

Theoretische Informatik II

Einheit 6.4

Grenzen überwinden



1. Approximation von Optimierungsproblemen
2. Probabilistische Algorithmen
3. Anwendungen in der Kryptographie

WIE KANN MAN “UNLÖSBARE” PROBLEME ANGEHEN?

- **Viele Probleme sind nachweislich schlecht lösbar**

- Terminierung, Korrektheit, Äquivalenz von Programmen (unentscheidbar)
- Gültigkeit prädikatenlogischer Formeln (unentscheidbar)
- SAT Solver / Hardwareverifikation (\mathcal{NP} - oder $PSPACE$ -vollständig)
- Strategische Spiele / Marktanalysen ($PSPACE$ -vollständig)
- Scheduling, Navigation, Verteilungsprobleme (\mathcal{NP} -vollständig)
- Erzeugung kryptographischer Schlüssel (\mathcal{NP} bzw. $\mathcal{O}(n^6)$)

WIE KANN MAN “UNLÖSBARE” PROBLEME ANGEHEN?

- **Viele Probleme sind nachweislich schlecht lösbar**
 - Terminierung, Korrektheit, Äquivalenz von Programmen (unentscheidbar)
 - Gültigkeit prädikatenlogischer Formeln (unentscheidbar)
 - SAT Solver / Hardwareverifikation (\mathcal{NP} - oder $PSPACE$ -vollständig)
 - Strategische Spiele / Marktanalysen ($PSPACE$ -vollständig)
 - Scheduling, Navigation, Verteilungsprobleme (\mathcal{NP} -vollständig)
 - Erzeugung kryptographischer Schlüssel (\mathcal{NP} bzw. $\mathcal{O}(n^6)$)
- **Lösungen werden dennoch gebraucht**
 - Die Probleme tauchen in der Praxis auf
 - Aufgeben ist keine akzeptable Antwort

WIE KANN MAN “UNLÖSBARE” PROBLEME ANGEHEN?

- **Viele Probleme sind nachweislich schlecht lösbar**
 - Terminierung, Korrektheit, Äquivalenz von Programmen (unentscheidbar)
 - Gültigkeit prädikatenlogischer Formeln (unentscheidbar)
 - SAT Solver / Hardwareverifikation (\mathcal{NP} - oder $PSPACE$ -vollständig)
 - Strategische Spiele / Marktanalysen ($PSPACE$ -vollständig)
 - Scheduling, Navigation, Verteilungsprobleme (\mathcal{NP} -vollständig)
 - Erzeugung kryptographischer Schlüssel (\mathcal{NP} bzw. $\mathcal{O}(n^6)$)
- **Lösungen werden dennoch gebraucht**
 - Die Probleme tauchen in der Praxis auf
 - Aufgeben ist keine akzeptable Antwort
- **Versuche die Unlösbarkeiten zu umgehen**
 - Suche Alternativen zur perfekten Lösung, die es nicht geben kann
 - Entwickle Verfahren zur Konstruktion “suboptimaler Lösungen”
 - Untersuche die Qualität dieser Lösungen relativ zum Optimum

WIE KANN MAN “UNLÖSBARE” PROBLEME ANGEHEN?

- **Viele Probleme sind nachweislich schlecht lösbar**

- Terminierung, Korrektheit, Äquivalenz von Programmen (unentscheidbar)
- Gültigkeit prädikatenlogischer Formeln (unentscheidbar)
- SAT Solver / Hardwareverifikation (\mathcal{NP} - oder $PSPACE$ -vollständig)
- Strategische Spiele / Marktanalysen ($PSPACE$ -vollständig)
- Scheduling, Navigation, Verteilungsprobleme (\mathcal{NP} -vollständig)
- Erzeugung kryptographischer Schlüssel (\mathcal{NP} bzw. $\mathcal{O}(n^6)$)

- **Lösungen werden dennoch gebraucht**

- Die Probleme tauchen in der Praxis auf
- Aufgeben ist keine akzeptable Antwort

- **Versuche die Unlösbarkeiten zu umgehen**

- Suche Alternativen zur perfekten Lösung, die es nicht geben kann
- Entwickle Verfahren zur Konstruktion “suboptimaler Lösungen”
- Untersuche die Qualität dieser Lösungen relativ zum Optimum

Suche nach neuen Lösungsmöglichkeiten liefert tieferes Verständnis

UNKONVENTIONELLE LÖSUNG “UNLÖSBARER” PROBLEME

- **Heuristische Lösung unentscheidbarer Probleme**

- **Verzicht auf Vollständigkeit** zugunsten einer “Entscheidung”
- Algorithmus “versucht” Standardlösungsweg und gibt auf, wenn dieser nicht zum Erfolg führt (Künstliche Intelligenz, Theorembeweisen, Verifikation)
- Algorithmus verwendet **Selbstorganisation** statt vorgefertigter Lösung (Lernverfahren, Neuronale Netze, genetische Algorithmen, ...)

UNKONVENTIONELLE LÖSUNG “UNLÖSBARER” PROBLEME

- **Heuristische Lösung unentscheidbarer Probleme**

- **Verzicht auf Vollständigkeit** zugunsten einer “Entscheidung”
- Algorithmus “versucht” Standardlösungsweg und gibt auf, wenn dieser nicht zum Erfolg führt (Künstliche Intelligenz, Theorembeweisen, Verifikation)
- Algorithmus verwendet **Selbstorganisation** statt vorgefertigter Lösung (Lernverfahren, Neuronale Netze, genetische Algorithmen, ...)

- **Approximationsverfahren**

- **Verzicht auf Optimalität** zugunsten einer schnellen Antwort
- Algorithmus bestimmt **Näherungslösung** anstelle des Optimums
- Liefert effiziente “Lösung” schwerer Optimierungsprobleme

UNKONVENTIONELLE LÖSUNG “UNLÖSBARER” PROBLEME

● **Heuristische Lösung unentscheidbarer Probleme**

- **Verzicht auf Vollständigkeit** zugunsten einer “Entscheidung”
- Algorithmus “versucht” Standardlösungsweg und gibt auf, wenn dieser nicht zum Erfolg führt (Künstliche Intelligenz, Theorembeweisen, Verifikation)
- Algorithmus verwendet **Selbstorganisation** statt vorgefertigter Lösung (Lernverfahren, Neuronale Netze, genetische Algorithmen, ...)

● **Approximationsverfahren**

- **Verzicht auf Optimalität** zugunsten einer schnellen Antwort
- Algorithmus bestimmt **Näherungslösung** anstelle des Optimums
- Liefert effiziente “Lösung” schwerer Optimierungsprobleme

● **Probabilistische Algorithmen**

- **Verzicht auf perfekte Korrektheit** zugunsten einer schnellen Antwort
- Algorithmus verwendet **Zufallsvariablen** bei Bestimmung der Lösung
Antwort kann mit geringer Fehlerwahrscheinlichkeit auch falsch sein
- Liefert effiziente “Lösung” schwerer Entscheidungsprobleme

- **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

- **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

- **Approximation umgeht Komplexitätsproblematik**

- Polynomielle Algorithmen können Näherungslösungen bestimmen
- Die Näherung kann niemals optimal sein (wenn $\mathcal{P} \neq \mathcal{NP}$)
- Ziel ist, so nahe wie möglich an das Optimum heranzukommen

- **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

- **Approximation umgeht Komplexitätsproblematik**

- Polynomielle Algorithmen können Näherungslösungen bestimmen
- Die Näherung kann niemals optimal sein (wenn $\mathcal{P} \neq \mathcal{NP}$)
- Ziel ist, so nahe wie möglich an das Optimum heranzukommen

- **Wie gut können Näherungslösungen sein?**

- Vergleiche Approximation mit bestmöglicher Lösung
- Wie gut ist das Ergebnis eines konkreten Approximationsalgorithmus?
- Was ist das günstigste Ergebnis, das überhaupt erreichbar ist

APPROXIMATIONS-LÖSUNG FÜR TRAVELLING SALESMAN

- **Anwendbar für TSP mit Dreiecksungleichung**

- Direkte Verbindung sind kürzer als Umwege: $\forall i, j, k. c_{i,j} \leq c_{i,k} + c_{k,j}$

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

APPROXIMATIONS-LÖSUNG FÜR TRAVELLING SALESMAN

- **Anwendbar für TSP mit Dreiecksungleichung**

- Direkte Verbindung sind kürzer als Umwege: $\forall i, j, k. c_{i,j} \leq c_{i,k} + c_{k,j}$

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

- **Approximationsalgorithmus**

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

- Konstruiere **minimal spannenden Baum** $T = (V, E_T)$

- Durchlaufe T so, daß jede Kante genau zweimal benutzt wird

- Verkürze den entstandenen Rundweg so, daß von einem Knoten zum nächsten noch nicht angesteuerten Knoten gesprungen wird

APPROXIMATIONS-LÖSUNG FÜR TRAVELLING SALESMAN

- **Anwendbar für TSP mit Dreiecksungleichung**

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

- **Approximationsalgorithmus**

- Zu $w = c_{12}, \dots, c_{n-1,n}, B$ konstruiere vollständigen Graphen $G=(V, E)$ mit $V = \{v_1, \dots, v_n\}$ und Gewichten $c_{i,j}$ für $\{v_i, v_j\} \in E$
- Konstruiere **minimal spannenden Baum** $T = (V, E_T)$
- Durchlaufe T so, daß jede Kante genau zweimal benutzt wird
- Verkürze den entstandenen Rundweg so, daß von einem Knoten zum nächsten noch nicht angesteuerten Knoten gesprungen wird

- **Erzeugte Lösung maximal doppelt so teuer wie nötig**

- Gewicht minimal spannender Bäume geringer als optimale Rundreise
- Wegen Dreiecksungleichung steigen bei Verkürzung die Kosten nicht

APPROXIMATIONS-LÖSUNG FÜR TRAVELLING SALESMAN

- **Anwendbar für TSP mit Dreiecksungleichung**

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

- **Approximationsalgorithmus**

- Zu $w = c_{12}, \dots, c_{n-1,n}, B$ konstruiere vollständigen Graphen $G=(V, E)$ mit $V = \{v_1, \dots, v_n\}$ und Gewichten $c_{i,j}$ für $\{v_i, v_j\} \in E$
- Konstruiere **minimal spannenden Baum** $T = (V, E_T)$
- Durchlaufe T so, daß jede Kante genau zweimal benutzt wird
- Verkürze den entstandenen Rundweg so, daß von einem Knoten zum nächsten noch nicht angesteuerten Knoten gesprungen wird

- **Erzeugte Lösung maximal doppelt so teuer wie nötig**

- Gewicht minimal spannender Bäume geringer als optimale Rundreise
- Wegen Dreiecksungleichung steigen bei Verkürzung die Kosten nicht

- **Laufzeit des Algorithmus ist $O(n^2)$**

- Kruskal Algorithmus (Einheit 6.1) hat Laufzeit $\mathcal{O}(|V| + |E| \log |E|)$

WIE GUT SIND APPROXIMATIONSLSÖSUNGEN?

- **Beschreibung von Optimierungsproblemen**

- Betrachte zugehöriges Entscheidungsproblem L als Menge aller akzeptablen Lösungen (x, y) für eine Eingabe x

- z.B. $CLIQUE := \{(G, k) \mid G \text{ hat Clique der Größe } k\}$

- Bei Eingabe G sucht Optimierung größtes k mit $(G, k) \in CLIQUE$

WIE GUT SIND APPROXIMATIONSLSÖSUNGEN?

