifikation und -synthese (unvollständig)
- **Approximierende und probabilistische Algorithmen**
  - Effiziente Bestimmung von nahezu optimalen Lösungen
  - Z.B. Primzahltest mit geringem Fehler (logarithmisch statt linear)
- **Selbstorganisation statt vorformulierter Lösungen**
  - Lernverfahren, Neuronale Netze, genetische Algorithmen, ...

Wie kann man "unlösbare" Probleme angehen?

- **Künstliche Intelligenz**
  - Heuristische Lösung unentscheidbarer Probleme (ohne Erfolgsgarantie)
  - Theorembeweisen, Programmverifikation und -synthese (unvollständig)
- **Approximierende und probabilistische Algorithmen**
  - Effiziente Bestimmung von nahezu optimalen Lösungen
  - Z.B. Primzahltest mit geringem Fehler (logarithmisch statt linear)
- **Selbstorganisation statt vorformulierter Lösungen**
  - Lernverfahren, Neuronale Netze, genetische Algorithmen, ...

Suche nach neuen Wegen liefert tieferes Verständnis der Materie

Gibt es leichte $\mathcal{NP}$-vollständige Probleme?

Gibt es leichte $\mathcal{NP}$-vollständige Probleme?

- **Was unterscheidet $\boldsymbol{CLIQUE}$ von $\boldsymbol{KP}$?**
  – Beide Probleme sind $\mathcal{NP}$-vollständig

Gibt es leichte $\mathcal{NP}$-vollständige Probleme?

- **Was unterscheidet $\boldsymbol{CLIQUE}$ von $\boldsymbol{KP}$?**
  - Beide Probleme sind $\mathcal{NP}$-vollständig, aber
    - $3SAT \leq_p KP$ benutzt exponentiell große Zahlen als Codierung
    - $3SAT \leq_p CLIQUE$ codiert Formel durch gleichgroßen Graph

Gibt es leichte $\mathcal{NP}$-vollständige Probleme?

- **Was unterscheidet $\boldsymbol{CLIQUE}$ von $\boldsymbol{KP}$?**
  - Beide Probleme sind $\mathcal{NP}$-vollständig, aber
    - $3SAT \leq_p KP$ benutzt exponentiell große Zahlen als Codierung
    - $3SAT \leq_p CLIQUE$ codiert Formel durch gleichgroßen Graph
  - Ist KP nur wegen der großen Zahlen $\mathcal{NP}$-vollständig?

Gibt es leichte $\mathcal{NP}$-vollständige Probleme?

- **Was unterscheidet $CLIQUE$ von KP?**
  - Beide Probleme sind $\mathcal{NP}$-vollständig, aber
    - $3SAT \leq_p KP$ benutzt exponentiell große Zahlen als Codierung
    - $3SAT \leq_p CLIQUE$ codiert Formel durch gleichgroßen Graph
  - Ist KP nur wegen der großen Zahlen $\mathcal{NP}$-vollständig?

- **Es gibt "bessere" Lösungen für KP**

  $KP = \{ (g_1..g_n,\ a_1..a_n,\ G,\ A) \mid \exists J \subseteq \{1..n\}.\ \Sigma_{i \in J} g_i \leq G \wedge \Sigma_{i \in J} a_i \geq A \}$

Pseudopolynomielle Algorithmen

Gibt es leichte $\mathcal{NP}$-vollständige Probleme?

- **Was unterscheidet $CLIQUE$ von KP?**
  - Beide Probleme sind $\mathcal{NP}$-vollständig, aber
    - · $3SAT\leq_p KP$ benutzt exponentiell große Zahlen als Codierung
    - · $3SAT\leq_p CLIQUE$ codiert Formel durch gleichgroßen Graph
  - Ist KP nur wegen der großen Zahlen $\mathcal{NP}$-vollständig?

- **Es gibt "bessere" Lösungen für KP**

  $KP = \{ (g_1..g_n,\ a_1..a_n,\ G,\ A) \mid \exists J\subseteq\{1..n\}.\ \Sigma_{i\in J} g_i \leq G \wedge \Sigma_{i\in J} a_i \geq A \}$

  - Man muß nicht alle Kombinationen von $\{1..n\}$ einzeln auswerten

Gibt es leichte $\mathcal{NP}$-vollständige Probleme?

- **Was unterscheidet $CLIQUE$ von KP?**
  – Beide Probleme sind $\mathcal{NP}$-vollständig, aber
    · $3SAT \leq_p KP$ benutzt exponentiell große Zahlen als Codierung
    · $3SAT \leq_p CLIQUE$ codiert Formel durch gleichgroßen Graph
  – Ist KP nur wegen der großen Zahlen $\mathcal{NP}$-vollständig?

- **Es gibt "bessere" Lösungen für KP**

  $KP = \{ (g_1..g_n,\ a_1..a_n,\ G,\ A) \mid \exists J \subseteq \{1..n\}.\ \Sigma_{i \in J} g_i \leq G \wedge \Sigma_{i \in J} a_i \geq A \}$
  – Man muß nicht alle Kombinationen von $\{1..n\}$ einzeln auswerten
  – Man kann iterativ den optimalen Nutzen bestimmen,
  indem man die Anzahl der Gegenstände und das Gewicht erhöht

Pseudopolynomielle Algorithmen

Gibt es leichte $\mathcal{NP}$-vollständige Probleme?

- **Was unterscheidet $\boldsymbol{CLIQUE}$ von $\boldsymbol{KP}$?**
  - Beide Probleme sind $\mathcal{NP}$-vollständig, aber
    - $3SAT \leq_p KP$ benutzt exponentiell große Zahlen als Codierung
    - $3SAT \leq_p CLIQUE$ codiert Formel durch gleichgroßen Graph
  - Ist KP nur wegen der großen Zahlen $\mathcal{NP}$-vollständig?
- **Es gibt "bessere" Lösungen für $\boldsymbol{KP}$**

  $KP = \{\ (g_1..g_n,\ a_1..a_n,\ G,\ A) \mid \exists J \subseteq \{1..n\}.\ \Sigma_{i\in J} g_i \leq G \wedge \Sigma_{i\in J} a_i \geq A\ \}$
  - Man muß nicht alle Kombinationen von $\{1..n\}$ einzeln auswerten
  - Man kann iterativ den optimalen Nutzen bestimmen, indem man die Anzahl der Gegenstände und das Gewicht erhöht
  - Sehr effizient, wenn das maximale Gewicht nicht zu groß wird

Iterative Lösung für KP

$$\boldsymbol{KP} = \{\ (g_1..g_n,\ a_1..a_n,\ G,\ A) \mid \exists J{\subseteq}\{1..n\}.\ \Sigma_{i\in J} g_i \le G \wedge \Sigma_{i\in J} a_i \ge A\ \}$$

- **Betrachte Subprobleme** $KP(k,g)$
  - Verwende Gegenstände $1,..,k$ und Maximalgewicht $g{\le}G$
  - Bestimme optimalen Nutzen $N(k,g)$

ITERATIVE LÖSUNG FÜR KP

$$\boldsymbol{KP} = \{\ (g_1..g_n,\ a_1..a_n,\ G,\ A) \mid \exists J{\subseteq}\{1..n\}.\ \Sigma_{i\in J} g_i \leq G \wedge \Sigma_{i\in J} a_i \geq A\ \}$$

- **Betrachte Subprobleme** $KP(k,g)$
  - Verwende Gegenstände $1,..,k$ und Maximalgewicht $g{\leq}G$
  - Bestimme optimalen Nutzen $N(k,g)$
    - $N(k,0) = 0$ für alle k

ITERATIVE LÖSUNG FÜR KP

$$\boldsymbol{KP} = \{\ (g_1..g_n,\ a_1..a_n,\ G,\ A) \mid \exists J\subseteq\{1..n\}.\ \Sigma_{i\in J} g_i \leq G \wedge \Sigma_{i\in J} a_i \geq A\,\}$$

- **Betrachte Subprobleme** $KP(k,g)$
  - Verwende Gegenstände $1,..,k$ und Maximalgewicht $g \leq G$
  - Bestimme optimalen Nutzen $N(k,g)$
    - $N(k,0)=0$ für alle k
    - $N(0,g)=0$ für alle g

Iterative Lösung für KP

$$\boldsymbol{KP} = \{\ (g_1..g_n,\ a_1..a_n,\ G,\ A) \mid \exists J{\subseteq}\{1..n\}.\ \Sigma_{i\in J} g_i \leq G \wedge \Sigma_{i\in J} a_i \geq A\ \}$$

- **Betrachte Subprobleme** $KP(k,g)$
  - Verwende Gegenstände $1,..,k$ und Maximalgewicht $g{\leq}G$
  - Bestimme optimalen Nutzen $N(k,g)$
    - $N(k,0) = 0$ für alle k
    - $N(0,g) = 0$ für alle g
    - $N(k,g) = max\{N(k{-}1, g{-}g_k) + a_k,\ N(k{-}1,g)\}$

Iterative Lösung für KP

$$\boldsymbol{KP} = \{\ (g_1..g_n,\ a_1..a_n,\ G,\ A) \mid \exists J{\subseteq}\{1..n\}.\ \Sigma_{i\in J} g_i \leq G \wedge \Sigma_{i\in J} a_i \geq A\ \}$$

- **Betrachte Subprobleme $KP(k,g)$**
  - Verwende Gegenstände $1,..,k$ und Maximalgewicht $g{\leq}G$
  - Bestimme optimalen Nutzen $N(k,g)$
    - $N(k,0) = 0$ für alle k
    - $N(0,g) = 0$ für alle g
    - $N(k,g) = max\{N(k{-}1, g{-}g_k) + a_k,\ N(k{-}1,g)\}$
- **Löse Rucksackproblem $\boldsymbol{KP}$**
  - Es gilt $(g_1..g_n,\ a_1..a_n,\ G,\ A) \in KP \Leftrightarrow N(n,G){\geq}A$

Iterative Lösung für KP

$$\boldsymbol{KP} = \{\ (g_1..g_n,\ a_1..a_n,\ G,\ A) \mid \exists J{\subseteq}\{1..n\}.\ \Sigma_{i\in J} g_i \leq G \wedge \Sigma_{i\in J} a_i \geq A\ \}$$

- **Betrachte Subprobleme $KP(k,g)$**
  - Verwende Gegenstände $1,..,k$ und Maximalgewicht $g{\leq}G$
  - Bestimme optimalen Nutzen $N(k,g)$
    - $N(k,0) = 0$ für alle k
    - $N(0,g) = 0$ für alle g
    - $N(k,g) = max\{N(k{-}1, g{-}g_k) + a_k,\ N(k{-}1,g)\}$
- **Löse Rucksackproblem $\boldsymbol{KP}$**
  - Es gilt $(g_1..g_n,\ a_1..a_n,\ G,\ A) \in KP \Leftrightarrow N(n,G){\geq}A$
  - Gleichungen beschreiben rekursiven Algorithmus für $N(n,G)$