- **Beschreibung von Optimierungsproblemen**

- Betrachte zugehöriges Entscheidungsproblem L als Menge aller akzeptablen Lösungen (x, y) für eine Eingabe x
 - z.B. $CLIQUE := \{(G, k) \mid G \text{ hat Clique der Größe } k\}$
Bei Eingabe G sucht Optimierung größtes k mit $(G, k) \in CLIQUE$
- Bestimme den Wert $W(x, y)$ einer Lösung $(x, y) \in L$
- $OPT_L(x)$: Wert einer optimalen Lösung für Eingabe x

WIE GUT SIND APPROXIMATIONSLSÖSUNGEN?

● Beschreibung von **Optimierungsproblemen**

- Betrachte zugehöriges Entscheidungsproblem L als Menge aller akzeptablen Lösungen (x, y) für eine Eingabe x
 - z.B. $CLIQUE := \{(G, k) \mid G \text{ hat Clique der Größe } k\}$
Bei Eingabe G sucht Optimierung größtes k mit $(G, k) \in CLIQUE$
- Bestimme den Wert $W(x, y)$ einer Lösung $(x, y) \in L$
- $OPT_L(x)$: Wert einer optimalen Lösung für Eingabe x

● Güte von Approximationsalgorithmen

- Algorithmus A berechne für jede Eingabe x ein $y=A(x)$ mit $(x, y) \in L$
- $R_A(x)$: Güte des Algorithmus A bei Eingabe x ($R_A(x) > 1$ gilt immer)
 - $R_A(x) \equiv \max \{OPT_L(x)/W(x, A(x)), W(x, A(x))/OPT_L(x)\}$

WIE GUT SIND APPROXIMATIONSLSÖSUNGEN?

● Beschreibung von **Optimierungsproblemen**

- Betrachte zugehöriges Entscheidungsproblem L als Menge aller akzeptablen Lösungen (x, y) für eine Eingabe x
 - z.B. $CLIQUE := \{(G, k) \mid G \text{ hat Clique der Größe } k\}$
Bei Eingabe G sucht Optimierung größtes k mit $(G, k) \in CLIQUE$
- Bestimme den Wert $W(x, y)$ einer Lösung $(x, y) \in L$
- $OPT_L(x)$: Wert einer optimalen Lösung für Eingabe x

● Güte von Approximationsalgorithmen

- Algorithmus A berechne für jede Eingabe x ein $y=A(x)$ mit $(x, y) \in L$
- $R_A(x)$: Güte des Algorithmus A bei Eingabe x ($R_A(x) > 1$ gilt immer)
 - $R_A(x) \equiv \max \{OPT_L(x)/W(x, A(x)), W(x, A(x))/OPT_L(x)\}$

● Asymptotische Güte von Approximationen

- $R_A^\infty = \inf\{r \geq 1 \mid \forall^\infty x. R_A(x) \leq r\}$: asymptotische worst-case Güte

WIE GUT SIND APPROXIMATIONSLSÖSUNGEN?

• Beschreibung von **Optimierungsproblemen**

- Betrachte zugehöriges Entscheidungsproblem L als Menge aller akzeptablen Lösungen (x, y) für eine Eingabe x
 - z.B. $CLIQUE := \{(G, k) \mid G \text{ hat Clique der Größe } k\}$
Bei Eingabe G sucht Optimierung größtes k mit $(G, k) \in CLIQUE$
- Bestimme den Wert $W(x, y)$ einer Lösung $(x, y) \in L$
- $OPT_L(x)$: Wert einer optimalen Lösung für Eingabe x

• Güte von Approximationsalgorithmen

- Algorithmus A berechne für jede Eingabe x ein $y=A(x)$ mit $(x, y) \in L$
- $R_A(x)$: Güte des Algorithmus A bei Eingabe x ($R_A(x) > 1$ gilt immer)
 - $R_A(x) \equiv \max \{OPT_L(x)/W(x, A(x)), W(x, A(x))/OPT_L(x)\}$

• Asymptotische Güte von Approximationen

- $R_A^\infty = \inf\{r \geq 1 \mid \forall^\infty x. R_A(x) \leq r\}$: asymptotische worst-case Güte

Finde bestmögliche worst-case Güte von Problemen

POSITIVE ERGEBNISSE

- **TSP:** $R_A^\infty = 3/2$ erreichbar bei Dreiecksungleichung
 - Verkürzter Durchlauf minimal spannender Bäume liefert Güte $R_A^\infty \leq 2$
 - Mit lokalen Optimierungen kommt man auf $R_A^\infty \leq 3/2$ (aufwendige Analyse)

POSITIVE ERGEBNISSE

- **TSP:** $R_A^\infty = 3/2$ erreichbar bei Dreiecksungleichung
 - Verkürzter Durchlauf minimal spannender Bäume liefert Güte $R_A^\infty \leq 2$
 - Mit lokalen Optimierungen kommt man auf $R_A^\infty \leq 3/2$ (aufwendige Analyse)
- **Knapsack:** beliebig kleiner multiplikativer Fehler
 - Für jedes ϵ gibt es einen Approximationsalgorithmus A mit Laufzeit $\mathcal{O}(n^3 * \epsilon^{-1})$ und Güte $R_A(x) \leq 1 + \epsilon$ für alle x (ohne Beweis)

POSITIVE ERGEBNISSE

- **TSP:** $R_A^\infty = 3/2$ erreichbar bei Dreiecksungleichung
 - Verkürzter Durchlauf minimal spannender Bäume liefert Güte $R_A^\infty \leq 2$
 - Mit lokalen Optimierungen kommt man auf $R_A^\infty \leq 3/2$ (aufwendige Analyse)
- **Knapsack:** beliebig kleiner multiplikativer Fehler
 - Für jedes ϵ gibt es einen Approximationsalgorithmus A mit Laufzeit $\mathcal{O}(n^3 * \epsilon^{-1})$ und Güte $R_A(x) \leq 1 + \epsilon$ für alle x (ohne Beweis)
- **Binpacking:** Asymptotische Güte $11/9$ erreichbar
 - *First-Fit Decreasing*: Sortiere Objekte in absteigender Reihenfolge und packe sie jeweils in erste freie Kiste, in der genügend Platz ist
 - Es gilt $W(x, FFD(x)) = 11/9 * OPT_{BPP}(x) + 4$ für alle x (ohne Beweis)
also $R_A^\infty = 11/9$

POSITIVE ERGEBNISSE

- **TSP:** $R_A^\infty = 3/2$ erreichbar bei Dreiecksungleichung
 - Verkürzter Durchlauf minimal spannender Bäume liefert Güte $R_A^\infty \leq 2$
 - Mit lokalen Optimierungen kommt man auf $R_A^\infty \leq 3/2$ (aufwendige Analyse)
- **Knapsack:** beliebig kleiner multiplikativer Fehler
 - Für jedes ϵ gibt es einen Approximationsalgorithmus A mit Laufzeit $\mathcal{O}(n^3 * \epsilon^{-1})$ und Güte $R_A(x) \leq 1 + \epsilon$ für alle x (ohne Beweis)
- **Binpacking:** Asymptotische Güte $11/9$ erreichbar
 - *First-Fit Decreasing*: Sortiere Objekte in absteigender Reihenfolge und packe sie jeweils in erste freie Kiste, in der genügend Platz ist
 - Es gilt $W(x, FFD(x)) = 11/9 * OPT_{BPP}(x) + 4$ für alle x (ohne Beweis)
also $R_A^\infty = 11/9$

Bessere Approximationen sind leider nicht möglich

NEGATIVE ERGEBNISSE (FALLS $\mathcal{P} \neq \mathcal{NP}$)

- **Knapsack: konstanter additiver Fehler unmöglich**
 - Für kein k gibt es einen polynomiellen Algorithmus A_{KP} mit der Eigenschaft $|OPT_{KP}(x) - W(x, A_{KP}(x))| \leq k$ für alle x

NEGATIVE ERGEBNISSE (FALLS $\mathcal{P} \neq \mathcal{NP}$)

- **Knapsack: konstanter additiver Fehler unmöglich**
 - Für kein k gibt es einen polynomiellen Algorithmus A_{KP} mit der Eigenschaft $|OPT_{KP}(x) - W(x, A_{KP}(x))| \leq k$ für alle x
- **Binpacking: absolute Güte $R_A < 3/2$ unmöglich**
 - Es gibt keinen polynomiellen Algorithmus A_{BPP} mit der Eigenschaft $R_{A_{BPP}}(x) < 3/2$ für alle x

NEGATIVE ERGEBNISSE (FALLS $\mathcal{P} \neq \mathcal{NP}$)

- **Knapsack: konstanter additiver Fehler unmöglich**
 - Für kein k gibt es einen polynomiellen Algorithmus A_{KP} mit der Eigenschaft $|OPT_{KP}(x) - W(x, A_{KP}(x))| \leq k$ für alle x
- **Binpacking: absolute Güte $R_A < 3/2$ unmöglich**
 - Es gibt keinen polynomiellen Algorithmus A_{BPP} mit der Eigenschaft $R_{A_{BPP}}(x) < 3/2$ für alle x
- **CLIQUE: Keine endliche absolute Güte möglich**
 - Es gibt ein $\epsilon > 0$, so daß es keinen polynomiellen Algorithmus A_{CL} geben kann mit $R_{A_{CL}}(x) < |x|^{1/2-\epsilon}$ für alle x (ohne Beweis)

NEGATIVE ERGEBNISSE (FALLS $\mathcal{P} \neq \mathcal{NP}$)

- **Knapsack: konstanter additiver Fehler unmöglich**
 - Für kein k gibt es einen polynomiellen Algorithmus A_{KP} mit der Eigenschaft $|OPT_{KP}(x) - W(x, A_{KP}(x))| \leq k$ für alle x
- **Binpacking: absolute Güte $R_A < 3/2$ unmöglich**
 - Es gibt keinen polynomiellen Algorithmus A_{BPP} mit der Eigenschaft $R_{A_{BPP}}(x) < 3/2$ für alle x
- **CLIQUE: Keine endliche absolute Güte möglich**
 - Es gibt ein $\epsilon > 0$, so daß es keinen polynomiellen Algorithmus A_{CL} geben kann mit $R_{A_{CL}}(x) < |x|^{1/2-\epsilon}$ für alle x (ohne Beweis)
- **TSP: Keine endliche worst-case Güte möglich**
 - Es gibt keinen polynomiellen Algorithmus A_{TSP} mit $R_{A_{TSP}}^\infty < \infty$

NEGATIVE ERGEBNISSE (FALLS $\mathcal{P} \neq \mathcal{NP}$)

- **Knapsack: konstanter additiver Fehler unmöglich**
 - Für kein k gibt es einen polynomiellen Algorithmus A_{KP} mit der Eigenschaft $|OPT_{KP}(x) - W(x, A_{KP}(x))| \leq k$ für alle x
- **Binpacking: absolute Güte $R_A < 3/2$ unmöglich**
 - Es gibt keinen polynomiellen Algorithmus A_{BPP} mit der Eigenschaft $R_{A_{BPP}}(x) < 3/2$ für alle x
- **CLIQUE: Keine endliche absolute Güte möglich**
 - Es gibt ein $\epsilon > 0$, so daß es keinen polynomiellen Algorithmus A_{CL} geben kann mit $R_{A_{CL}}(x) < |x|^{1/2-\epsilon}$ für alle x (ohne Beweis)
- **TSP: Keine endliche worst-case Güte möglich**
 - Es gibt keinen polynomiellen Algorithmus A_{TSP} mit $R_{A_{TSP}}^\infty < \infty$

Nachweise verwenden verschiedene Beweistechniken

KEIN KONSTANTER ADDITIVER FEHLER FÜR KNAPSACK

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

Für kein k gibt es einen polynomiellen Algorithmus A_{KP} mit der Eigenschaft $|OPT_{KP}(x) - W(x, A_{KP}(x))| \leq k$ für alle x

KEIN KONSTANTER ADDITIVER FEHLER FÜR KNAPSACK

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

Für kein k gibt es einen polynomiellen Algorithmus A_{KP} mit der Eigenschaft $|OPT_{KP}(x) - W(x, A_{KP}(x))| \leq k$ für alle x

Gäbe es A_{KP} , dann könnten wir KP wie folgt polynomiell entscheiden

KEIN KONSTANTER ADDITIVER FEHLER FÜR KNAPSACK

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

Für kein k gibt es einen polynomiellen Algorithmus A_{KP} mit der Eigenschaft $|OPT_{KP}(x) - W(x, A_{KP}(x))| \leq k$ für alle x

Gäbe es A_{KP} , dann könnten wir KP wie folgt polynomiell entscheiden

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

$$x' = (g_1..g_n, a_1*(k+1)..a_n*(k+1), G, A*(k+1))$$

KEIN KONSTANTER ADDITIVER FEHLER FÜR KNAPSACK

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

Für kein k gibt es einen polynomiellen Algorithmus A_{KP} mit der Eigenschaft $|OPT_{KP}(x) - W(x, A_{KP}(x))| \leq k$ für alle x

Gäbe es A_{KP} , dann könnten wir KP wie folgt polynomiell entscheiden

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

$$x' = (g_1..g_n, a_1 * (k+1) .. a_n * (k+1), G, A * (k+1))$$

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

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

KEIN KONSTANTER ADDITIVER FEHLER FÜR KNAPSACK

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

Für kein k gibt es einen polynomiellen Algorithmus A_{KP} mit der Eigenschaft $|OPT_{KP}(x) - W(x, A_{KP}(x))| \leq k$ für alle x

Gäbe es A_{KP} , dann könnten wir KP wie folgt polynomiell entscheiden

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

$$x' = (g_1..g_n, a_1*(k+1)..a_n*(k+1), G, A*(k+1))$$

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

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

– Also gilt $x \in KP \Leftrightarrow OPT_{KP}(x) = A$

$$\Leftrightarrow \lfloor W(x', A_{KP}(x')) / (k+1) \rfloor \geq A$$

KEIN KONSTANTER ADDITIVER FEHLER FÜR KNAPSACK

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

Für kein k gibt es einen polynomiellen Algorithmus A_{KP} mit der Eigenschaft $|OPT_{KP}(x) - W(x, A_{KP}(x))| \leq k$ für alle x

Gäbe es A_{KP} , dann könnten wir KP wie folgt polynomiell entscheiden

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

$$x' = (g_1..g_n, a_1 * (k+1) .. a_n * (k+1), G, A * (k+1))$$

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

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

– Also gilt $x \in KP \Leftrightarrow OPT_{KP}(x) = A$

$$\Leftrightarrow \lfloor W(x', A_{KP}(x')) / (k+1) \rfloor \geq A$$

Beweistechnik: Multiplikation des Problems, nachträgliche Division des Fehlers

KEINE ABSOLUTE GÜTE $R_A < 3/2$ FÜR BINPACKING

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

Es gibt keinen polynomiellen Algorithmus A_{BPP} mit der Eigenschaft $R_{A_{BPP}}(x) < 3/2$ für alle x

KEINE ABSOLUTE GÜTE $R_A < 3/2$ FÜR BINPACKING

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

Es gibt keinen polynomiellen Algorithmus A_{BPP} mit der Eigenschaft $R_{A_{BPP}}(x) < 3/2$ für alle x

- Die Reduktion $PART \leq_p BPP$ benötigt nur $k=2$ Behälter der Größe $b := \sum_{i=1}^n a_i/2$, um im Erfolgsfall alle Objekte a_i aufzuteilen

KEINE ABSOLUTE GÜTE $R_A < 3/2$ FÜR BINPACKING

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

Es gibt keinen polynomiellen Algorithmus A_{BPP} mit der Eigenschaft $R_{A_{BPP}}(x) < 3/2$ für alle x

- Die Reduktion $PART \leq_p BPP$ benötigt nur $k=2$ Behälter der Größe $b := \sum_{i=1}^n a_i / 2$, um im Erfolgsfall alle Objekte a_i aufzuteilen
- Für die Transformationsfunktion f gilt also
$$x \in PART \Leftrightarrow OPT_{BPP}(f(x)) = 2$$

KEINE ABSOLUTE GÜTE $R_A < 3/2$ FÜR BINPACKING

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

Es gibt keinen polynomiellen Algorithmus A_{BPP} mit der Eigenschaft $R_{A_{BPP}}(x) < 3/2$ für alle x

– Die Reduktion $PART \leq_p BPP$ benötigt nur $k=2$ Behälter der Größe $b := \sum_{i=1}^n a_i / 2$, um im Erfolgsfall alle Objekte a_i aufzuteilen

– Für die Transformationsfunktion f gilt also

$$x \in PART \Leftrightarrow OPT_{BPP}(f(x)) = 2$$

– Jeder Approximationsalgorithmus A mit $R_A(x) < 3/2$ liefert damit einen Entscheidungsalgorithmus für das Partitionsproblem, denn

$$\begin{aligned} W(f(x), A(f(x))) &= \lfloor 2 * R_A(x) \rfloor = 2, & \text{falls } x \in PART \\ W(f(x), A(f(x))) &\geq 3 & \text{sonst} \end{aligned}$$

KEINE ABSOLUTE GÜTE $R_A < 3/2$ FÜR BINPACKING

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

Es gibt keinen polynomiellen Algorithmus A_{BPP} mit der Eigenschaft $R_{A_{BPP}}(x) < 3/2$ für alle x

– Die Reduktion $PART \leq_p BPP$ benötigt nur $k=2$ Behälter der Größe $b := \sum_{i=1}^n a_i / 2$, um im Erfolgsfall alle Objekte a_i aufzuteilen

– Für die Transformationsfunktion f gilt also

$$x \in PART \Leftrightarrow OPT_{BPP}(f(x)) = 2$$

– Jeder Approximationsalgorithmus A mit $R_A(x) < 3/2$ liefert damit einen Entscheidungsalgorithmus für das Partitionsproblem, denn

$$W(f(x), A(f(x))) = \lfloor 2 * R_A(x) \rfloor = 2, \quad \text{falls } x \in PART$$

$$W(f(x), A(f(x))) \geq 3 \quad \text{sonst}$$

– Wegen $PART \in \mathcal{NP}$ kann A nicht polynomiell sein

KEINE ABSOLUTE GÜTE $R_A < 3/2$ FÜR BINPACKING

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

Es gibt keinen polynomiellen Algorithmus A_{BPP} mit der Eigenschaft $R_{A_{BPP}}(x) < 3/2$ für alle x

– Die Reduktion $PART \leq_p BPP$ benötigt nur $k=2$ Behälter der Größe $b := \sum_{i=1}^n a_i/2$, um im Erfolgsfall alle Objekte a_i aufzuteilen

– Für die Transformationsfunktion f gilt also

$$x \in PART \Leftrightarrow OPT_{BPP}(f(x)) = 2$$

– Jeder Approximationsalgorithmus A mit $R_A(x) < 3/2$ liefert damit einen Entscheidungsalgorithmus für das Partitionsproblem, denn

$$W(f(x), A(f(x))) = \lfloor 2 * R_A(x) \rfloor = 2, \text{ falls } x \in PART$$

$$W(f(x), A(f(x))) \geq 3 \quad \text{sonst}$$

– Wegen $PART \in \mathcal{NP}$ kann A nicht polynomiell sein

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

KEINE ENDLICHE WORST-CASE GÜTE FÜR TSP

$$TSP = \{ c_{12}, \dots, c_{n-1,n}, B \mid \exists \text{ Bijektion } \pi. \sum_{i=1}^{n-1} c_{\pi(i)\pi(i+1)} + c_{\pi(n)\pi(1)} \leq B \}$$

Es gibt keinen polynomiellen Algorithmus A_{TSP} mit der Eigenschaft

$$R_{A_{TSP}}^{\infty} = r \text{ für ein } r \in \mathbb{N}$$

KEINE ENDLICHE WORST-CASE GÜTE FÜR TSP

$$TSP = \{ c_{12}, \dots, c_{n-1,n}, B \mid \exists \text{ Bijektion } \pi. \sum_{i=1}^{n-1} c_{\pi(i)\pi(i+1)} + c_{\pi(n)\pi(1)} \leq B \}$$

Es gibt keinen polynomiellen Algorithmus A_{TSP} mit der Eigenschaft

$$R_{A_{TSP}}^{\infty} = r \text{ für ein } r \in \mathbb{N}$$

Die Reduktion $HC \leq_p TSP$ stellt Kanten $\{v_i, v_j\}$ durch Kosten $c_{ij} = 1$ und Nichtkanten durch höhere Kosten dar. Mit einem Algorithmus A_{TSP} mit $R_{A_{TSP}}^{\infty} = r$ könnte man HC polynomiell wie folgt entscheiden

KEINE ENDLICHE WORST-CASE GÜTE FÜR TSP

$$TSP = \{ c_{12}, \dots, c_{n-1,n}, B \mid \exists \text{ Bijektion } \pi. \sum_{i=1}^{n-1} c_{\pi(i)\pi(i+1)} + c_{\pi(n)\pi(1)} \leq B \}$$

Es gibt keinen polynomiellen Algorithmus A_{TSP} mit der Eigenschaft $R_{A_{TSP}}^\infty = r$ für ein $r \in \mathbb{N}$

Die Reduktion $HC \leq_p TSP$ stellt Kanten $\{v_i, v_j\}$ durch Kosten $c_{ij} = 1$ und Nichtkanten durch höhere Kosten dar. Mit einem Algorithmus A_{TSP} mit $R_{A_{TSP}}^\infty = r$ könnte man HC polynomiell wie folgt entscheiden

- Transformiere $G = (V, E)$ in das TSP $x = c_{12}, \dots, c_{n-1,n}, |V|$ mit $c_{ij} = 1$, falls $\{v_i, v_j\} \in E$ und $c_{ij} = r|V| + 2$, sonst

KEINE ENDLICHE WORST-CASE GÜTE FÜR TSP

$$TSP = \{ c_{12}, \dots, c_{n-1,n}, B \mid \exists \text{ Bijektion } \pi. \sum_{i=1}^{n-1} c_{\pi(i)\pi(i+1)} + c_{\pi(n)\pi(1)} \leq B \}$$

Es gibt keinen polynomiellen Algorithmus A_{TSP} mit der Eigenschaft $R_{A_{TSP}}^\infty = r$ für ein $r \in \mathbb{N}$