ITERATIVE LÖSUNG FÜR KP

$$\boldsymbol{KP} = \{\ (g_1..g_n,\ a_1..a_n,\ G,\ A) \mid \exists J{\subseteq}\{1..n\}.\ \Sigma_{i\in J} g_i \leq G \wedge \Sigma_{i\in J} a_i \geq A\ \}$$

- **Betrachte Subprobleme** $KP(k,g)$
  - Verwende Gegenstände $1,..,k$ und Maximalgewicht $g{\leq}G$
  - Bestimme optimalen Nutzen $N(k,g)$
    - $N(k,0) = 0$ für alle k
    - $N(0,g) = 0$ für alle g
    - $N(k,g) = max\{N(k{-}1, g{-}g_k) + a_k,\ N(k{-}1, g)\}$
- **Löse Rucksackproblem** $\boldsymbol{KP}$
  - Es gilt $(g_1..g_n,\ a_1..a_n,\ G,\ A) \in KP \Leftrightarrow N(n,G){\geq}A$
  - Gleichungen beschreiben rekursiven Algorithmus für $N(n,G)$
  - Tabellarischer Algorithmus bestimmt alle $N(k,g)$ mit $k{\leq}n$ und $g{\leq}G$

Iterative Lösung für KP

$$\boldsymbol{KP} = \{\ (g_1..g_n,\ a_1..a_n,\ G,\ A) \mid \exists J{\subseteq}\{1..n\}.\ \Sigma_{i\in J} g_i \leq G \wedge \Sigma_{i\in J} a_i \geq A\ \}$$

- **Betrachte Subprobleme** $KP(k,g)$
  - Verwende Gegenstände $1,..,k$ und Maximalgewicht $g{\leq}G$
  - Bestimme optimalen Nutzen $N(k,g)$
    - $N(k,0) = 0$ für alle k
    - $N(0,g) = 0$ für alle g
    - $N(k,g) = max\{N(k{-}1, g{-}g_k) + a_k,\ N(k{-}1,g)\}$
- **Löse Rucksackproblem** $\boldsymbol{KP}$
  - Es gilt $(g_1..g_n,\ a_1..a_n,\ G,\ A) \in KP \Leftrightarrow N(n,G){\geq}A$
  - Gleichungen beschreiben rekursiven Algorithmus für $N(n,G)$
  - Tabellarischer Algorithmus bestimmt alle $N(k,g)$ mit $k{\leq}n$ und $g{\leq}G$
  - Laufzeit ist $\mathcal{O}(n*G)$

Iterative Lösung für KP

$$\boldsymbol{KP} = \{\ (g_1..g_n,\ a_1..a_n,\ G,\ A) \mid \exists J \subseteq \{1..n\}.\ \Sigma_{i\in J} g_i \leq G \wedge \Sigma_{i\in J} a_i \geq A\ \}$$

- **Betrachte Subprobleme $KP(k, g)$**
  - Verwende Gegenstände $1, .., k$ und Maximalgewicht $g \leq G$
  - Bestimme optimalen Nutzen $N(k, g)$
    - $N(k, 0) = 0$ für alle k
    - $N(0, g) = 0$ für alle g
    - $N(k, g) = max\{N(k-1, g-g_k) + a_k,\ N(k-1, g)\}$
- **Löse Rucksackproblem $\boldsymbol{KP}$**
  - Es gilt $(g_1..g_n,\ a_1..a_n,\ G,\ A) \in KP \Leftrightarrow N(n, G) \geq A$
  - Gleichungen beschreiben rekursiven Algorithmus für $N(n, G)$
  - Tabellarischer Algorithmus bestimmt alle $N(k, g)$ mit $k \leq n$ und $g \leq G$
  - Laufzeit ist $\mathcal{O}(n * G)$

$$\Downarrow$$

$(g_1..g_n,\ a_1..a_n,\ G,\ A) \in KP$ ist IN $\mathcal{O}(n * G)$ Schritten lösbar

Liegt das Rucksackproblem KP etwa in $\mathcal{P}$?

Liegt das Rucksackproblem KP etwa in $\mathcal{P}$?

- **Lösung für $\boldsymbol{KP}$ ist nicht wirklich polynomiell**
  - $n * G$ kann exponentiell wachsen relativ zur Größe der Eingabe

Liegt das Rucksackproblem KP etwa in $\mathcal{P}$?

- **Lösung für $\boldsymbol{KP}$ ist nicht wirklich polynomiell**
  – $n * G$ kann exponentiell wachsen relativ zur Größe der Eingabe
  – Größe von $(g_1..g_n,\ a_1..a_n,\ G,\ A)$ ist $\mathcal{O}(n * (logG + logA))$

Liegt das Rucksackproblem KP etwa in $\mathcal{P}$?

- **Lösung für $\boldsymbol{KP}$ ist nicht wirklich polynomiell**
  - $n * G$ kann exponentiell wachsen relativ zur Größe der Eingabe
  - Größe von $(g_1..g_n,\ a_1..a_n,\ G,\ A)$ ist $\mathcal{O}(n * (logG + logA))$

- **$\boldsymbol{KP}$ ist ein Zahlproblem**
  - $M \subseteq X^*$ ist **Zahlproblem**, wenn es kein Polynom p gibt mit $MAX(w) \leq p(|w|)$ für alle $w \in X^*$

    $MAX(w)$ ist die größte im Wort w codierte Zahl

Liegt das Rucksackproblem KP etwa in $\mathcal{P}$?

- **Lösung für KP ist nicht wirklich polynomiell**
  - $n*G$ kann exponentiell wachsen relativ zur Größe der Eingabe
  - Größe von $(g_1..g_n,\ a_1..a_n,\ G,\ A)$ ist $\mathcal{O}(n*(logG+logA))$
- **KP ist ein Zahlproblem**
  - $M \subseteq X^*$ ist **Zahlproblem**, wenn es kein Polynom p gibt mit $MAX(w) \leq p(|w|)$ für alle $w \in X^*$

    $MAX(w)$ ist die größte im Wort w codierte Zahl
  - Weitere Zahlprobleme: $PARTITION$, BPP, TSP, MSP, ...

Liegt das Rucksackproblem KP etwa in $\mathcal{P}$?

- **Lösung für KP ist nicht wirklich polynomiell**
  - $n * G$ kann exponentiell wachsen relativ zur Größe der Eingabe
  - Größe von $(g_1..g_n,\ a_1..a_n,\ G,\ A)$ ist $\mathcal{O}(n * (logG + logA))$

- **KP ist ein Zahlproblem**
  - $M \subseteq X^*$ ist **Zahlproblem**, wenn es kein Polynom p gibt mit $MAX(w) \leq p(|w|)$ für alle $w \in X^*$

    $MAX(w)$ ist die größte im Wort w codierte Zahl
  - Weitere Zahlprobleme: $PARTITION$, BPP, TSP, MSP, ...
  - Keine Zahlprobleme: $CLIQUE$, VC, IS, SGI, LCS, DHC, HC, GC,...

Pseudopolynomielle Algorithmen

Liegt das Rucksackproblem KP etwa in $\mathcal{P}$?

- **Lösung für $\boldsymbol{KP}$ ist nicht wirklich polynomiell**
  - $n * G$ kann exponentiell wachsen relativ zur Größe der Eingabe
  - Größe von $(g_1..g_n,\ a_1..a_n,\ G,\ A)$ ist $\mathcal{O}(n * (logG + logA))$
- **$\boldsymbol{KP}$ ist ein Zahlproblem**
  - $M \subseteq X^*$ ist **Zahlproblem**, wenn es kein Polynom p gibt mit $MAX(w) \leq p(|w|)$ für alle $w \in X^*$

    $MAX(w)$ ist die größte im Wort w codierte Zahl
  - Weitere Zahlprobleme: $PARTITION$, BPP, TSP, MSP, ...
  - Keine Zahlprobleme: $CLIQUE$, VC, IS, SGI, LCS, DHC, HC, GC,...
- **$\boldsymbol{KP}$ hat pseudopolynomielle Lösung**
  - Ein Algorithmus für ein Zahlproblem $M \subseteq X^*$ ist pseudopolynomiell, wenn seine Rechenzeit durch ein Polynom in $|w|$ und $MAX(w)$ beschränkt ist

Starke $\mathcal{NP}$-vollständigkeit

- **Pseudopolynomiell** $\hat{=}$ **effizient bei kleinen Zahlen**
  - Ist $M \subseteq X^*$ pseudopolynomiell lösbar, so ist für jedes Polynom p
    $M_p \equiv \{w \in M \mid MAX(w) \leq p(|w|)\} \in \mathcal{P}$

Starke $\mathcal{NP}$-vollständigkeit

- **Pseudopolynomiell $\hat{=}$ effizient bei kleinen Zahlen**
  - Ist $M \subseteq X^*$ pseudopolynomiell lösbar, so ist für jedes Polynom p
    $M_p \equiv \{w \in M \mid MAX(w) \leq p(|w|)\} \in \mathcal{P}$
  - Die Restriktion von KP auf polynomiell große Gewichte liegt in $\mathcal{P}$

Starke $\mathcal{NP}$-vollständigkeit

- **Pseudopolynomiell $\hat{=}$ effizient bei kleinen Zahlen**
  - Ist $M \subseteq X^*$ pseudopolynomiell lösbar, so ist für jedes Polynom p
    $M_p \equiv \{w \in M \mid MAX(w) \leq p(|w|)\} \in \mathcal{P}$
  - Die Restriktion von KP auf polynomiell große Gewichte liegt in $\mathcal{P}$
  - Hat jedes Zahlproblem eine pseudopolynomielle Lösung?

STARKE $\mathcal{NP}$-VOLLSTÄNDIGKEIT

- **Pseudopolynomiell $\hat{=}$ effizient bei kleinen Zahlen**
  - Ist $M \subseteq X^*$ pseudopolynomiell lösbar, so ist für jedes Polynom p $M_p \equiv \{w \in M \mid MAX(w) \leq p(|w|)\} \in \mathcal{P}$
  - Die Restriktion von KP auf polynomiell große Gewichte liegt in $\mathcal{P}$
  - Hat jedes Zahlproblem eine pseudopolynomielle Lösung?

- ***TSP* ohne pseudopolynomielle Lösung** (falls $\mathcal{P} \neq \mathcal{NP}$)

Starke $\mathcal{NP}$-vollständigkeit

- **Pseudopolynomiell $\hat{=}$ effizient bei kleinen Zahlen**
  - Ist $M \subseteq X^*$ pseudopolynomiell lösbar, so ist für jedes Polynom p
    $M_p \equiv \{w \in M \mid MAX(w) \leq p(|w|)\} \in \mathcal{P}$
  - Die Restriktion von KP auf polynomiell große Gewichte liegt in $\mathcal{P}$
  - Hat jedes Zahlproblem eine pseudopolynomielle Lösung?