Die Reduktion $HC \leq_p TSP$ stellt Kanten $\{v_i, v_j\}$ durch Kosten $c_{ij} = 1$ und Nichtkanten durch höhere Kosten dar. Mit einem Algorithmus A_{TSP} mit $R_{A_{TSP}}^\infty = r$ könnte man HC polynomiell wie folgt entscheiden

- Transformiere $G = (V, E)$ in das TSP $x = c_{12}, \dots, c_{n-1,n}, |V|$
mit $c_{ij} = 1$, falls $\{v_i, v_j\} \in E$ und $c_{ij} = r|V| + 2$, sonst
- Dann gilt $G \in HC \Rightarrow OPT_{TSP}(x) = |V|$
 $G \notin HC \Rightarrow OPT_{TSP}(x) \geq r|V| + 2 + (|V| - 1) > (r+1) * |V|$

KEINE ENDLICHE WORST-CASE GÜTE FÜR TSP

$$TSP = \{ c_{12}, \dots, c_{n-1,n}, B \mid \exists \text{ Bijektion } \pi. \sum_{i=1}^{n-1} c_{\pi(i)\pi(i+1)} + c_{\pi(n)\pi(1)} \leq B \}$$

Es gibt keinen polynomiellen Algorithmus A_{TSP} mit der Eigenschaft $R_{A_{TSP}}^\infty = r$ für ein $r \in \mathbb{N}$

Die Reduktion $HC \leq_p TSP$ stellt Kanten $\{v_i, v_j\}$ durch Kosten $c_{ij} = 1$ und Nichtkanten durch höhere Kosten dar. Mit einem Algorithmus A_{TSP} mit $R_{A_{TSP}}^\infty = r$ könnte man HC polynomiell wie folgt entscheiden

- Transformiere $G = (V, E)$ in das TSP $x = c_{12}, \dots, c_{n-1,n}, |V|$ mit $c_{ij} = 1$, falls $\{v_i, v_j\} \in E$ und $c_{ij} = r|V| + 2$, sonst
- Dann gilt $G \in HC \Rightarrow OPT_{TSP}(x) = |V|$
 $G \notin HC \Rightarrow OPT_{TSP}(x) \geq r|V| + 2 + (|V| - 1) > (r+1)*|V|$
- Für große Graphen gilt aber: $W(x, A(x)) \leq r*OPT_{TSP}(x)$
also $G \in HC \Leftrightarrow W(x, A(x)) \leq r*|V|$

Für kleine Graphen ist die Laufzeit des Entscheidungsalgorithmus irrelevant

KEINE ENDLICHE WORST-CASE GÜTE FÜR TSP

$$TSP = \{ c_{12}, \dots, c_{n-1,n}, B \mid \exists \text{ Bijektion } \pi. \sum_{i=1}^{n-1} c_{\pi(i)\pi(i+1)} + c_{\pi(n)\pi(1)} \leq B \}$$

Es gibt keinen polynomiellen Algorithmus A_{TSP} mit der Eigenschaft $R_{A_{TSP}}^\infty = r$ für ein $r \in \mathbb{N}$

Die Reduktion $HC \leq_p TSP$ stellt Kanten $\{v_i, v_j\}$ durch Kosten $c_{ij} = 1$ und Nichtkanten durch höhere Kosten dar. Mit einem Algorithmus A_{TSP} mit $R_{A_{TSP}}^\infty = r$ könnte man HC polynomiell wie folgt entscheiden

- Transformiere $G = (V, E)$ in das TSP $x = c_{12}, \dots, c_{n-1,n}, |V|$ mit $c_{ij} = 1$, falls $\{v_i, v_j\} \in E$ und $c_{ij} = r|V| + 2$, sonst
- Dann gilt $G \in HC \Rightarrow OPT_{TSP}(x) = |V|$
 $G \notin HC \Rightarrow OPT_{TSP}(x) \geq r|V| + 2 + (|V| - 1) > (r+1)*|V|$
- Für große Graphen gilt aber: $W(x, A(x)) \leq r*OPT_{TSP}(x)$
also $G \in HC \Leftrightarrow W(x, A(x)) \leq r*|V|$

Für kleine Graphen ist die Laufzeit des Entscheidungsalgorithmus irrelevant

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

PROBABILISTISCHE ALGORITHMEN

“Approximation” einer Entscheidung

“Approximation” einer Entscheidung

- **Verhalten gesteuert durch Zufallszahlgenerator**
 - Falsche Entscheidungen sind möglich aber unwahrscheinlich
 - Approximation $\hat{=}$ Verringerung der Fehlerwahrscheinlichkeit
 - Fehler unter 2^{-100} liegt unter Wahrscheinlichkeit von Hardwarefehlern

“Approximation” einer Entscheidung

- **Verhalten gesteuert durch Zufallszahlgenerator**
 - Falsche Entscheidungen sind möglich aber unwahrscheinlich
 - Approximation $\hat{=}$ Verringerung der Fehlerwahrscheinlichkeit
 - Fehler unter 2^{-100} liegt unter Wahrscheinlichkeit von Hardwarefehlern
- **Viele sinnvolle Anwendungen**
 - **Quicksort**: schnellstes Sortierverfahren in der Praxis
 - Linearer **Primzahltest** (relativ zur Anzahl der Bits)

“Approximation” einer Entscheidung

- **Verhalten gesteuert durch Zufallszahlgenerator**
 - Falsche Entscheidungen sind möglich aber unwahrscheinlich
 - Approximation $\hat{=}$ Verringerung der Fehlerwahrscheinlichkeit
 - Fehler unter 2^{-100} liegt unter Wahrscheinlichkeit von Hardwarefehlern
- **Viele sinnvolle Anwendungen**
 - **Quicksort**: schnellstes Sortierverfahren in der Praxis
 - Linearer **Primzahltest** (relativ zur Anzahl der Bits)
- **Wie weist man gute Eigenschaften nach?**
 - **Einfaches Modell** für probabilistische Algorithmen formulieren
 - Eigenschaften abstrakter **probabilistischer Sprachklassen** analysieren

QUICKSORT ALS ZUFALLSABHÄNGIGER ALGORITHMUS

- **Divide & Conquer Ansatz für Sortierung**
 - Wähle **Pivotelement** a_i aus Liste a_1, \dots, a_n
 - Zerlege Liste in Elemente, die größer oder kleiner als a_i sind
 - Sortiere Teillisten und hänge Ergebnisse aneinander

QUICKSORT ALS ZUFALLSABHÄNGIGER ALGORITHMUS

- **Divide & Conquer Ansatz für Sortierung**
 - Wähle **Pivotelement** a_i aus Liste a_1, \dots, a_n
 - Zerlege Liste in Elemente, die größer oder kleiner als a_i sind
 - Sortiere Teillisten und hänge Ergebnisse aneinander
- **Laufzeit abhängig vom Pivotelement**
 - $\mathcal{O}(n * \log_2 n)$, wenn Teillisten in etwa gleich groß sind
 - Pivotelement muß nahe am Mittelwert sein

QUICKSORT ALS ZUFALLSABHÄNGIGER ALGORITHMUS

- **Divide & Conquer Ansatz für Sortierung**
 - Wähle **Pivotelement** a_i aus Liste a_1, \dots, a_n
 - Zerlege Liste in Elemente, die größer oder kleiner als a_i sind
 - Sortiere Teillisten und hänge Ergebnisse aneinander
- **Laufzeit abhängig vom Pivotelement**
 - $\mathcal{O}(n * \log_2 n)$, wenn Teillisten in etwa gleich groß sind
 - Pivotelement muß nahe am Mittelwert sein
 - **Deterministische Bestimmung des Mittelwertes zu zeitaufwendig**
 - Wahl eines festen Pivotelements erhöht Laufzeit von Quicksort für bestimmte Eingabelisten auf $\mathcal{O}(n^2)$

QUICKSORT ALS ZUFALLSABHÄNGIGER ALGORITHMUS

- **Divide & Conquer Ansatz für Sortierung**

- Wähle **Pivotelement** a_i aus Liste a_1, \dots, a_n
- Zerlege Liste in Elemente, die größer oder kleiner als a_i sind
- Sortiere Teillisten und hänge Ergebnisse aneinander

- **Laufzeit abhängig vom Pivotelement**

- $\mathcal{O}(n * \log_2 n)$, wenn Teillisten in etwa gleich groß sind
- Pivotelement muß nahe am Mittelwert sein
- **Deterministische Bestimmung des Mittelwertes zu zeitaufwendig**
- Wahl eines festen Pivotelements erhöht Laufzeit von Quicksort für bestimmte Eingabelisten auf $\mathcal{O}(n^2)$

- **Gute Pivotelemente sind in der Mehrzahl**

- Zufällige Wahl führt **für jede Eingabe zum Erwartungswert** $\mathcal{O}(n * \log_2 n)$
- Im Durchschnitt schneller als deterministische $\mathcal{O}(n * \log_2 n)$ -Verfahren

- **Probabilistische Turingmaschine**

- Struktur: $M = (Q, \Sigma, \Gamma, \delta, q_0, B, F)$

- Zustandsüberföhrungsfunktion: $\delta : Q \times \Gamma \rightarrow (Q \times \Gamma \times \{L, R\})^2$

- Jede Alternative wird mit **Wahrscheinlichkeit 1/2** gewöhlt

- **Rechenzeit**: maximale Rechenzeit aller möglichen Rechenwege

- **PTM**: polynomiell zeitbeschränkte probabilistische Turingmaschine

- **Probabilistische Turingmaschine**

- Struktur: $M = (Q, \Sigma, \Gamma, \delta, q_0, B, F)$

- Zustandsüberföhrungsfunktion: $\delta : Q \times \Gamma \rightarrow (Q \times \Gamma \times \{L, R\})^2$

- Jede Alternative wird mit **Wahrscheinlichkeit** $1/2$ gewöhlt

- **Rechenzeit**: maximale Rechenzeit aller möglichen Rechenwege

- **PTM**: polynomiell zeitbeschränkte probabilistische Turingmaschine

- **$Prob(M \downarrow w)$: Wahrscheinlichkeit der Akzeptanz**

- Wahrscheinlichkeit aller akzeptierenden Konfigurationsfolgen relativ zu allen möglichen Konfigurationsfolgen bei Eingabe w

- **Probabilistische Turingmaschine**

- Struktur: $M = (Q, \Sigma, \Gamma, \delta, q_0, B, F)$

- Zustandsüberföhrungsfunktion: $\delta : Q \times \Gamma \rightarrow (Q \times \Gamma \times \{L, R\})^2$

- Jede Alternative wird mit **Wahrscheinlichkeit 1/2** gewöhlt

- **Rechenzeit**: maximale Rechenzeit aller möglichen Rechenwege

- **PTM**: polynomiell zeitbeschränkte probabilistische Turingmaschine

- **$Prob(M \downarrow w)$: Wahrscheinlichkeit der Akzeptanz**

- Wahrscheinlichkeit aller akzeptierenden Konfigurationsfolgen relativ zu allen möglichen Konfigurationsfolgen bei Eingabe w

- **Probabilistische Algorithmen**

- Abstrakteres Modell: Programme mit zufälligen Entscheidungen

- Komplexitätsbestimmung durch asymptotische Analyse wie bisher

- **Probabilistische Turingmaschine**

- Struktur: $M = (Q, \Sigma, \Gamma, \delta, q_0, B, F)$

- Zustandsüberföhrungsfunktion: $\delta : Q \times \Gamma \rightarrow (Q \times \Gamma \times \{L, R\})^2$

- Jede Alternative wird mit **Wahrscheinlichkeit 1/2** gewöhlt

- **Rechenzeit**: maximale Rechenzeit aller möglichen Rechenwege

- **PTM**: polynomiell zeitbeschränkte probabilistische Turingmaschine

- **$Prob(M \downarrow w)$: Wahrscheinlichkeit der Akzeptanz**

- Wahrscheinlichkeit aller akzeptierenden Konfigurationsfolgen relativ zu allen möglichen Konfigurationsfolgen bei Eingabe w

- **Probabilistische Algorithmen**

- Abstrakteres Modell: Programme mit zufälligen Entscheidungen

- Komplexitätsbestimmung durch asymptotische Analyse wie bisher

Was kann man mit polynomiell zeitbeschränkten probabilistischen Algorithmen erreichen?

- ***PP*: Probabilistic Polynomial** Monte-Carlo-Algorithmen
 - Wahrscheinlichkeit für korrekte Antwort ist mindestens $1/2$
 $w \in L \Rightarrow \text{Prob}(M \downarrow w) \geq 1/2$ und $w \notin L \Rightarrow \text{Prob}(M \not\downarrow w) > 1/2$
 - Laufzeit polynomiell

- ***PP*: Probabilistic Polynomial** Monte-Carlo-Algorithmen
 - Wahrscheinlichkeit für korrekte Antwort ist mindestens $1/2$
 $w \in L \Rightarrow \text{Prob}(M \downarrow w) \geq 1/2$ und $w \notin L \Rightarrow \text{Prob}(M \not\downarrow w) > 1/2$
 - Laufzeit polynomiell
- ***BPP*: Bounded error Probabilistic Polynomial**
 - Wahrscheinlichkeit für korrekte Antwort ist mindestens $1/2 + \epsilon$
 - Laufzeit polynomiell

- ***PP*: Probabilistic Polynomial** Monte-Carlo-Algorithmen
 - Wahrscheinlichkeit für korrekte Antwort ist mindestens $1/2$
 $w \in L \Rightarrow \text{Prob}(M \downarrow w) \geq 1/2$ und $w \notin L \Rightarrow \text{Prob}(M \not\downarrow w) > 1/2$
 - Laufzeit polynomiell
- ***BPP*: Bounded error Probabilistic Polynomial**
 - Wahrscheinlichkeit für korrekte Antwort ist mindestens $1/2 + \epsilon$
 - Laufzeit polynomiell
- ***RP*: Random Polynomial**
 - Wahrscheinlichkeit für Akzeptanz von $w \in L$ ist mindestens $1/2$
 - Eingaben $w \notin L$ werden mit Sicherheit nicht akzeptiert
 - Laufzeit polynomiell

- **PP: Probabilistic Polynomial** Monte-Carlo-Algorithmen
 - Wahrscheinlichkeit für korrekte Antwort ist mindestens $1/2$
 $w \in L \Rightarrow \text{Prob}(M \downarrow w) \geq 1/2$ und $w \notin L \Rightarrow \text{Prob}(M \not\downarrow w) > 1/2$
 - Laufzeit polynomiell
- **BPP: Bounded error Probabilistic Polynomial**
 - Wahrscheinlichkeit für korrekte Antwort ist mindestens $1/2 + \epsilon$
 - Laufzeit polynomiell
- **RP: Random Polynomial**
 - Wahrscheinlichkeit für Akzeptanz von $w \in L$ ist mindestens $1/2$
 - Eingaben $w \notin L$ werden mit Sicherheit nicht akzeptiert
 - Laufzeit polynomiell
- **ZPP: Zero error PP** Las-Vegas-Algorithmen
 - M gibt immer eine korrekte Antwort
 - Erwartungswert der Laufzeit ist polynomiell
 - Alternative Definitionen verwenden Erkenntnis $ZPP = RP \cap co-RP$ (Folie 26)

BEISPIEL PROBABILISTISCHER ALGORITHMEN

Prüfe ob ein Graph $G=(V, E)$ eine 3-Clique enthält

Prüfe ob ein Graph $G=(V, E)$ eine 3-Clique enthält

- **Teste zufällige Dreieckskandidaten**

- Zufällige Auswahl von $v_1 \in V$ und $\{v_2, v_3\} \in E$ mit $v_1 \notin \{v_2, v_3\}$
- Teste deterministisch, ob $\{v_1, v_3\} \in E$ und $\{v_1, v_2\} \in E$ gilt
- Akzeptiere, wenn Dreieck in k Iterationen gefunden, sonst verwerfe

Prüfe ob ein Graph $G=(V, E)$ eine 3-Clique enthält

- **Teste zufällige Dreieckskandidaten**

- Zufällige Auswahl von $v_1 \in V$ und $\{v_2, v_3\} \in E$ mit $v_1 \notin \{v_2, v_3\}$
- Teste deterministisch, ob $\{v_1, v_3\} \in E$ und $\{v_1, v_2\} \in E$ gilt
- Akzeptiere, wenn Dreieck in k Iterationen gefunden, sonst verwirfe

- **Wahrscheinlichkeit korrekter Akzeptanz $> 1/2$**

- Wahrscheinlichkeit für Auswahl einer Dreieckskante $> \frac{3}{|E|}$
- Wahrscheinlichkeit für Auswahl des dritten Knotens $> \frac{1}{|V|-2}$
- Akzeptanzwahrscheinlichkeit $> 1 - \left(1 - \frac{3}{|E| \cdot (|V|-2)}\right)^k \doteq 1 - e^{-k \frac{3}{|E| \cdot (|V|-2)}}$

Prüfe ob ein Graph $G=(V, E)$ eine 3-Clique enthält

- **Teste zufällige Dreieckskandidaten**

- Zufällige Auswahl von $v_1 \in V$ und $\{v_2, v_3\} \in E$ mit $v_1 \notin \{v_2, v_3\}$
- Teste deterministisch, ob $\{v_1, v_3\} \in E$ und $\{v_1, v_2\} \in E$ gilt
- Akzeptiere, wenn Dreieck in k Iterationen gefunden, sonst verwirfe

- **Wahrscheinlichkeit korrekter Akzeptanz $> 1/2$**

- Wahrscheinlichkeit für Auswahl einer Dreieckskante $> \frac{3}{|E|}$
- Wahrscheinlichkeit für Auswahl des dritten Knotens $> \frac{1}{|V|-2}$
- Akzeptanzwahrscheinlichkeit $> 1 - \left(1 - \frac{3}{|E| \cdot (|V|-2)}\right)^k \doteq 1 - e^{-k \frac{3}{|E| \cdot (|V|-2)}}$
- $k := \frac{|E| \cdot (|V|-2)}{3}$ erhöht Akzeptanzwahrscheinlichkeit auf mehr als $1/2$

Prüfe ob ein Graph $G=(V, E)$ eine 3-Clique enthält

- **Teste zufällige Dreieckskandidaten**

- Zufällige Auswahl von $v_1 \in V$ und $\{v_2, v_3\} \in E$ mit $v_1 \notin \{v_2, v_3\}$
- Teste deterministisch, ob $\{v_1, v_3\} \in E$ und $\{v_1, v_2\} \in E$ gilt
- Akzeptiere, wenn Dreieck in k Iterationen gefunden, sonst verwirfe

- **Wahrscheinlichkeit korrekter Akzeptanz $> 1/2$**

- Wahrscheinlichkeit für Auswahl einer Dreieckskante $> \frac{3}{|E|}$
- Wahrscheinlichkeit für Auswahl des dritten Knotens $> \frac{1}{|V|-2}$
- Akzeptanzwahrscheinlichkeit $> 1 - \left(1 - \frac{3}{|E| \cdot (|V|-2)}\right)^k \doteq 1 - e^{-k \frac{3}{|E| \cdot (|V|-2)}}$
- $k := \frac{|E| \cdot (|V|-2)}{3}$ erhöht Akzeptanzwahrscheinlichkeit auf mehr als $1/2$

- **Keine falschen Positive**

- Es wird nur akzeptiert, wenn wirklich ein Dreieck gefunden wird

Prüfe ob ein Graph $G=(V, E)$ eine 3-Clique enthält

- **Teste zufällige Dreieckskandidaten**

- Zufällige Auswahl von $v_1 \in V$ und $\{v_2, v_3\} \in E$ mit $v_1 \notin \{v_2, v_3\}$
- Teste deterministisch, ob $\{v_1, v_3\} \in E$ und $\{v_1, v_2\} \in E$ gilt
- Akzeptiere, wenn Dreieck in k Iterationen gefunden, sonst verwirfe

- **Wahrscheinlichkeit korrekter Akzeptanz $> 1/2$**

- Wahrscheinlichkeit für Auswahl einer Dreieckskante $> \frac{3}{|E|}$
- Wahrscheinlichkeit für Auswahl des dritten Knotens $> \frac{1}{|V|-2}$
- Akzeptanzwahrscheinlichkeit $> 1 - \left(1 - \frac{3}{|E| \cdot (|V|-2)}\right)^k \doteq 1 - e^{-k \frac{3}{|E| \cdot (|V|-2)}}$
- $k := \frac{|E| \cdot (|V|-2)}{3}$ erhöht Akzeptanzwahrscheinlichkeit auf mehr als $1/2$

- **Keine falschen Positive**

- Es wird nur akzeptiert, wenn wirklich ein Dreieck gefunden wird

- **Laufzeit polynomiell**

- Individueller Test ist linear in $|G|$, Anzahl der Iterationen quadratisch

Prüfe ob ein Graph $G=(V, E)$ eine 3-Clique enthält

- **Teste zufällige Dreieckskandidaten**

- Zufällige Auswahl von $v_1 \in V$ und $\{v_2, v_3\} \in E$ mit $v_1 \notin \{v_2, v_3\}$
- Teste deterministisch, ob $\{v_1, v_3\} \in E$ und $\{v_1, v_2\} \in E$ gilt
- Akzeptiere, wenn Dreieck in k Iterationen gefunden, sonst verwirfe

- **Wahrscheinlichkeit korrekter Akzeptanz $> 1/2$**

- Wahrscheinlichkeit für Auswahl einer Dreieckskante $> \frac{3}{|E|}$
- Wahrscheinlichkeit für Auswahl des dritten Knotens $> \frac{1}{|V|-2}$
- Akzeptanzwahrscheinlichkeit $> 1 - \left(1 - \frac{3}{|E| \cdot (|V|-2)}\right)^k \doteq 1 - e^{-k \frac{3}{|E| \cdot (|V|-2)}}$
- $k := \frac{|E| \cdot (|V|-2)}{3}$ erhöht Akzeptanzwahrscheinlichkeit auf mehr als $1/2$

- **Keine falschen Positive**

- Es wird nur akzeptiert, wenn wirklich ein Dreieck gefunden wird

- **Laufzeit polynomiell (*RP* Algorithmus)**

- Individueller Test ist linear in $|G|$, Anzahl der Iterationen quadratisch

ITERATION PROBABILISTISCHER ALGORITHMEN

Iteration kann Fehler extrem klein machen

ITERATION PROBABILISTISCHER ALGORITHMEN

Iteration kann Fehler extrem klein machen

- k -fache Iteration von RP Algorithmen reduziert Fehlerwahrscheinlichkeit auf 2^{-k}