- ***TSP* ohne pseudopolynomielle Lösung** (falls $\mathcal{P} \neq \mathcal{NP}$)
  - Der Reduktionsbeweis $HC \leq_p TSP$ zeigt $HC \leq_p TSP_n$

Starke $\mathcal{NP}$-vollständigkeit

- **Pseudopolynomiell $\hat{=}$ effizient bei kleinen Zahlen**
  - Ist $M \subseteq X^*$ pseudopolynomiell lösbar, so ist für jedes Polynom p $M_p \equiv \{w \in M \mid MAX(w) \leq p(|w|)\} \in \mathcal{P}$
  - Die Restriktion von KP auf polynomiell große Gewichte liegt in $\mathcal{P}$
  - Hat jedes Zahlproblem eine pseudopolynomielle Lösung?

- ***TSP*** **ohne pseudopolynomielle Lösung** (falls $\mathcal{P} \neq \mathcal{NP}$)
  - Der Reduktionsbeweis $HC \leq_p TSP$ zeigt $HC \leq_p TSP_n$
  - Eine Restriktion von TSP auf kleine Zahlen bleibt $\mathcal{NP}$-vollständig

Starke $\mathcal{NP}$-vollständigkeit

- **Pseudopolynomiell $\hat{=}$ effizient bei kleinen Zahlen**
  - Ist $M{\subseteq}X^*$ pseudopolynomiell lösbar, so ist für jedes Polynom p $M_p \equiv \{w \in M \mid MAX(w) \leq p(|w|)\} \in \mathcal{P}$
  - Die Restriktion von KP auf polynomiell große Gewichte liegt in $\mathcal{P}$
  - Hat jedes Zahlproblem eine pseudopolynomielle Lösung?

- ***TSP* ohne pseudopolynomielle Lösung** (falls $\mathcal{P}{\neq}\mathcal{NP}$)
  - Der Reduktionsbeweis $HC{\leq_p}TSP$ zeigt $HC{\leq_p}TSP_n$
  - Eine Restriktion von TSP auf kleine Zahlen bleibt $\mathcal{NP}$-vollständig

- ***TSP* ist stark $\mathcal{NP}$-vollständig**
  - $M{\subseteq}X^*$ stark $\mathcal{NP}$-vollständig $\equiv$ M_p $\mathcal{NP}$-vollständig für ein Polynom p

Starke $\mathcal{NP}$-vollständigkeit

- **Pseudopolynomiell $\hat{=}$ effizient bei kleinen Zahlen**
  - Ist $M \subseteq X^*$ pseudopolynomiell lösbar, so ist für jedes Polynom p $M_p \equiv \{w \in M \mid MAX(w) \leq p(|w|)\} \in \mathcal{P}$
  - Die Restriktion von KP auf polynomiell große Gewichte liegt in $\mathcal{P}$
  - Hat jedes Zahlproblem eine pseudopolynomielle Lösung?
- ***TSP* ohne pseudopolynomielle Lösung** (falls $\mathcal{P} \neq \mathcal{NP}$)
  - Der Reduktionsbeweis $HC \leq_p TSP$ zeigt $HC \leq_p TSP_n$
  - Eine Restriktion von TSP auf kleine Zahlen bleibt $\mathcal{NP}$-vollständig
- ***TSP* ist stark $\mathcal{NP}$-vollständig**
  - $M \subseteq X^*$ stark $\mathcal{NP}$-vollständig $\equiv$ M_p $\mathcal{NP}$-vollständig für ein Polynom p
  - M stark $\mathcal{NP}$-vollständig $\Rightarrow$ M hat keine pseudopolynomielle Lösung

Starke $\mathcal{NP}$-vollständigkeit

- **Pseudopolynomiell $\hat{=}$ effizient bei kleinen Zahlen**
  - Ist $M \subseteq X^*$ pseudopolynomiell lösbar, so ist für jedes Polynom p
    $M_p \equiv \{w \in M \mid MAX(w) \leq p(|w|)\} \in \mathcal{P}$
  - Die Restriktion von KP auf polynomiell große Gewichte liegt in $\mathcal{P}$
  - Hat jedes Zahlproblem eine pseudopolynomielle Lösung?

- ***TSP* ohne pseudopolynomielle Lösung** (falls $\mathcal{P} \neq \mathcal{NP}$)
  - Der Reduktionsbeweis $HC \leq_p TSP$ zeigt $HC \leq_p TSP_n$
  - Eine Restriktion von TSP auf kleine Zahlen bleibt $\mathcal{NP}$-vollständig

- ***TSP* ist stark $\mathcal{NP}$-vollständig**
  - $M \subseteq X^*$ stark $\mathcal{NP}$-vollständig $\equiv$ M_p $\mathcal{NP}$-vollständig für ein Polynom p
  - M stark $\mathcal{NP}$-vollständig $\Rightarrow$ M hat keine pseudopolynomielle Lösung

$\Downarrow$

Einschränkung auf kleine Zahlen ist nur zuweilen eine Antwort auf das $\mathcal{P}$–$\mathcal{NP}$ Dilemma

Approximationsalgorithmen für Optimierungsprobleme

- **Viele Probleme haben Optimierungsvariante**

Approximationsalgorithmen für Optimierungsprobleme

- **Viele Probleme haben Optimierungsvariante**
  - $CLIQUE_{opt}$: bestimme die größte Clique im Graphen
  - TSP_{opt}: bestimme die kostengünstigste Rundreise
  - BPP_{opt}: bestimme die kleinste Anzahl der nötigen Behälter
  - KP_{opt}: bestimme das geringstmögliche Gewicht für einen festen Nutzen

Approximationsalgorithmen für Optimierungsprobleme

- **Viele Probleme haben Optimierungsvariante**
  - $CLIQUE_{opt}$: bestimme die größte Clique im Graphen
  - TSP_{opt}: bestimme die kostengünstigste Rundreise
  - BPP_{opt}: bestimme die kleinste Anzahl der nötigen Behälter
  - KP_{opt}: bestimme das geringstmögliche Gewicht für einen festen Nutzen

  Alle Probleme sind $\mathcal{NP}$-hart

Approximationsalgorithmen für Optimierungsprobleme

- **Viele Probleme haben Optimierungsvariante**
  - $CLIQUE_{opt}$: bestimme die größte Clique im Graphen
  - TSP_{opt}: bestimme die kostengünstigste Rundreise
  - BPP_{opt}: bestimme die kleinste Anzahl der nötigen Behälter
  - KP_{opt}: bestimme das geringstmögliche Gewicht für einen festen Nutzen

  Alle Probleme sind $\mathcal{NP}$-hart

  Wie effizient kann man eine optimale Lösung annähern?

Approximationsalgorithmen für Optimierungsprobleme

- **Viele Probleme haben Optimierungsvariante**
  - $CLIQUE_{opt}$: bestimme die größte Clique im Graphen
  - TSP_{opt}: bestimme die kostengünstigste Rundreise
  - BPP_{opt}: bestimme die kleinste Anzahl der nötigen Behälter
  - KP_{opt}: bestimme das geringstmögliche Gewicht für einen festen Nutzen

  Alle Probleme sind $\mathcal{NP}$-hart

  Wie effizient kann man eine optimale Lösung annähern?

- **Optimierungsproblem $M \subseteq X^*$**
  - Für $w \in X^*$ gibt es ggf. mehrere (akzeptable) Lösungen x mit $(w,x) \in M$

Approximationsalgorithmen für Optimierungsprobleme

- **Viele Probleme haben Optimierungsvariante**
  - $CLIQUE_{opt}$: bestimme die größte Clique im Graphen
  - TSP_{opt}: bestimme die kostengünstigste Rundreise
  - BPP_{opt}: bestimme die kleinste Anzahl der nötigen Behälter
  - KP_{opt}: bestimme das geringstmögliche Gewicht für einen festen Nutzen

  Alle Probleme sind $\mathcal{NP}$-hart

  Wie effizient kann man eine optimale Lösung annähern?

- **Optimierungsproblem $M \subseteq X^*$**
  - Für $w \in X^*$ gibt es ggf. mehrere (akzeptable) Lösungen x mit $(w, x) \in M$
  - $OPT_M(w)$: Wert einer optimalen (maxi-/minimalen) Lösung für $w \in X^*$

Approximationsalgorithmen für Optimierungsprobleme

- **Viele Probleme haben Optimierungsvariante**
  - $CLIQUE_{opt}$: bestimme die größte Clique im Graphen
  - TSP_{opt}: bestimme die kostengünstigste Rundreise
  - BPP_{opt}: bestimme die kleinste Anzahl der nötigen Behälter
  - KP_{opt}: bestimme das geringstmögliche Gewicht für einen festen Nutzen

  Alle Probleme sind $\mathcal{NP}$-hart

  Wie effizient kann man eine optimale Lösung annähern?

- **Optimierungsproblem $M \subseteq X^*$**
  - Für $w \in X^*$ gibt es ggf. mehrere (akzeptable) Lösungen x mit $(w, x) \in M$
  - $OPT_M(w)$: Wert einer optimalen (maxi-/minimalen) Lösung für $w \in X^*$

- **Approximationsalgorithmus A für $M \subseteq X^*$**
  - A berechnet für $w \in X^*$ ein $x = A(w)$ mit $(w, x) \in M$

Approximationsalgorithmen für Optimierungsprobleme

- **Viele Probleme haben Optimierungsvariante**
  - $CLIQUE_{opt}$: bestimme die größte Clique im Graphen
  - TSP_{opt}: bestimme die kostengünstigste Rundreise
  - BPP_{opt}: bestimme die kleinste Anzahl der nötigen Behälter
  - KP_{opt}: bestimme das geringstmögliche Gewicht für einen festen Nutzen

  Alle Probleme sind $\mathcal{NP}$-hart

  Wie effizient kann man eine optimale Lösung annähern?

- **Optimierungsproblem $M \subseteq X^*$**
  - Für $w \in X^*$ gibt es ggf. mehrere (akzeptable) Lösungen x mit $(w, x) \in M$
  - $OPT_M(w)$: Wert einer optimalen (maxi-/minimalen) Lösung für $w \in X^*$

- **Approximationsalgorithmus A für $M \subseteq X^*$**
  - A berechnet für $w \in X^*$ ein $x = A(w)$ mit $(w, x) \in M$
  - $R_A(w)$: Güte des Algorithmus A (normiertes Verhältnis $OPT_M(w)$ zu $A(w)$)

Approximationsalgorithmen für Optimierungsprobleme

- **Viele Probleme haben Optimierungsvariante**
  - $CLIQUE_{opt}$: bestimme die größte Clique im Graphen
  - TSP_{opt}: bestimme die kostengünstigste Rundreise
  - BPP_{opt}: bestimme die kleinste Anzahl der nötigen Behälter
  - KP_{opt}: bestimme das geringstmögliche Gewicht für einen festen Nutzen

  Alle Probleme sind $\mathcal{NP}$-hart

  Wie effizient kann man eine optimale Lösung annähern?

- **Optimierungsproblem $M \subseteq X^*$**
  - Für $w \in X^*$ gibt es ggf. mehrere (akzeptable) Lösungen x mit $(w,x) \in M$
  - $OPT_M(w)$: Wert einer optimalen (maxi-/minimalen) Lösung für $w \in X^*$

- **Approximationsalgorithmus A für $M \subseteq X^*$**
  - A berechnet für $w \in X^*$ ein $x = A(w)$ mit $(w,x) \in M$
  - $R_A(w)$: Güte des Algorithmus A (normiertes Verhältnis $OPT_M(w)$ zu $A(w)$)
  - R_A^∞: asymptotische worst-case Güte von A $(\inf\{r \geq 1 \mid \forall^\infty\, w \in X^*. R_A(w) \leq r\})$

Approximationsalgorithmen für Optimierungsprobleme

- **Viele Probleme haben Optimierungsvariante**
  - $CLIQUE_{opt}$: bestimme die größte Clique im Graphen
  - TSP_{opt}: bestimme die kostengünstigste Rundreise
  - BPP_{opt}: bestimme die kleinste Anzahl der nötigen Behälter
  - KP_{opt}: bestimme das geringstmögliche Gewicht für einen festen Nutzen

    Alle Probleme sind $\mathcal{NP}$-hart

  Wie effizient kann man eine optimale Lösung annähern?

- **Optimierungsproblem $M \subseteq X^*$**
  - Für $w \in X^*$ gibt es ggf. mehrere (akzeptable) Lösungen x mit $(w,x) \in M$
  - $OPT_M(w)$: Wert einer optimalen (maxi-/minimalen) Lösung für $w \in X^*$

- **Approximationsalgorithmus A für $M \subseteq X^*$**
  - A berechnet für $w \in X^*$ ein $x = A(w)$ mit $(w,x) \in M$
  - $R_A(w)$: Güte des Algorithmus A (normiertes Verhältnis $OPT_M(w)$ zu $A(w)$)
  - R_A^∞: asymptotische worst-case Güte von A $\left(\inf\{r \geq 1 \mid \forall^\infty\, w \in X^*. R_A(w) \leq r\}\right)$
  - $R_{min}(M,w) := \inf\{R_A^\infty(w) \mid A \text{ approximiert } M \text{ in polynomieller Zeit}\}$

Approximationsschemata für das Rucksackproblem

$$\boldsymbol{KP} = \{\ (g_1..g_n,\ a_1..a_n,\ G,\ A) \mid \exists J{\subseteq}\{1..n\}.\ \Sigma_{i\in J} g_i \leq G \wedge \Sigma_{i\in J} a_i \geq A\ \}$$

Approximationsschemata für das Rucksackproblem

$\boldsymbol{KP} = \{\ (g_1..g_n,\ a_1..a_n,\ G,\ A) \mid \exists J {\subseteq} \{1..n\}.\ \Sigma_{i\in J} g_i \leq G \wedge \Sigma_{i\in J} a_i \geq A\ \}$

- **Beliebig guter multiplikativer Fehler**
  - Für jedes ϵ gibt es einen Approximationsalgorithmus A mit Laufzeit $\mathcal{O}(n^3 * \epsilon^{-1})$ und Güte $R_A(w) \leq 1+\epsilon$ für alle w

Approximationsschemata für das Rucksackproblem

$$\boldsymbol{KP} = \{ \, (g_1..g_n,\, a_1..a_n,\, G,\, A) \mid \exists J{\subseteq}\{1..n\}.\ \Sigma_{i\in J} g_i \leq G \wedge \Sigma_{i\in J} a_i \geq A \, \}$$

- **Beliebig guter multiplikativer Fehler**
  - Für jedes ϵ gibt es einen Approximationsalgorithmus A mit Laufzeit $\mathcal{O}(n^3 * \epsilon^{-1})$ und Güte $R_A(w) \leq 1+\epsilon$ für alle w

- **Kein konstanter additiver Fehler möglich**
  - Für kein k gibt es einen polynomiellen Algorithmus A_{KP} mit der Eigenschaft $|OPT_{KP}(w) - A_{KP}(w)| \leq k$ für alle w

Approximationsschemata für das Rucksackproblem

$\boldsymbol{KP} = \{\ (g_1..g_n,\ a_1..a_n,\ G,\ A) \mid \exists J{\subseteq}\{1..n\}.\ \Sigma_{i\in J} g_i \leq G \wedge \Sigma_{i\in J} a_i \geq A\ \}$

- **Beliebig guter multiplikativer Fehler**
  - Für jedes ϵ gibt es einen Approximationsalgorithmus A mit Laufzeit $\mathcal{O}(n^3 * \epsilon^{-1})$ und Güte $R_A(w) {\leq} 1{+}\epsilon$ für alle w
- **Kein konstanter additiver Fehler möglich**
  - Für kein k gibt es einen polynomiellen Algorithmus A_{KP} mit der Eigenschaft $|OPT_{KP}(w) - A_{KP}(w)| {\leq} k$ für alle w

  Wenn es A_{KP} geben würde, dann entscheiden wir KP polynomiell wie folgt

Approximationsschemata für das Rucksackproblem

$$\boldsymbol{KP} = \{\ (g_1..g_n,\ a_1..a_n,\ G,\ A) \mid \exists J{\subseteq}\{1..n\}.\ \Sigma_{i\in J} g_i \leq G \wedge \Sigma_{i\in J} a_i \geq A\ \}$$

- **Beliebig guter multiplikativer Fehler**
  - Für jedes ϵ gibt es einen Approximationsalgorithmus A mit Laufzeit $\mathcal{O}(n^3 * \epsilon^{-1})$ und Güte $R_A(w) \leq 1+\epsilon$ für alle w
- **Kein konstanter additiver Fehler möglich**
  - Für kein k gibt es einen polynomiellen Algorithmus A_{KP} mit der Eigenschaft $|OPT_{KP}(w) - A_{KP}(w)| \leq k$ für alle w

  Wenn es A_{KP} geben würde, dann entscheiden wir KP polynomiell wie folgt
  - Transformiere $w = (g_1..g_n,\ a_1..a_n,\ G,\ A)$ in

$$w' = (g_1..g_n,\ a_1{*}(k{+}1)..a_n{*}(k{+}1),\ G,\ A{*}(k{+}1))$$

Approximationsschemata für das Rucksackproblem

$\boldsymbol{KP} = \{\ (g_1..g_n,\ a_1..a_n,\ G,\ A) \mid \exists J{\subseteq}\{1..n\}.\ \Sigma_{i\in J} g_i \leq G \wedge \Sigma_{i\in J} a_i \geq A\ \}$

- **Beliebig guter multiplikativer Fehler**
  - Für jedes ϵ gibt es einen Approximationsalgorithmus A mit Laufzeit $\mathcal{O}(n^3 * \epsilon^{-1})$ und Güte $R_A(w){\leq}1{+}\epsilon$ für alle w
- **Kein konstanter additiver Fehler möglich**
  - Für kein k gibt es einen polynomiellen Algorithmus A_{KP} mit der Eigenschaft $|OPT_{KP}(w) - A_{KP}(w)|{\leq}k$ für alle w

Wenn es A_{KP} geben würde, dann entscheiden wir KP polynomiell wie folgt

- Transformiere $w = (g_1..g_n,\ a_1..a_n,\ G,\ A)$ in

$$w' = (g_1..g_n,\ a_1{*}(k{+}1)..a_n{*}(k{+}1),\ G,\ A{*}(k{+}1))$$

- Wegen $|OPT_{KP}(w') - A_{KP}(w')|{\leq}k$ folgt

$$|OPT_{KP}(w) - \lfloor A_{KP}(w')/(k{+}1)\rfloor\,| \leq \lfloor k/(k{+}1)\rfloor = 0$$

Approximationsschemata für das Rucksackproblem

$\boldsymbol{KP} = \{\ (g_1..g_n,\ a_1..a_n,\ G,\ A)\ |\ \exists J\subseteq\{1..n\}.\ \Sigma_{i\in J} g_i \leq G \wedge \Sigma_{i\in J} a_i \geq A\ \}$

- **Beliebig guter multiplikativer Fehler**
  - Für jedes ϵ gibt es einen Approximationsalgorithmus A mit Laufzeit $\mathcal{O}(n^3 * \epsilon^{-1})$ und Güte $R_A(w) \leq 1+\epsilon$ für alle w
- **Kein konstanter additiver Fehler möglich**
  - Für kein k gibt es einen polynomiellen Algorithmus A_{KP} mit der Eigenschaft $|OPT_{KP}(w) - A_{KP}(w)| \leq k$ für alle w

  Wenn es A_{KP} geben würde, dann entscheiden wir KP polynomiell wie folgt
  - Transformiere $w = (g_1..g_n,\ a_1..a_n,\ G,\ A)$ in
  $$w' = (g_1..g_n,\ a_1*(k+1)..a_n*(k+1),\ G,\ A*(k+1))$$
  - Wegen $|OPT_{KP}(w') - A_{KP}(w')| \leq k$ folgt
  $$|OPT_{KP}(w) - \lfloor A_{KP}(w')/(k+1)\rfloor\,| \leq \lfloor k/(k+1)\rfloor = 0$$
  - Also gilt $w \in KP \Leftrightarrow \lfloor A_{KP}(w')/(k+1)\rfloor \geq A$

Approximationsschemata für das Rucksackproblem

$\boldsymbol{KP} = \{\ (g_1..g_n,\ a_1..a_n,\ G,\ A) \mid \exists J{\subseteq}\{1..n\}.\ \Sigma_{i\in J} g_i \leq G \wedge \Sigma_{i\in J} a_i \geq A\ \}$

- **Beliebig guter multiplikativer Fehler**
  – Für jedes ϵ gibt es einen Approximationsalgorithmus A mit Laufzeit $\mathcal{O}(n^3 * \epsilon^{-1})$ und Güte $R_A(w) \leq 1+\epsilon$ für alle w

- **Kein konstanter additiver Fehler möglich**
  – Für kein k gibt es einen polynomiellen Algorithmus A_{KP} mit der Eigenschaft $|OPT_{KP}(w) - A_{KP}(w)| \leq k$ für alle w

  Wenn es A_{KP} geben würde, dann entscheiden wir KP polynomiell wie folgt

  – Transformiere $w = (g_1..g_n,\ a_1..a_n,\ G,\ A)$ in

$$w' = (g_1..g_n,\ a_1{*}(k{+}1)..a_n{*}(k{+}1),\ G,\ A{*}(k{+}1))$$

  – Wegen $|OPT_{KP}(w') - A_{KP}(w')| \leq k$ folgt

$$|OPT_{KP}(w) - \lfloor A_{KP}(w')/(k{+}1)\rfloor\,| \leq \lfloor k/(k{+}1)\rfloor = 0$$

  – Also gilt $w \in KP \ \Leftrightarrow\ \lfloor A_{KP}(w')/(k{+}1)\rfloor \geq A$