Iteration kann Fehler extrem klein machen

- **k -fache Iteration von RP Algorithmen reduziert Fehlerwahrscheinlichkeit auf 2^{-k}**
 - Ist M die k -fache **stochastisch unabhängige** Iteration einer PTM M_L für $L \in RP$, so gilt $w \in L \Rightarrow Prob(M \downarrow w) \geq 1 - 2^{-k}$
und $w \notin L \Rightarrow Prob(M \not\downarrow w) = 1$
 - Einfaches wahrscheinlichkeitstheoretisches Argument

Iteration kann Fehler extrem klein machen

- **k -fache Iteration von RP Algorithmen reduziert Fehlerwahrscheinlichkeit auf 2^{-k}**
 - Ist M die k -fache stochastisch unabhängige Iteration einer PTM M_L für $L \in RP$, so gilt $w \in L \Rightarrow Prob(M \downarrow w) \geq 1 - 2^{-k}$
und $w \notin L \Rightarrow Prob(M \uparrow w) = 2^{-k}$
 - Einfaches wahrscheinlichkeitstheoretisches Argument
- **$(2t+1)$ -fache Iteration eines BPP Algorithmus für $t > \frac{k}{-\log(1-4\epsilon^2)}$ liefert Fehlerwahrscheinlichkeit $< 2^{-k}$**

Iteration kann Fehler extrem klein machen

- **k -fache Iteration von RP Algorithmen reduziert Fehlerwahrscheinlichkeit auf 2^{-k}**
 - Ist M die k -fache stochastisch unabhängige Iteration einer PTM M_L für $L \in RP$, so gilt $w \in L \Rightarrow Prob(M \downarrow w) \geq 1 - 2^{-k}$
und $w \notin L \Rightarrow Prob(M \not\downarrow w) = 1$
 - Einfaches wahrscheinlichkeitstheoretisches Argument
- **$(2t+1)$ -fache Iteration eines BPP Algorithmus für $t > \frac{k}{-\log(1-4\epsilon^2)}$ liefert Fehlerwahrscheinlichkeit $< 2^{-k}$**
 - Sei M^t die $(2t+1)$ -fache stochastisch unabhängige Iteration einer PTM M für $L \in BPP$, die genau dann akzeptiert, wenn M mindestens $t+1$ -mal akzeptiert, so gilt für $t > \frac{k-1}{-\log(1-4\epsilon^2)}$
 $w \in L \Rightarrow Prob(M^t \downarrow w) > 1 - 2^{-k}$ und $w \notin L \Rightarrow Prob(M^t \not\downarrow w) > 1 - 2^{-k}$
 - Aufwendige Analyse (siehe Wegener 75–77 für Details)

Iteration kann Fehler extrem klein machen

- **k -fache Iteration von RP Algorithmen reduziert Fehlerwahrscheinlichkeit auf 2^{-k}**
 - Ist M die k -fache stochastisch unabhängige Iteration einer PTM M_L für $L \in RP$, so gilt $w \in L \Rightarrow Prob(M \downarrow w) \geq 1 - 2^{-k}$
und $w \notin L \Rightarrow Prob(M \not\downarrow w) = 1$
 - Einfaches wahrscheinlichkeitstheoretisches Argument
- **$(2t+1)$ -fache Iteration eines BPP Algorithmus für $t > \frac{k}{-\log(1-4\epsilon^2)}$ liefert Fehlerwahrscheinlichkeit $< 2^{-k}$**
 - Sei M^t die $(2t+1)$ -fache stochastisch unabhängige Iteration einer PTM M für $L \in BPP$, die genau dann akzeptiert, wenn M mindestens $t+1$ -mal akzeptiert, so gilt für $t > \frac{k-1}{-\log(1-4\epsilon^2)}$
 $w \in L \Rightarrow Prob(M^t \downarrow w) > 1 - 2^{-k}$ und $w \notin L \Rightarrow Prob(M^t \not\downarrow w) > 1 - 2^{-k}$
 - Aufwendige Analyse (siehe Wegener 75–77 für Details)
- **Keine Aussagen für PP Algorithmen möglich**

ANWENDUNGEN IN DER KRYPTOGRAPHIE

- **Public-Key Kryptographie mit dem RSA Verfahren**
 - Sichere spontane Verbindung zwischen beliebigen Teilnehmern
 - Ver- und Entschlüsselung benutzen verschiedene Schlüssel
 - Empfänger erzeugt beide Schlüssel, hält Entschlüsselung geheim
legt nur den Verschlüsselungsschlüssel offen
 - Ältestes und bedeutendstes Verfahren ist RSA Rivest, Shamir & Adleman, 1977
 - **Fester Bestandteil von allen modernen Internetbrowsern**

ANWENDUNGEN IN DER KRYPTOGRAPHIE

● Public-Key Kryptographie mit dem RSA Verfahren

- Sichere spontane Verbindung zwischen beliebigen Teilnehmern
- Ver- und Entschlüsselung benutzen verschiedene Schlüssel
- Empfänger erzeugt beide Schlüssel, hält Entschlüsselung geheim
legt nur den Verschlüsselungsschlüssel offen
- Ältestes und bedeutendstes Verfahren ist RSA Rivest, Shamir & Adleman, 1977
- Fester Bestandteil von allen modernen Internetbrowsern

● Schlüsselerzeugung

- Generiere n als Produkt zweier großer Primzahlen p und q
- Erzeuge e mit $\text{ggT}(e, (p-1)(q-1))=1$, berechne $d = e^{-1} \bmod (p-1)(q-1)$
- Mache n, e öffentlich, halte d, p und q geheim

ANWENDUNGEN IN DER KRYPTOGRAPHIE

● Public-Key Kryptographie mit dem RSA Verfahren

- Sichere spontane Verbindung zwischen beliebigen Teilnehmern
- Ver- und Entschlüsselung benutzen verschiedene Schlüssel
- Empfänger erzeugt beide Schlüssel, hält Entschlüsselung geheim
legt nur den Verschlüsselungsschlüssel offen
- Ältestes und bedeutendstes Verfahren ist RSA Rivest, Shamir & Adleman, 1977
- Fester Bestandteil von allen modernen Internetbrowsern

● Schlüsselerzeugung

- Generiere n als Produkt zweier großer Primzahlen p und q
- Erzeuge e mit $ggT(e, (p-1)(q-1))=1$, berechne $d = e^{-1} \bmod (p-1)(q-1)$
- Mache n, e öffentlich, halte d, p und q geheim

● Verschlüsselungsverfahren

- Zerlege Text in Blöcke der Länge $\log_2 n/8$ (ein Byte pro Buchstabe)
- Verschlüsselung wird Potenzieren mit e modulo n : $e_K(x) = x^e \bmod n$
- Entschlüsselung wird Potenzieren mit d modulo n : $d_K(y) = y^d \bmod n$

ANWENDUNGEN IN DER KRYPTOGRAPHIE

● Public-Key Kryptographie mit dem RSA Verfahren

- Sichere spontane Verbindung zwischen beliebigen Teilnehmern
- Ver- und Entschlüsselung benutzen verschiedene Schlüssel
- Empfänger erzeugt beide Schlüssel, hält Entschlüsselung geheim
legt nur den Verschlüsselungsschlüssel offen
- Ältestes und bedeutendstes Verfahren ist RSA Rivest, Shamir & Adleman, 1977
- Fester Bestandteil von allen modernen Internetbrowsern

● Schlüsselerzeugung

- Generiere n als Produkt zweier großer Primzahlen p und q
- Erzeuge e mit $ggT(e, (p-1)(q-1))=1$, berechne $d = e^{-1} \bmod (p-1)(q-1)$
- Mache n, e öffentlich, halte d, p und q geheim

● Verschlüsselungsverfahren

- Zerlege Text in Blöcke der Länge $\log_2 n/8$ (ein Byte pro Buchstabe)
- Verschlüsselung wird Potenzieren mit e modulo n : $e_K(x) = x^e \bmod n$
- Entschlüsselung wird Potenzieren mit d modulo n : $d_K(y) = y^d \bmod n$

Sicherheit basiert auf Zahlentheorie und Komplexität

WARUM FUNKTIONIERT RSA IN DER PRAXIS?

- **Korrektheit basiert auf Gesetzen der Zahlentheorie**

- Aus $n=p \cdot q$ und $\text{ggT}(x, n) = 1$ folgt $x^{(p-1)(q-1)} \bmod n = 1$ (Euler-Fermat)

- Aus $e \cdot d \bmod (p-1)(q-1) = 1$ und $x < n$ folgt $(x^e)^d \bmod n = x$

WARUM FUNKTIONIERT RSA IN DER PRAXIS?

- **Korrektheit basiert auf Gesetzen der Zahlentheorie**

- Aus $n=p \cdot q$ und $\text{ggT}(x, n) = 1$ folgt $x^{(p-1)(q-1)} \bmod n = 1$ (Euler-Fermat)

- Aus $e \cdot d \bmod (p-1)(q-1) = 1$ und $x < n$ folgt $(x^e)^d \bmod n = x$

- **Sicherheit: Faktorisierung ist schwer**

- Um RSA zu brechen muß die Zahl n in p und q zerlegt werden können

- Probedivision benötigt Laufzeit $\mathcal{O}(\sqrt{n})$ (1024 bit $\hat{=}$ 10^{160} Schritte)

- Bester bekannter Algorithmus benötigt $\mathcal{O}(e^{\sqrt{\log n \cdot \log(\log n)}})$ ($\hat{=}$ 10^{34} Schritte)

WARUM FUNKTIONIERT RSA IN DER PRAXIS?

- **Korrektheit basiert auf Gesetzen der Zahlentheorie**

- Aus $n=p \cdot q$ und $\text{ggT}(x, n) = 1$ folgt $x^{(p-1)(q-1)} \bmod n = 1$ (Euler-Fermat)
- Aus $e \cdot d \bmod (p-1)(q-1) = 1$ und $x < n$ folgt $(x^e)^d \bmod n = x$

- **Sicherheit: Faktorisierung ist schwer**

- Um RSA zu brechen muß die Zahl n in p und q zerlegt werden können
- Probedivision benötigt Laufzeit $\mathcal{O}(\sqrt{n})$ (1024 bit $\hat{=}$ 10^{160} Schritte)
- Bester bekannter Algorithmus benötigt $\mathcal{O}(e^{\sqrt{\log n \cdot \log(\log n)}})$ ($\hat{=}$ 10^{34} Schritte)

- **Durchführbarkeit: Ver-/Entschlüsselung**

- Naive Potenzierung $x^e \bmod n$ liegt in $\mathcal{O}(n \cdot \log n^2)$
- Aber iteriertes Quadrieren und Multiplizieren liegt in $\mathcal{O}(\log n^3)$

Es ist $x^e = \prod_{i \leq k, e_i=1} x^{2^i}$ wobei $e = \sum_{i=0}^k e_i 2^i$ Binärdarstellung von e

WARUM FUNKTIONIERT RSA IN DER PRAXIS?