Beweistechnik: Multiplikation des Problems, nachträgliche Division des Fehlers

Approximationsschemata für Binpacking

$$\boldsymbol{BPP} = \{ a_1, ...a_n,\ b,\ k) \mid \exists f: \{1..n\} \rightarrow \{1..k\}.\ \forall j \leq k.\ \sum_{i \in \{i|f(i)=j\}} a_i \leq b \}$$

- **Asymptotische Güte 11/9 erreichbar**

Approximationsschemata für Binpacking

$\boldsymbol{BPP} = \{ a_1, ...a_n,\ b,\ k) \mid \exists f : \{1..n\} \rightarrow \{1..k\}.\ \forall j \leq k.\ \sum_{i\,\in\,\{i|f(i)=j\}} a_i \leq b \}$

- **Asymptotische Güte 11/9 erreichbar**
  – FIRST-FIT DECREASING: Sortiere Objekte in absteigender Reihenfolge und packe sie jeweils in die erste freie Kiste, in der genügend Platz ist

Approximationsschemata für Binpacking

$\boldsymbol{BPP} = \{\, a_1, ...a_n,\ b,\ k) \mid \exists f : \{1..n\} \rightarrow \{1..k\}.\ \forall j \leq k.\ \sum_{i \in \{i|f(i)=j\}} a_i \leq b \,\}$

- **Asymptotische Güte 11/9 erreichbar**
  - FIRST-FIT DECREASING: Sortiere Objekte in absteigender Reihenfolge und packe sie jeweils in die erste freie Kiste, in der genügend Platz ist
  - Es gilt $FFD(w) = 11/9{*}OPT_{BPP}(w) + 4$ für alle w

  $\mapsto$ Polynomielle Approximation mit $R_A^\infty = 11/9$

Approximationsschemata für Binpacking

$\boldsymbol{BPP} = \{ a_1, ...a_n,\ b,\ k) \mid \exists f : \{1..n\}{\to}\{1..k\}.\ \forall j \leq k.\ \sum_{i \in \{i|f(i)=j\}} a_i \leq b \}$

- **Asymptotische Güte 11/9 erreichbar**
  - FIRST-FIT DECREASING: Sortiere Objekte in absteigender Reihenfolge und packe sie jeweils in die erste freie Kiste, in der genügend Platz ist
  - Es gilt $FFD(w) = 11/9{*}OPT_{BPP}(w) + 4$ für alle w
  - $\mapsto$ Polynomielle Approximation mit $R_A^\infty = 11/9$
- **Keine absolute Güte besser als 3/2** (falls $\mathcal{P}{\neq}\mathcal{NP}$)
  - Kein polynomieller Approximationsalgorithmus mit $R_A < 3/2$ möglich

Approximationsschemata für Binpacking

$\boldsymbol{BPP} = \{ a_1, ...a_n,\ b,\ k) \mid \exists f : \{1..n\} \rightarrow \{1..k\}.\ \forall j \leq k.\ \sum_{i \in \{i \mid f(i)=j\}} a_i \leq b \}$

- **Asymptotische Güte 11/9 erreichbar**
  - FIRST-FIT DECREASING: Sortiere Objekte in absteigender Reihenfolge und packe sie jeweils in die erste freie Kiste, in der genügend Platz ist
  - Es gilt $FFD(w) = 11/9{*}OPT_{BPP}(w) + 4$ für alle w
  - $\mapsto$ Polynomielle Approximation mit $R_A^\infty = 11/9$
- **Keine absolute Güte besser als 3/2** (falls $\mathcal{P} \neq \mathcal{NP}$)
  - Kein polynomieller Approximationsalgorithmus mit $R_A < 3/2$ möglich
  - Die Reduktion $PARTITION \leq_p BPP$ benutzt 2 Behälter der Größe $S := \sum_{i=1}^{n} a_i/2$, auf die Zahlen verteilt werden

Approximationsschemata für Binpacking

$\boldsymbol{BPP} = \{ a_1, ...a_n,\ b,\ k) \mid \exists f : \{1..n\} \rightarrow \{1..k\}.\ \forall j \leq k.\ \sum_{i \in \{i|f(i)=j\}} a_i \leq b \}$

- **Asymptotische Güte 11/9 erreichbar**
  - FIRST-FIT DECREASING: Sortiere Objekte in absteigender Reihenfolge und packe sie jeweils in die erste freie Kiste, in der genügend Platz ist
  - Es gilt $FFD(w) = 11/9 * OPT_{BPP}(w) + 4$ für alle w
  - $\mapsto$ Polynomielle Approximation mit $R_A^\infty = 11/9$
- **Keine absolute Güte besser als 3/2** (falls $\mathcal{P} \neq \mathcal{NP}$)
  - Kein polynomieller Approximationsalgorithmus mit $R_A < 3/2$ möglich
  - Die Reduktion $PARTITION \leq_p BPP$ benutzt 2 Behälter der Größe $S := \sum_{i=1}^n a_i/2$, auf die Zahlen verteilt werden
  - Jeder Approximationsalgorithmus A mit $R_A < 3/2$ liefert
    $A(w) = 2$, falls $w \in PARTITION$ und sonst $A(w) \geq 3$

Approximationsschemata für Binpacking

$BPP = \{\, a_1, ...a_n,\ b,\ k) \mid \exists f : \{1..n\} \to \{1..k\}.\ \forall j \leq k.\ \sum_{i \in \{i|f(i)=j\}} a_i \leq b \,\}$

- **Asymptotische Güte 11/9 erreichbar**
  – FIRST-FIT DECREASING: Sortiere Objekte in absteigender Reihenfolge und packe sie jeweils in die erste freie Kiste, in der genügend Platz ist
  – Es gilt $FFD(w) = 11/9 * OPT_{BPP}(w) + 4$ für alle w
  $\mapsto$ Polynomielle Approximation mit $R_A^\infty = 11/9$

- **Keine absolute Güte besser als 3/2** (falls $\mathcal{P} \neq \mathcal{NP}$)
  – Kein polynomieller Approximationsalgorithmus mit $R_A < 3/2$ möglich
  – Die Reduktion $PARTITION \leq_p BPP$ benutzt 2 Behälter der Größe $S := \sum_{i=1}^n a_i/2$, auf die Zahlen verteilt werden
  – Jeder Approximationsalgorithmus A mit $R_A < 3/2$ liefert
    $A(w) = 2$, falls $w \in PARTITION$ und sonst $A(w) \geq 3$
  – Wegen $PARTITION \in \mathcal{NPC}$ kann A nicht polynomiell sein

Approximationsschemata für Binpacking

$\boldsymbol{BPP} = \{ a_1, ...a_n,\ b,\ k) \mid \exists f : \{1..n\} \to \{1..k\}.\ \forall j \leq k.\ \sum_{i \in \{i|f(i)=j\}} a_i \leq b \}$

- **Asymptotische Güte 11/9 erreichbar**
  - FIRST-FIT DECREASING: Sortiere Objekte in absteigender Reihenfolge und packe sie jeweils in die erste freie Kiste, in der genügend Platz ist
  - Es gilt $FFD(w) = 11/9 * OPT_{BPP}(w) + 4$ für alle w
  - $\mapsto$ Polynomielle Approximation mit $R_A^\infty = 11/9$
- **Keine absolute Güte besser als 3/2** (falls $\mathcal{P} \neq \mathcal{NP}$)
  - Kein polynomieller Approximationsalgorithmus mit $R_A < 3/2$ möglich
  - Die Reduktion $PARTITION \leq_p BPP$ benutzt 2 Behälter der Größe $S := \sum_{i=1}^n a_i/2$, auf die Zahlen verteilt werden
  - Jeder Approximationsalgorithmus A mit $R_A < 3/2$ liefert
    $A(w) = 2$, falls $w \in PARTITION$ und sonst $A(w) \geq 3$
  - Wegen $PARTITION \in \mathcal{NPC}$ kann A nicht polynomiell sein

Beweistechnik: Einbettung eines $\mathcal{NP}$-vollständigen Entscheidungsproblems

Approximation des Traveling Salesman Problems

$$\boldsymbol{TSP} = \{\ c_{12}, ..., c_{n-1,n},\ B \mid\ \exists\pi{:}\{1..n\}{\rightarrow}\{1..n\}.\ \pi \text{ bijektiv} \\ \wedge\ \sum_{i=1}^{n-1} c_{\pi(i)\pi(i+1)} + c_{\pi(n)\pi(1)} \leq B\ \}$$

Approximation des Traveling Salesman Problems

$$\boldsymbol{TSP} = \{\ c_{12}, ..., c_{n-1,n},\ B \mid\ \exists\pi{:}\{1..n\}{\to}\{1..n\}.\ \pi \text{ bijektiv} \\ \wedge\ \sum_{i=1}^{n-1} c_{\pi(i)\pi(i+1)} + c_{\pi(n)\pi(1)} \leq B\ \}$$

- **$R_A^\infty = 3/2$ erreichbar bei Dreiecksungleichung**
  - Direkte Verbindung ist kürzer als ein Umweg: $\forall i, j, k.\ c_{i,j}{\leq}c_{i,k}{+}c_{k,j}$

Approximation des Traveling Salesman Problems

$$\boldsymbol{TSP} = \{\ c_{12}, ..., c_{n-1,n},\ B \mid\ \exists\pi{:}\{1..n\}{\rightarrow}\{1..n\}.\ \pi \text{ bijektiv} \\ \wedge \sum_{i=1}^{n-1} c_{\pi(i)\pi(i+1)} + c_{\pi(n)\pi(1)} \leq B\ \}$$

- **$R_A^\infty = 3/2$ erreichbar bei Dreiecksungleichung**
  - Direkte Verbindung ist kürzer als ein Umweg: $\forall i,j,k.\ c_{i,j} \leq c_{i,k} + c_{k,j}$
- **Keine endliche Grenze für multiplikativen Fehler**
  - Es gibt keinen polynomiellen Algorithmus A mit $R_A^\infty = r$ für ein $r \in \mathbb{N}$

Approximation des Traveling Salesman Problems

$$\boldsymbol{TSP} = \{\ c_{12}, ..., c_{n-1,n},\ B \mid\ \exists\pi{:}\{1..n\}{\to}\{1..n\}.\ \pi \text{ bijektiv} \\ \wedge \sum_{i=1}^{n-1} c_{\pi(i)\pi(i+1)} + c_{\pi(n)\pi(1)} \leq B\ \}$$

- **$R_A^\infty = 3/2$ erreichbar bei Dreiecksungleichung**
  – Direkte Verbindung ist kürzer als ein Umweg: $\forall i, j, k.\ c_{i,j}{\leq}c_{i,k}{+}c_{k,j}$

- **Keine endliche Grenze für multiplikativen Fehler**
  – Es gibt keinen polynomiellen Algorithmus A mit $R_A^\infty{=}r$ für ein $r \in \mathbb{N}$

  Wenn es A geben würde, dann entscheiden wir HC polynomiell wie folgt

Approximation des Traveling Salesman Problems

$$\boldsymbol{TSP} = \{\ c_{12}, ..., c_{n-1,n},\ B \mid\ \exists\pi{:}\{1..n\}{\rightarrow}\{1..n\}.\ \pi \text{ bijektiv} \\ \wedge \sum_{i=1}^{n-1} c_{\pi(i)\pi(i+1)} + c_{\pi(n)\pi(1)} \leq B\ \}$$

- **$R_A^\infty = 3/2$ erreichbar bei Dreiecksungleichung**
  - Direkte Verbindung ist kürzer als ein Umweg: $\forall i,j,k.\ c_{i,j}{\leq}c_{i,k}{+}c_{k,j}$
- **Keine endliche Grenze für multiplikativen Fehler**
  - Es gibt keinen polynomiellen Algorithmus A mit $R_A^\infty{=}r$ für ein $r \in \mathbb{N}$

  Wenn es A geben würde, dann entscheiden wir HC polynomiell wie folgt
  - Transformiere einen Graphen $G = (V, E)$ in $w = c_{12}, ..., c_{n-1,n},\ |V|$
    mit $c_{ij} = 1$ falls $\{i,j\} \in E$ und $c_{ij} = r|V| + 1$ sonst

Approximation des Traveling Salesman Problems

$$\boldsymbol{TSP} = \{\ c_{12}, ..., c_{n-1,n},\ B \mid\ \exists \pi{:}\{1..n\}{\to}\{1..n\}.\ \pi \text{ bijektiv} \\ \wedge \sum_{i=1}^{n-1} c_{\pi(i)\pi(i+1)} + c_{\pi(n)\pi(1)} \leq B\ \}$$

- **$R_A^\infty = 3/2$ erreichbar bei Dreiecksungleichung**
  – Direkte Verbindung ist kürzer als ein Umweg: $\forall i, j, k.\ c_{i,j} \leq c_{i,k} + c_{k,j}$

- **Keine endliche Grenze für multiplikativen Fehler**
  – Es gibt keinen polynomiellen Algorithmus A mit $R_A^\infty = r$ für ein $r \in \mathbb{N}$

  Wenn es A geben würde, dann entscheiden wir HC polynomiell wie folgt
  – Transformiere einen Graphen $G = (V, E)$ in $w = c_{12}, ..., c_{n-1,n},\ |V|$
  mit $c_{ij} = 1$ falls $\{i, j\} \in E$ und $c_{ij} = r|V| + 1$ sonst
  – Dann $G \in HC \Rightarrow OPT_{TSP}(w) = |V|$ und $G \notin HC \Rightarrow OPT_{TSP}(w) > (r+1) * |V|$

Approximation des Traveling Salesman Problems

$$\boldsymbol{TSP} = \{\ c_{12}, ..., c_{n-1,n},\ B \mid\ \exists\pi{:}\{1..n\}{\rightarrow}\{1..n\}.\ \pi \text{ bijektiv} \\ \wedge \sum_{i=1}^{n-1} c_{\pi(i)\pi(i+1)} + c_{\pi(n)\pi(1)} \leq B\ \}$$

- **$R_A^\infty = 3/2$ erreichbar bei Dreiecksungleichung**
  - Direkte Verbindung ist kürzer als ein Umweg: $\forall i, j, k.\ c_{i,j} \leq c_{i,k} + c_{k,j}$

- **Keine endliche Grenze für multiplikativen Fehler**
  - Es gibt keinen polynomiellen Algorithmus A mit $R_A^\infty = r$ für ein $r \in \mathbb{N}$

  Wenn es A geben würde, dann entscheiden wir HC polynomiell wie folgt
  - Transformiere einen Graphen $G = (V, E)$ in $w = c_{12}, ..., c_{n-1,n},\ |V|$ mit $c_{ij} = 1$ falls $\{i, j\} \in E$ und $c_{ij} = r|V| + 1$ sonst
  - Dann $G \in HC \Rightarrow OPT_{TSP}(w) = |V|$ und $G \notin HC \Rightarrow OPT_{TSP}(w) > (r+1) * |V|$
  - Für große Graphen: $A(w) \leq r * OPT_{TSP}(w)$ also $G \in HC \ \Leftrightarrow\ A(w) \leq r * |V|$

    (Für kleine Graphen verwende den exponentiellen Entscheidungsalgorithmus)

Approximation des Traveling Salesman Problems

$$\boldsymbol{TSP} = \{\ c_{12}, ..., c_{n-1,n},\ B \mid\ \exists \pi{:}\{1..n\}{\to}\{1..n\}.\ \pi \text{ bijektiv} \\ \wedge \sum_{i=1}^{n-1} c_{\pi(i)\pi(i+1)} + c_{\pi(n)\pi(1)} \leq B\ \}$$

- **$R_A^\infty = 3/2$ erreichbar bei Dreiecksungleichung**
  – Direkte Verbindung ist kürzer als ein Umweg: $\forall i, j, k.\ c_{i,j} \leq c_{i,k} + c_{k,j}$

- **Keine endliche Grenze für multiplikativen Fehler**
  – Es gibt keinen polynomiellen Algorithmus A mit $R_A^\infty = r$ für ein $r \in \mathbb{N}$

  Wenn es A geben würde, dann entscheiden wir HC polynomiell wie folgt

  – Transformiere einen Graphen $G = (V, E)$ in $w = c_{12}, ..., c_{n-1,n},\ |V|$
  mit $c_{ij} = 1$ falls $\{i, j\} \in E$ und $c_{ij} = r|V| + 1$ sonst

  – Dann $G \in HC \Rightarrow OPT_{TSP}(w) = |V|$ und $G \notin HC \Rightarrow OPT_{TSP}(w) > (r+1) * |V|$

  – Für große Graphen: $A(w) \leq r * OPT_{TSP}(w)$ also $G \in HC \ \Leftrightarrow\ A(w) \leq r * |V|$
  (Für kleine Graphen verwende den exponentiellen Entscheidungsalgorithmus)

Beweistechnik: Reduktion auf $\mathcal{NP}$-vollständiges Problem mit Multiplikation des Kostenunterschieds zwischen positiver und negativer Antwort

Travelling Salesman mit Dreiecksungleichung

$$\boldsymbol{TSP_\Delta} = \{\ c_{12}, ..., c_{n-1,n},\ B \mid \forall i,j,k.\ c_{i,j} \leq c_{i,k} + c_{k,j}\ \wedge\ \exists \pi{:}\{1..n\} \to \{1..n\}.\ \pi \text{ bijektiv } \wedge \sum_{i=1}^{n-1} c_{\pi(i)\pi(i+1)} + c_{\pi(n)\pi(1)} \leq B\ \}$$

- **Approximationsalgorithmus**
  - Zu $w = c_{12}, ..., c_{n-1,n},\ B$ konstruiere vollständigen Graphen $G = (V, E)$ mit $V = \{v_1, .., v_n\}$ und Gewichten $c_{i,j}$ für $\{v_i, v_j\} \in E$

Travelling Salesman mit Dreiecksungleichung

$$\boldsymbol{TSP_\Delta} = \{\ c_{12}, ..., c_{n-1,n},\ B \mid\ \forall i,j,k.\ c_{i,j} \leq c_{i,k} + c_{k,j}\ \wedge\ \exists \pi{:}\{1..n\} \rightarrow \{1..n\}.\ \pi \text{ bijektiv } \wedge \sum_{i=1}^{n-1} c_{\pi(i)\pi(i+1)} + c_{\pi(n)\pi(1)} \leq B\ \}$$

- **Approximationsalgorithmus**
  - Zu $w = c_{12}, ..., c_{n-1,n},\ B$ konstruiere vollständigen Graphen $G = (V, E)$ mit $V = \{v_1, .., v_n\}$ und Gewichten $c_{i,j}$ für $\{v_i, v_j\} \in E$
  - Konstruiere spannenden Baum $T = (V, E_T)$ mit minimaler Kantensumme

    Beginnend mit $E_T = \emptyset$, $V_T = \{v_1\}$ wiederhole bis $V_T = V$
    - Wähle Kante $\{v_i, v_j\}$ mit minimalem Gewicht, so daß $v_i \in V_T$, $v_j \notin V_T$
    - Setze $V_T := V_T \cup \{v_j\}$ und $E_T := E_T \cup \{v_i, v_j\}$

Travelling Salesman mit Dreiecksungleichung

$$\boldsymbol{TSP_\Delta} = \{\ c_{12}, ..., c_{n-1,n},\ B \mid \forall i,j,k.\ c_{i,j} \leq c_{i,k} + c_{k,j}\ \wedge\ \exists \pi{:}\{1..n\} \to \{1..n\}.\ \pi \text{ bijektiv } \wedge \sum_{i=1}^{n-1} c_{\pi(i)\pi(i+1)} + c_{\pi(n)\pi(1)} \leq B\ \}$$

- **Approximationsalgorithmus**
  - Zu $w = c_{12}, ..., c_{n-1,n},\ B$ konstruiere vollständigen Graphen $G = (V, E)$ mit $V = \{v_1, .., v_n\}$ und Gewichten $c_{i,j}$ für $\{v_i, v_j\} \in E$
  - Konstruiere spannenden Baum $T = (V, E_T)$ mit minimaler Kantensumme

    Beginnend mit $E_T = \emptyset$, $V_T = \{v_1\}$ wiederhole bis $V_T = V$
    - Wähle Kante $\{v_i, v_j\}$ mit minimalem Gewicht, so daß $v_i \in V_T$, $v_j \notin V_T$
    - Setze $V_T := V_T \cup \{v_j\}$ und $E_T := E_T \cup \{v_i, v_j\}$
  - Durchlaufe T so, daß jede Kante genau zweimal benutzt wird

Travelling Salesman mit Dreiecksungleichung

$$\boldsymbol{TSP_\Delta} = \{\ c_{12}, ..., c_{n-1,n},\ B \mid\ \forall i,j,k.\ c_{i,j} \leq c_{i,k} + c_{k,j}\ \wedge\ \exists \pi{:}\{1..n\} \rightarrow \{1..n\}.\ \pi \text{ bijektiv}\ \wedge\ \sum_{i=1}^{n-1} c_{\pi(i)\pi(i+1)} + c_{\pi(n)\pi(1)} \leq B\ \}$$

- **Approximationsalgorithmus**
  - Zu $w = c_{12}, ..., c_{n-1,n},\ B$ konstruiere vollständigen Graphen $G = (V, E)$ mit $V = \{v_1, .., v_n\}$ und Gewichten $c_{i,j}$ für $\{v_i, v_j\} \in E$
  - Konstruiere spannenden Baum $T = (V, E_T)$ mit minimaler Kantensumme