- **Korrektheit basiert auf Gesetzen der Zahlentheorie**

- Aus $n=p \cdot q$ und $\text{ggT}(x, n) = 1$ folgt $x^{(p-1)(q-1)} \bmod n = 1$ (Euler-Fermat)
- Aus $e \cdot d \bmod (p-1)(q-1) = 1$ und $x < n$ folgt $(x^e)^d \bmod n = x$

- **Sicherheit: Faktorisierung ist schwer**

- Um RSA zu brechen muß die Zahl n in p und q zerlegt werden können
- Probedivision benötigt Laufzeit $\mathcal{O}(\sqrt{n})$ (1024 bit $\hat{=}$ 10^{160} Schritte)
- Bester bekannter Algorithmus benötigt $\mathcal{O}(e^{\sqrt{\log n \cdot \log(\log n)}})$ ($\hat{=}$ 10^{34} Schritte)

- **Durchführbarkeit: Ver-/Entschlüsselung**

- Naive Potenzierung $x^e \bmod n$ liegt in $\mathcal{O}(n \cdot \log n^2)$
- Aber iteriertes Quadrieren und Multiplizieren liegt in $\mathcal{O}(\log n^3)$
- Es ist $x^e = \prod_{i \leq k, e_i=1} x^{2^i}$ wobei $e = \sum_{i=0}^k e_i 2^i$ Binärdarstellung von e

- **Durchführbarkeit: Schlüsselerzeugung**

- Um sicher zu sein benötigt RSA sehr große Primzahlen
- Naive Primzahltests sind genauso langsam wie Faktorisierung
- Probabilistische Algorithmen machen Primzahltests praktikabel

NOTWENDIGE KRITERIEN FÜR PRIMZAHLEN

- **Jacobi Symbol $\left(\frac{a}{n}\right)$ für ganze Zahlen a und n**

- Zahlentheoretisch definiert über “quadratische Reste”

- Rechengesetze für $\left(\frac{a}{n}\right)$ und ungerade $n > 2$

1. Für ungerade a ist
$$\left(\frac{a}{n}\right) = \begin{cases} -\left(\frac{n}{a}\right) & \text{falls } a \equiv n \equiv 3 \pmod{4} \\ \left(\frac{n}{a}\right) & \text{sonst} \end{cases}$$

2. Für alle a ist
$$\left(\frac{a}{n}\right) = \left(\frac{a \bmod n}{n}\right)$$

3. Für $a = b \cdot 2^k$ ist
$$\left(\frac{a}{n}\right) = \left(\frac{b}{n}\right) \left(\frac{2}{n}\right)^k$$

4.
$$\left(\frac{2}{n}\right) = \begin{cases} 1 & \text{falls } n \equiv \pm 1 \pmod{8} \\ -1 & \text{falls } n \equiv \pm 3 \pmod{8} \end{cases} \quad \left(\frac{1}{n}\right) = 1 \quad \left(\frac{0}{n}\right) = 0$$

- Rechenzeit für $\left(\frac{a}{n}\right)$ liegt in $\mathcal{O}(\log n^3)$

NOTWENDIGE KRITERIEN FÜR PRIMZAHLEN

- **Jacobi Symbol $\left(\frac{a}{n}\right)$ für ganze Zahlen a und n**

- Zahlentheoretisch definiert über “quadratische Reste”

- Rechengesetze für $\left(\frac{a}{n}\right)$ und ungerade $n > 2$

1. Für ungerade a ist
$$\left(\frac{a}{n}\right) = \begin{cases} -\left(\frac{n}{a}\right) & \text{falls } a \equiv n \equiv 3 \pmod{4} \\ \left(\frac{n}{a}\right) & \text{sonst} \end{cases}$$

2. Für alle a ist
$$\left(\frac{a}{n}\right) = \left(\frac{a \bmod n}{n}\right)$$

3. Für $a = b \cdot 2^k$ ist
$$\left(\frac{a}{n}\right) = \left(\frac{b}{n}\right) \left(\frac{2}{n}\right)^k$$

4.
$$\left(\frac{2}{n}\right) = \begin{cases} 1 & \text{falls } n \equiv \pm 1 \pmod{8} \\ -1 & \text{falls } n \equiv \pm 3 \pmod{8} \end{cases} \quad \left(\frac{1}{n}\right) = 1 \quad \left(\frac{0}{n}\right) = 0$$

- Rechenzeit für $\left(\frac{a}{n}\right)$ liegt in $\mathcal{O}(\log n^3)$

- **Für jede Primzahl $n > 2$ gilt $\left(\frac{a}{n}\right) \equiv a^{(n-1)/2} \pmod{n}$ (Euler)**

- Ist n keine Primzahl, dann gilt $\left(\frac{a}{n}\right) \equiv a^{(n-1)/2} \pmod{n}$ für maximal

- die Hälfte aller $a < n$

(folgt aus der Gruppentheorie)

- Kriterium ist als Grundlage für einen RP -Algorithmus geeignet

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

1. Ist n gerade dann halte ohne zu akzeptieren

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

1. Ist n gerade dann halte ohne zu akzeptieren
2. Ansonsten wähle $a \in \{1 \dots n-1\}$ zufällig

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

1. Ist n gerade dann halte ohne zu akzeptieren
2. Ansonsten wähle $a \in \{1 \dots n-1\}$ zufällig
3. Ist $ggT(n, a) \neq 1$ dann halte ohne zu akzeptieren

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

1. Ist n gerade dann halte ohne zu akzeptieren
 2. Ansonsten wähle $a \in \{1 \dots n-1\}$ zufällig
 3. Ist $\text{ggT}(n, a) \neq 1$ dann halte ohne zu akzeptieren
 4. Ansonsten teste $\left(\frac{a}{n}\right) \equiv a^{(n-1)/2} \pmod{n}$
Ist dies der Fall, dann akzeptiere n , sonst verwerfe n
-

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

1. Ist n gerade dann halte ohne zu akzeptieren
2. Ansonsten wähle $a \in \{1 \dots n-1\}$ zufällig
3. Ist $ggT(n, a) \neq 1$ dann halte ohne zu akzeptieren
4. Ansonsten teste $\left(\frac{a}{n}\right) \equiv a^{(n-1)/2} \pmod{n}$
Ist dies der Fall, dann akzeptiere n , sonst verwerfe n

- **RP-Algorithmus für Erkennung von Zusammengesetztheit**

- Für jede Primzahl gilt $a^{(n-1)/2} \equiv J(a, n) \pmod{n}$ (Ausgabe korrekt)
- Für zusammengesetztes n gilt dies für maximal die Hälfte aller $a < n$
(Wahrscheinlichkeit für korrektes Verwerfen ist mindestens $1/2$)

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

1. Ist n gerade dann halte ohne zu akzeptieren
2. Ansonsten wähle $a \in \{1 \dots n-1\}$ zufällig
3. Ist $ggT(n, a) \neq 1$ dann halte ohne zu akzeptieren
4. Ansonsten teste $\left(\frac{a}{n}\right) \equiv a^{(n-1)/2} \pmod{n}$
Ist dies der Fall, dann akzeptiere n , sonst verwerfe n

- **RP-Algorithmus für Erkennung von Zusammengesetztheit**

- Für jede Primzahl gilt $a^{(n-1)/2} \equiv J(a, n) \pmod{n}$ (Ausgabe korrekt)
- Für zusammengesetztes n gilt dies für maximal die Hälfte aller $a < n$
(Wahrscheinlichkeit für korrektes Verwerfen ist mindestens $1/2$)

- **Rechenzeit maximal $6 * (\log n)^3$**

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

1. Ist n gerade dann halte ohne zu akzeptieren
2. Ansonsten wähle $a \in \{1 \dots n-1\}$ zufällig
3. Ist $ggT(n, a) \neq 1$ dann halte ohne zu akzeptieren
4. Ansonsten teste $\left(\frac{a}{n}\right) \equiv a^{(n-1)/2} \pmod{n}$
Ist dies der Fall, dann akzeptiere n , sonst verwerfe n

- **RP-Algorithmus für Erkennung von Zusammengesetztheit**

- Für jede Primzahl gilt $a^{(n-1)/2} \equiv J(a, n) \pmod{n}$ (Ausgabe korrekt)
- Für zusammengesetztes n gilt dies für maximal die Hälfte aller $a < n$
(Wahrscheinlichkeit für korrektes Verwerfen ist mindestens 1/2)

- Rechenzeit maximal $6 * (\log n)^3$

- Fehlerwahrscheinlichkeit bei 100 Iterationen maximal 10^{-30}

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

1. Ist n gerade dann halte ohne zu akzeptieren
2. Ansonsten wähle $a \in \{1 \dots n-1\}$ zufällig
3. Ist $ggT(n, a) \neq 1$ dann halte ohne zu akzeptieren
4. Ansonsten teste $\left(\frac{a}{n}\right) \equiv a^{(n-1)/2} \pmod{n}$
Ist dies der Fall, dann akzeptiere n , sonst verwerfe n

- **RP-Algorithmus für Erkennung von Zusammengesetztheit**

- Für jede Primzahl gilt $a^{(n-1)/2} \equiv J(a, n) \pmod{n}$ (Ausgabe korrekt)
- Für zusammengesetztes n gilt dies für maximal die Hälfte aller $a < n$
(Wahrscheinlichkeit für korrektes Verwerfen ist mindestens $1/2$)