    Beginnend mit $E_T = \emptyset$, $V_T = \{v_1\}$ wiederhole bis $V_T = V$
    - Wähle Kante $\{v_i, v_j\}$ mit minimalem Gewicht, so daß $v_i \in V_T$, $v_j \notin V_T$
    - Setze $V_T := V_T \cup \{v_j\}$ und $E_T := E_T \cup \{v_i, v_j\}$
  - Durchlaufe T so, daß jede Kante genau zweimal benutzt wird
  - Verkürze den entstandenen Rundweg so, daß einem Knoten zum nächsten noch nicht angesteuerten Knoten verzweigt wird

Travelling Salesman mit Dreiecksungleichung

$\boldsymbol{TSP_\Delta} = \{\ c_{12}, ..., c_{n-1,n},\ B \mid\ \forall i,j,k.\ c_{i,j} \leq c_{i,k} + c_{k,j}\ \wedge\ \exists \pi{:}\{1..n\} \rightarrow \{1..n\}.$
$\pi \text{ bijektiv } \wedge \sum_{i=1}^{n-1} c_{\pi(i)\pi(i+1)} + c_{\pi(n)\pi(1)} \leq B\ \}$

- **Approximationsalgorithmus**
  - Zu $w = c_{12}, ..., c_{n-1,n},\ B$ konstruiere vollständigen Graphen $G = (V, E)$ mit $V = \{v_1, .., v_n\}$ und Gewichten $c_{i,j}$ für $\{v_i, v_j\} \in E$
  - Konstruiere spannenden Baum $T = (V, E_T)$ mit minimaler Kantensumme

    Beginnend mit $E_T = \emptyset$, $V_T = \{v_1\}$ wiederhole bis $V_T = V$
    - Wähle Kante $\{v_i, v_j\}$ mit minimalem Gewicht, so daß $v_i \in V_T$, $v_j \notin V_T$
    - Setze $V_T := V_T \cup \{v_j\}$ und $E_T := E_T \cup \{v_i, v_j\}$
  - Durchlaufe T so, daß jede Kante genau zweimal benutzt wird
  - Verkürze den entstandenen Rundweg so, daß einem Knoten zum nächsten noch nicht angesteuerten Knoten verzweigt wird
- **Laufzeit des Algorithmus ist $O(n^3)$**

Travelling Salesman mit Dreiecksungleichung

$$\boldsymbol{TSP_\Delta} = \{\ c_{12}, ..., c_{n-1,n},\ B \mid \forall i,j,k.\ c_{i,j} \leq c_{i,k} + c_{k,j}\ \wedge\ \exists \pi{:}\{1..n\} \to \{1..n\}.\ \pi \text{ bijektiv } \wedge \sum_{i=1}^{n-1} c_{\pi(i)\pi(i+1)} + c_{\pi(n)\pi(1)} \leq B\ \}$$

- **Approximationsalgorithmus**
  – Zu $w = c_{12}, ..., c_{n-1,n},\ B$ konstruiere vollständigen Graphen $G = (V, E)$ mit $V = \{v_1, .., v_n\}$ und Gewichten $c_{i,j}$ für $\{v_i, v_j\} \in E$
  – Konstruiere spannenden Baum $T = (V, E_T)$ mit minimaler Kantensumme
    Beginnend mit $E_T = \emptyset$, $V_T = \{v_1\}$ wiederhole bis $V_T = V$
    · Wähle Kante $\{v_i, v_j\}$ mit minimalem Gewicht, so daß $v_i \in V_T$, $v_j \notin V_T$
    · Setze $V_T := V_T \cup \{v_j\}$ und $E_T := E_T \cup \{v_i, v_j\}$
  – Durchlaufe T so, daß jede Kante genau zweimal benutzt wird
  – Verkürze den entstandenen Rundweg so, daß einem Knoten zum nächsten noch nicht angesteuerten Knoten verzweigt wird
- **Laufzeit des Algorithmus ist $O(n^3)$**
- **Güte des Algorithmus ist $R_A^\infty \leq 3/2$** (aufwendig)

“Approximation” einer Entscheidung

- **Verhalten gesteuert durch Zufallszahlen**
  – Falsche Entscheidung kann nicht ausgeschlossen werden

“Approximation” einer Entscheidung

- **Verhalten gesteuert durch Zufallszahlen**
  - Falsche Entscheidung kann nicht ausgeschlossen werden
  - Approximation $\equiv$ Verringerung der Fehlerwahrscheinlichkeit
  - Fehlerwahrscheinlichkeit unter 2^{-100} besser als die von Hardwarefehlern

“Approximation” einer Entscheidung

- **Verhalten gesteuert durch Zufallszahlen**
  - Falsche Entscheidung kann nicht ausgeschlossen werden
  - Approximation $\equiv$ Verringerung der Fehlerwahrscheinlichkeit
  - Fehlerwahrscheinlichkeit unter 2^{-100} besser als die von Hardwarefehlern
- **Anwendungen**
  - Primzahltest in linearer Zeit
  - Optimierung von Quicksort auf $\mathcal{O}(n{*}logn)$ (Bestimmung Pivotelement)

"Approximation" einer Entscheidung

- **Verhalten gesteuert durch Zufallszahlen**
  - Falsche Entscheidung kann nicht ausgeschlossen werden
  - Approximation $\equiv$ Verringerung der Fehlerwahrscheinlichkeit
  - Fehlerwahrscheinlichkeit unter 2^{-100} besser als die von Hardwarefehlern
- **Anwendungen**
  - Primzahltest in linearer Zeit
  - Optimierung von Quicksort auf $\mathcal{O}(n{*}logn)$ (Bestimmung Pivotelement)
- **Wie weist man gut Eigenschaften nach?**

“Approximation” einer Entscheidung

- **Verhalten gesteuert durch Zufallszahlen**
  – Falsche Entscheidung kann nicht ausgeschlossen werden
  – Approximation $\equiv$ Verringerung der Fehlerwahrscheinlichkeit
  – Fehlerwahrscheinlichkeit unter 2^{-100} besser als die von Hardwarefehlern

- **Anwendungen**
  – Primzahltest in linearer Zeit
  – Optimierung von Quicksort auf $\mathcal{O}(n{*}logn)$ (Bestimmung Pivotelement)

- **Wie weist man gut Eigenschaften nach?**
  – Einfaches Modell für probabilistische Algorithmen formulieren

"Approximation" einer Entscheidung

- **Verhalten gesteuert durch Zufallszahlen**
  - Falsche Entscheidung kann nicht ausgeschlossen werden
  - Approximation $\equiv$ Verringerung der Fehlerwahrscheinlichkeit
  - Fehlerwahrscheinlichkeit unter 2^{-100} besser als die von Hardwarefehlern
- **Anwendungen**
  - Primzahltest in linearer Zeit
  - Optimierung von Quicksort auf $\mathcal{O}(n*logn)$ (Bestimmung Pivotelement)
- **Wie weist man gut Eigenschaften nach?**
  - Einfaches Modell für probabilistische Algorithmen formulieren
  - Eigenschaften abstrakter probabilistischer Sprachklassen analysieren

- **Probabilistische Turingmaschine**
  - Struktur: $\boldsymbol{\tau} = (S,\, X,\, \Gamma,\, \delta,\, s_0,\, b)$
  - Zustandsüberführungsfunktion: $\delta{:}S{\times}\Gamma \rightarrow (S{\times}\Gamma{\times}\{r,l,h\})^2$

    Jede Alternative wird mit Wahrscheinlichkeit 1/2 ausgewählt

- **Probabilistische Turingmaschine**
  - Struktur: $\boldsymbol{\tau} = (S,\ X,\ \Gamma,\ \delta,\ s_0,\ b)$
  - Zustandsüberführungsfunktion: $\delta{:}S\times\Gamma \rightarrow (S\times\Gamma\times\{r,l,h\})^2$

    Jede Alternative wird mit Wahrscheinlichkeit 1/2 ausgewählt
  - Ausgabe: $h_\tau(w) \in \{0, 1, ?\}$ (Akzeptieren – Verwerfen – keine Aussage)

- **Probabilistische Turingmaschine**
  - Struktur: $\boldsymbol{\tau} = (S,\ X,\ \Gamma,\ \delta,\ s_0,\ b)$
  - Zustandsüberführungsfunktion: $\delta{:}S\times\Gamma \longrightarrow (S\times\Gamma\times\{r,l,h\})^2$

    Jede Alternative wird mit Wahrscheinlichkeit 1/2 ausgewählt
  - Ausgabe: $h_\tau(w) \in \{0, 1, ?\}$ (Akzeptieren – Verwerfen – keine Aussage)
  - Rechenzeit: maximale Rechenzeit aller möglichen Rechenwege

- **Probabilistische Turingmaschine**
  - Struktur: $\boldsymbol{\tau} = (S,\, X,\, \Gamma,\, \delta,\, s_0,\, b)$
  - Zustandsüberführungsfunktion: $\delta{:}S\times\Gamma \rightarrow (S\times\Gamma\times\{r,l,h\})^2$

    Jede Alternative wird mit Wahrscheinlichkeit 1/2 ausgewählt
  - Ausgabe: $h_\tau(w) \in \{0, 1, ?\}$ (Akzeptieren – Verwerfen – keine Aussage)
  - Rechenzeit: maximale Rechenzeit aller möglichen Rechenwege
  - PTM: polynomiell zeitbeschränkte probabilistische Turingmaschine

Probabilistische Berechnungsmodelle

- **Probabilistische Turingmaschine**
  - Struktur: $\boldsymbol{\tau} = (S,\ X,\ \Gamma,\ \delta,\ s_0,\ b)$
  - Zustandsüberführungsfunktion: $\delta{:}S{\times}\Gamma \to (S{\times}\Gamma{\times}\{r,l,h\})^2$

    Jede Alternative wird mit Wahrscheinlichkeit 1/2 ausgewählt
  - Ausgabe: $h_\tau(w) \in \{0, 1, ?\}$ (Akzeptieren – Verwerfen – keine Aussage)
  - Rechenzeit: maximale Rechenzeit aller möglichen Rechenwege
  - PTM: polynomiell zeitbeschränkte probabilistische Turingmaschine
- **Abstrakteres Modell: Probabilistische Algorithmen**
  - Programme mit zufälligen Entscheidungen
  - Abstrakte Komplexität wie bisher

Probabilistische Berechnungsmodelle

- **Probabilistische Turingmaschine**
  - Struktur: $\boldsymbol{\tau} = (S,\ X,\ \Gamma,\ \delta,\ s_0,\ b)$
  - Zustandsüberführungsfunktion: $\delta{:}S{\times}\Gamma \to (S{\times}\Gamma{\times}\{r,l,h\})^2$

    Jede Alternative wird mit Wahrscheinlichkeit 1/2 ausgewählt
  - Ausgabe: $h_\tau(w) \in \{0, 1, ?\}$ (Akzeptieren – Verwerfen – keine Aussage)
  - Rechenzeit: maximale Rechenzeit aller möglichen Rechenwege
  - PTM: polynomiell zeitbeschränkte probabilistische Turingmaschine
- **Abstrakteres Modell: Probabilistische Algorithmen**
  - Programme mit zufälligen Entscheidungen
  - Abstrakte Komplexität wie bisher

Was kann man mit polynomiell zeitbeschränkten probabilistischen Algorithmen erreichen?

Wichtige Probabilistische Sprachklassen

- **PP: Probabilistic Polynomial** Monte-Carlo-Algorithmen
  - Wahrscheinlichkeit für korrekte Antwort größer als 1/2
  - PP $= \{L \mid \exists\ \text{PTM}\ \tau.\ \forall w.\ Prob(\ h_\tau(w){=}\chi_L(w)\ ) > 1/2\}$

Wichtige Probabilistische Sprachklassen

- **PP: Probabilistic Polynomial** Monte-Carlo-Algorithmen
  - Wahrscheinlichkeit für korrekte Antwort größer als 1/2
  - PP $= \{L \mid \exists$ PTM $\tau.\ \forall w.\ Prob(\ h_\tau(w)=\chi_L(w)\ ) > 1/2\}$
- **BPP: Bounded error Probabilistic Polynomial**
  - Wahrscheinlichkeit für korrekte Antwort größer als $1/2+\epsilon$
  - BPP $= \{L \mid \exists$ PTM $\tau.\ \exists\epsilon > 0\ \forall w.\ Prob(\ h_\tau(w)=\chi_L(w)\ ) > 1/2+\epsilon\}$

Wichtige Probabilistische Sprachklassen

- **PP: Probabilistic Polynomial** Monte-Carlo-Algorithmen
  - Wahrscheinlichkeit für korrekte Antwort größer als 1/2
  - PP $= \{L \mid \exists$ PTM $\tau.\ \forall w.\ Prob(\ h_\tau(w){=}\chi_L(w)\ ) > 1/2\}$
- **BPP: Bounded error Probabilistic Polynomial**
  - Wahrscheinlichkeit für korrekte Antwort größer als $1/2{+}\epsilon$
  - BPP $= \{L \mid \exists$ PTM $\tau.\ \exists\epsilon{>}\ 0\ \forall w.\ Prob(\ h_\tau(w){=}\chi_L(w)\ ) > 1/2{+}\epsilon\}$
- **RP: Random Polynomial**
  - Nichtzugehörige korrekt identifiziert, andere mit Wahrscheinlichkeit $> 1/2$
  - RP $= \{L \mid \exists$ PTM $\tau.\ \forall w{\in}L.\ Prob(\ h_\tau(w){=}1\ ) > 1/2$
    $\wedge\ \forall w{\notin}L.\ Prob(\ h_\tau(w){=}0\ ) = 1\}$

Wichtige Probabilistische Sprachklassen

- **PP: Probabilistic Polynomial** Monte-Carlo-Algorithmen
  - Wahrscheinlichkeit für korrekte Antwort größer als 1/2
  - PP $= \{L \mid \exists$ PTM $\tau.\ \forall w.\ Prob(\ h_\tau(w){=}\chi_L(w)\ ) > 1/2\}$
- **BPP: Bounded error Probabilistic Polynomial**
  - Wahrscheinlichkeit für korrekte Antwort größer als $1/2{+}\epsilon$
  - BPP $= \{L \mid \exists$ PTM $\tau.\ \exists\epsilon{>}\ 0\ \forall w.\ Prob(\ h_\tau(w){=}\chi_L(w)\ ) > 1/2{+}\epsilon\}$
- **RP: Random Polynomial**
  - Nichtzugehörige korrekt identifiziert, andere mit Wahrscheinlichkeit $> 1/2$
  - RP $= \{L \mid \exists$ PTM $\tau.\ \forall w \in L.\ Prob(\ h_\tau(w){=}1\ ) > 1/2$
    $\wedge\ \forall w \notin L.\ Prob(\ h_\tau(w){=}0\ ) = 1\}$
- **ZPP: Zero error PP** Las-Vegas-Algorithmen
  - Wahrscheinlichkeit für korrekte Antwort $> 1/2$, keine falschen Antworten
  - ZPP $= \{L \mid \exists$ PTM $\tau.$
    $\forall w \in L.\ (\ Prob(\ h_\tau(w){=}1\ ) > 1/2\ \wedge\ Prob(\ h_\tau(w){=}0\ ) = 0\ )$
    $\wedge\ \forall w \notin L.\ Prob(\ h_\tau(w){=}0\ ) > 1/2\ \wedge\ Prob(\ h_\tau(w){=}1\ ) = 0\ )\}$

Probabilistischer Primzahltest für $n \geq 3$ (Solovay/Strassen)

1. Wenn n gerade ist: Antwort "keine Primzahl"

Probabilistischer Primzahltest für $n \geq 3$ (Solovay/Strassen)

1. Wenn n gerade ist: Antwort "keine Primzahl"
2. Ansonsten wähle $a \in \{1...n\}$ zufällig

Probabilistischer Primzahltest für $n \geq 3$ (Solovay/Strassen)

1. Wenn n gerade ist: Antwort "keine Primzahl"
2. Ansonsten wähle $a \in \{1...n\}$ zufällig
3. Falls $gcd(n, a) \neq 1$: Antwort "keine Primzahl"

Probabilistischer Primzahltest für $n \geq 3$ (Solovay/Strassen)

1. Wenn n gerade ist: Antwort "keine Primzahl"
2. Ansonsten wähle $a \in \{1...n\}$ zufällig
3. Falls $gcd(n,a) \neq 1$: Antwort "keine Primzahl"
4. Ansonsten setze $\epsilon := a^{(n-1)/2} (mod\ n)$

   $\delta := J(a,n)$ *(Jacobi Symbol)*

PROBABILISTISCHER PRIMZAHLTEST FÜR $n\geq 3$ (Solovay/Strassen)

1. Wenn n gerade ist: Antwort "keine Primzahl"

2. Ansonsten wähle $a \in \{1...n\}$ zufällig

3. Falls $gcd(n, a) \neq 1$: Antwort "keine Primzahl"

4. Ansonsten setze $\epsilon := a^{(n-1)/2}(mod\ n)$

   $\delta := J(a, n)$ *(Jacobi Symbol)*

5. Falls $\epsilon = \delta$: Antwort "Primzahl"

Probabilistischer Primzahltest für $n \geq 3$ (Solovay/Strassen)

1. Wenn n gerade ist: Antwort "keine Primzahl"
2. Ansonsten wähle $a \in \{1...n\}$ zufällig
3. Falls $gcd(n,a) \neq 1$: Antwort "keine Primzahl"
4. Ansonsten setze $\epsilon := a^{(n-1)/2} (mod\ n)$
   $\delta := J(a,n)$ *(Jacobi Symbol)*
5. Falls $\epsilon = \delta$: Antwort "Primzahl"
6. Ansonsten: Antwort "keine Primzahl"

Probabilistischer Primzahltest für $n \geq 3$ (Solovay/Strassen)

1. Wenn n gerade ist: Antwort "keine Primzahl"
2. Ansonsten wähle $a \in \{1...n\}$ zufällig
3. Falls $gcd(n, a) \neq 1$: Antwort "keine Primzahl"
4. Ansonsten setze $\epsilon := a^{(n-1)/2} (mod\ n)$
   $\delta := J(a, n)$ *(Jacobi Symbol)*
5. Falls $\epsilon = \delta$: Antwort "Primzahl"
6. Ansonsten: Antwort "keine Primzahl"

RP-Algorithmus

– Korrekte Ausgabe, falls n Primzahl

– Fehlerwahrscheinlichkeit unter 1/2, falls n keine Primzahl

Probabilistischer Primzahltest für $n \geq 3$ (Solovay/Strassen)

1. Wenn n gerade ist: Antwort "keine Primzahl"
2. Ansonsten wähle $a \in \{1...n\}$ zufällig
3. Falls $gcd(n, a) \neq 1$: Antwort "keine Primzahl"
4. Ansonsten setze $\epsilon := a^{(n-1)/2} (mod\ n)$
   $\delta := J(a, n)$ *(Jacobi Symbol)*
5. Falls $\epsilon = \delta$: Antwort "Primzahl"
6. Ansonsten: Antwort "keine Primzahl"

RP-Algorithmus

– Korrekte Ausgabe, falls n Primzahl

– Fehlerwahrscheinlichkeit unter 1/2, falls n keine Primzahl

Rechenzeit $\leq 6 * \log n$

Eigenschaften probabilistischer Sprachklassen

- **k-fache Iteration von RP Algorithmen verringert die Wahrscheinlichkeit einer falschen Antwort auf 2^{-k}**

Eigenschaften probabilistischer Sprachklassen

- **k-fache Iteration von RP Algorithmen verringert die Wahrscheinlichkeit einer falschen Antwort auf 2^{-k}**
  - Ist τ die k-fache statistisch unabhängige Iteration einer PTM für $L \in RP$, so gilt

$$\forall w \in L. Prob(\ h_\tau(w)=1\ ) > 1-2^{-k}\ \wedge\ \forall w \notin L. Prob(\ h_\tau(w)=0\ ) = 1$$

Eigenschaften probabilistischer Sprachklassen

- **k-fache Iteration von RP Algorithmen verringert die Wahrscheinlichkeit einer falschen Antwort auf 2^{-k}**
  - Ist τ die k-fache statistisch unabhängige Iteration einer PTM für $L \in RP$, so gilt

$$\forall w \in L. Prob(\ h_\tau(w){=}1\ ) > 1{-}2^{-k}\ \wedge\ \forall w \notin L. Prob(\ h_\tau(w){=}0\ ) = 1$$

- **t-fache Iteration eines BPP Algorithmus für $t > \frac{k}{-\log(1-4\epsilon^2)}$ verringert die Wahrscheinlichkeit der falschen Antwort auf 2^{-k}**

Eigenschaften probabilistischer Sprachklassen

- **k-fache Iteration von RP Algorithmen verringert die Wahrscheinlichkeit einer falschen Antwort auf 2^{-k}**
  - Ist τ die k-fache statistisch unabhängige Iteration einer PTM für $L \in RP$, so gilt

$$\forall w \in L. Prob(\ h_\tau(w)=1\ ) > 1-2^{-k} \ \wedge\ \forall w \notin L. Prob(\ h_\tau(w)=0\ ) = 1$$

- **t-fache Iteration eines BPP Algorithmus für $t > \frac{k}{-\log(1-4\epsilon^2)}$ verringert die Wahrscheinlichkeit der falschen Antwort auf 2^{-k}**
  - Sei τ^t die $(2t+1)$-fache statistisch unabhängige Iteration einer PTM τ für $L \in BPP$, die genau dann akzeptiert, wenn τ mindestens $t+1$-mal akzeptiert, so gilt für $t > \frac{k-1}{-\log(1-4\epsilon^2)}$

$$\forall w.\ Prob(\ h_{\tau^t}(w)=\chi_L(w)\ ) > 1-2^{-k}$$

Wegener 75–77

Sprachklassenhierarchie