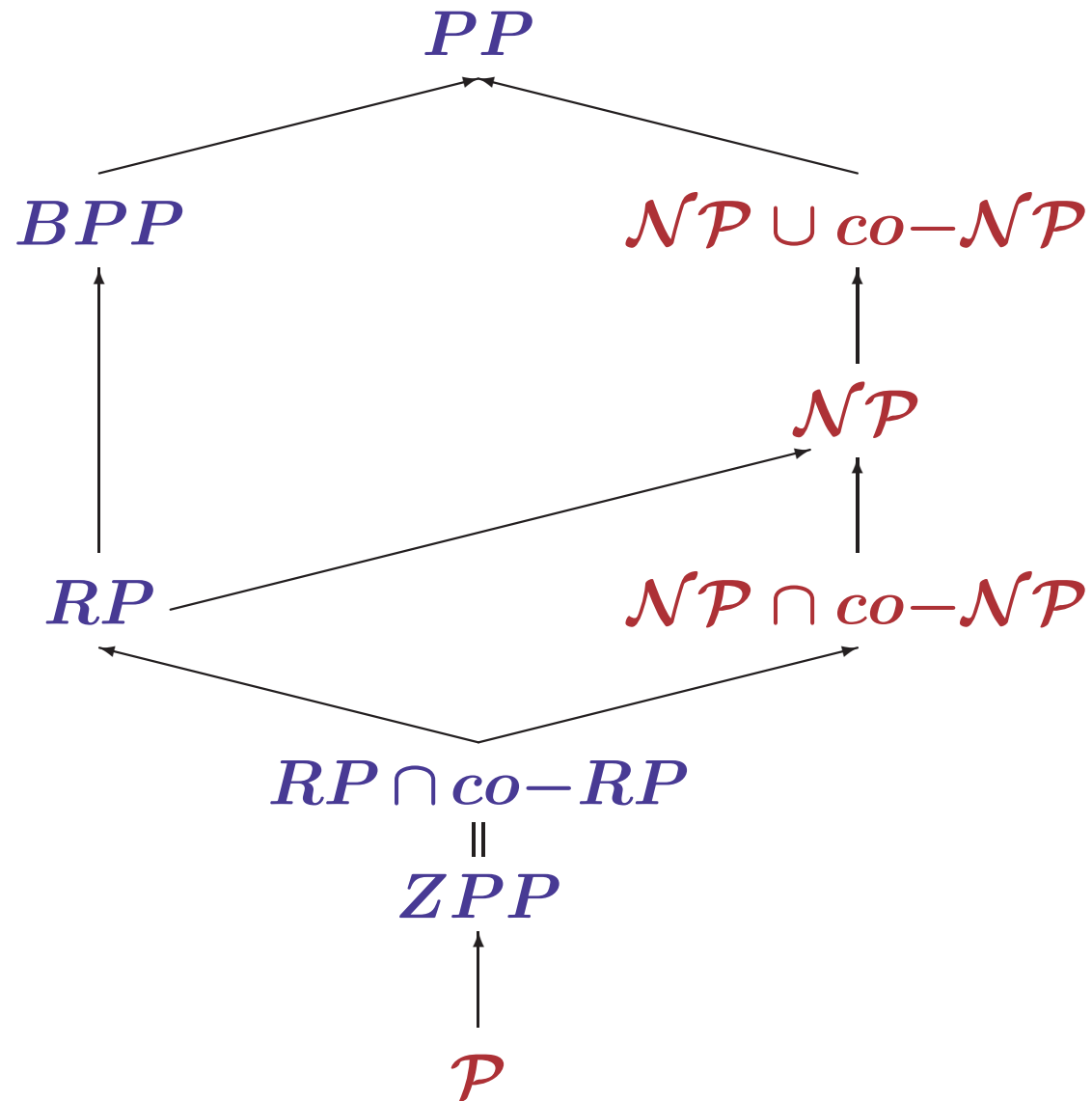
- Rechenzeit maximal $6 * (\log n)^3$

- Fehlerwahrscheinlichkeit bei 100 Iterationen maximal 10^{-30}

Mehr hierzu in der Vorlesung “Kryptographie und Komplexität” im WS 2009/10

ANHANG

HIERARCHIE PROBABILISTISCHER SPRACHKLASSEN



ZUSAMMENHÄNGE ZWISCHEN DEN SPRACHKLASSEN

- $\mathcal{P} \subseteq \underline{ZPP} \subseteq \underline{RP} \subseteq \underline{BPP} \subseteq \underline{PP}$
 - \underline{ZPP} ist wie \mathcal{P} , aber Laufzeit ist nur im Erwartungswert polynomiell
 - $\underline{BPP} \subseteq \underline{PP}$ folgt direkt aus den Definitionen
 - $\underline{RP} \subseteq \underline{BPP}$ folgt aus dem Iterationssatz für \underline{RP}

ZUSAMMENHÄNGE ZWISCHEN DEN SPRACHKLASSEN

- $\mathcal{P} \subseteq ZPP \subseteq RP \subseteq BPP \subseteq PP$

- ZPP ist wie \mathcal{P} , aber Laufzeit ist nur im Erwartungswert polynomiell

- $BPP \subseteq PP$ folgt direkt aus den Definitionen

- $RP \subseteq BPP$ folgt aus dem Iterationssatz für RP

- $ZPP = RP \cap co-RP$

- Beweis durch gegenseitige Simulation

HMU, Satz 11.17

(siehe nächste Folie)

ZUSAMMENHÄNGE ZWISCHEN DEN SPRACHKLASSEN

- $\mathcal{P} \subseteq \underline{ZPP} \subseteq RP \subseteq BPP \subseteq PP$

- ZPP ist wie \mathcal{P} , aber Laufzeit ist nur im Erwartungswert polynomiell
- $BPP \subseteq PP$ folgt direkt aus den Definitionen
- $RP \subseteq BPP$ folgt aus dem Iterationssatz für RP

- $ZPP = RP \cap co-RP$

HMU, Satz 11.17

- Beweis durch gegenseitige Simulation

(siehe nächste Folie)

- $RP \subseteq \mathcal{NP}$

HMU, Satz 11.19

- Das Verhalten einer PTM kann durch eine NTM M simuliert werden
- Da die PTM kein Wort $w \notin L$ akzeptiert, akzeptiert M ebenfalls nicht

ZUSAMMENHÄNGE ZWISCHEN DEN SPRACHKLASSEN

- $\mathcal{P} \subseteq \underline{ZPP} \subseteq \underline{RP} \subseteq \underline{BPP} \subseteq \underline{PP}$

- ZPP ist wie \mathcal{P} , aber Laufzeit ist nur im Erwartungswert polynomiell
- $BPP \subseteq PP$ folgt direkt aus den Definitionen
- $RP \subseteq BPP$ folgt aus dem Iterationssatz für RP

- $ZPP = RP \cap co-RP$

HMU, Satz 11.17

- Beweis durch gegenseitige Simulation

(siehe nächste Folie)

- $RP \subseteq \mathcal{NP}$

HMU, Satz 11.19

- Das Verhalten einer PTM kann durch eine NTM M simuliert werden
- Da die PTM kein Wort $w \notin L$ akzeptiert, akzeptiert M ebenfalls nicht

- $\mathcal{NP} \cup co-\mathcal{NP} \subseteq PP$

- NTM akzeptiert, wenn Wahrscheinlichkeit der Akzeptanz nicht Null
- Aufwendige Simulation durch PP Algorithmen möglich

BEWEIS VON $ZPP = RP \cap co-RP$

- $ZPP \supseteq RP \cap co-RP$

Seien A und \bar{A} Algorithmen für $L, \bar{L} \in RP$ mit Laufzeitgrenze $p(n)$

Wir konstruieren für L eine Las Vegas Maschine M wie folgt

BEWEIS VON $ZPP = RP \cap co-RP$

- $ZPP \supseteq RP \cap co-RP$

Seien A und \bar{A} Algorithmen für $L, \bar{L} \in RP$ mit Laufzeitgrenze $p(n)$

Wir konstruieren für L eine Las Vegas Maschine M wie folgt

(1) Lasse A auf Eingabe w laufen und akzeptiere, wenn A akzeptiert.

(2) Ansonsten lasse \bar{A} auf w laufen und verwerfe, wenn \bar{A} akzeptiert.

Wenn weder A noch \bar{A} akzeptiert haben fahre mit Schritt (1) fort.

BEWEIS VON $ZPP = RP \cap co-RP$

- $ZPP \supseteq RP \cap co-RP$

Seien A und \bar{A} Algorithmen für $L, \bar{L} \in RP$ mit Laufzeitgrenze $p(n)$

Wir konstruieren für L eine Las Vegas Maschine M wie folgt

- (1) Lasse A auf Eingabe w laufen und akzeptiere, wenn A akzeptiert.
- (2) Ansonsten lasse \bar{A} auf w laufen und verwerfe, wenn \bar{A} akzeptiert.

Wenn weder A noch \bar{A} akzeptiert haben fahre mit Schritt (1) fort.

Per Konstruktion gibt M immer nur korrekte Antworten

BEWEIS VON $ZPP = RP \cap co-RP$

- $ZPP \supseteq RP \cap co-RP$

Seien A und \bar{A} Algorithmen für $L, \bar{L} \in RP$ mit Laufzeitgrenze $p(n)$

Wir konstruieren für L eine Las Vegas Maschine M wie folgt

- (1) Lasse A auf Eingabe w laufen und akzeptiere, wenn A akzeptiert.
- (2) Ansonsten lasse \bar{A} auf w laufen und verwerfe, wenn \bar{A} akzeptiert.

Wenn weder A noch \bar{A} akzeptiert haben fahre mit Schritt (1) fort.

Per Konstruktion gibt M immer nur korrekte Antworten

Jede Runde dauert $2p(n)$ Schritte und terminiert mit Wahrscheinlichkeit $\frac{1}{2}$

Die erwartete Laufzeit ist also $2p(n) * (1 + \frac{1}{2} + \frac{1}{4} + \frac{1}{8} + \dots) = 4p(n)$

BEWEIS VON $ZPP = RP \cap co-RP$

- $ZPP \supseteq RP \cap co-RP$

Seien A und \bar{A} Algorithmen für $L, \bar{L} \in RP$ mit Laufzeitgrenze $p(n)$

Wir konstruieren für L eine Las Vegas Maschine M wie folgt

(1) Lasse A auf Eingabe w laufen und akzeptiere, wenn A akzeptiert.

(2) Ansonsten lasse \bar{A} auf w laufen und verwerfe, wenn \bar{A} akzeptiert.

Wenn weder A noch \bar{A} akzeptiert haben fahre mit Schritt (1) fort.

Per Konstruktion gibt M immer nur korrekte Antworten

Jede Runde dauert $2p(n)$ Schritte und terminiert mit Wahrscheinlichkeit $\frac{1}{2}$

Die erwartete Laufzeit ist also $2p(n) * (1 + \frac{1}{2} + \frac{1}{4} + \frac{1}{8} + \dots) = 4p(n)$

- $ZPP \subseteq RP \cap co-RP$

Sei M eine ZPP maschine für L mit erwarteter Laufzeit $p(n)$

Wir konstruieren für L (analog \bar{L}) einen RP Algorithmus A wie folgt

BEWEIS VON $ZPP = RP \cap co-RP$

- $ZPP \supseteq RP \cap co-RP$

Seien A und \bar{A} Algorithmen für $L, \bar{L} \in RP$ mit Laufzeitgrenze $p(n)$

Wir konstruieren für L eine Las Vegas Maschine M wie folgt

- (1) Lasse A auf Eingabe w laufen und akzeptiere, wenn A akzeptiert.
- (2) Ansonsten lasse \bar{A} auf w laufen und verwerfe, wenn \bar{A} akzeptiert.

Wenn weder A noch \bar{A} akzeptiert haben fahre mit Schritt (1) fort.

Per Konstruktion gibt M immer nur korrekte Antworten

Jede Runde dauert $2p(n)$ Schritte und terminiert mit Wahrscheinlichkeit $\frac{1}{2}$

Die erwartete Laufzeit ist also $2p(n) * (1 + \frac{1}{2} + \frac{1}{4} + \frac{1}{8} + \dots) = 4p(n)$

- $ZPP \subseteq RP \cap co-RP$

Sei M eine ZPP maschine für L mit erwarteter Laufzeit $p(n)$

Wir konstruieren für L (analog \bar{L}) einen RP Algorithmus A wie folgt

- (1) Simuliere M für $2p(n)$ Schritte.
- (2) Akzeptiere, wenn M akzeptiert und verwerfe sonst.

BEWEIS VON $ZPP = RP \cap co-RP$

- $ZPP \supseteq RP \cap co-RP$

Seien A und \bar{A} Algorithmen für $L, \bar{L} \in RP$ mit Laufzeitgrenze $p(n)$

Wir konstruieren für L eine Las Vegas Maschine M wie folgt

(1) Lasse A auf Eingabe w laufen und akzeptiere, wenn A akzeptiert.

(2) Ansonsten lasse \bar{A} auf w laufen und verwerfe, wenn \bar{A} akzeptiert.

Wenn weder A noch \bar{A} akzeptiert haben fahre mit Schritt (1) fort.

Per Konstruktion gibt M immer nur korrekte Antworten

Jede Runde dauert $2p(n)$ Schritte und terminiert mit Wahrscheinlichkeit $\frac{1}{2}$

Die erwartete Laufzeit ist also $2p(n) * (1 + \frac{1}{2} + \frac{1}{4} + \frac{1}{8} + \dots) = 4p(n)$

- $ZPP \subseteq RP \cap co-RP$

Sei M eine ZPP maschine für L mit erwarteter Laufzeit $p(n)$

Wir konstruieren für L (analog \bar{L}) einen RP Algorithmus A wie folgt

(1) Simuliere M für $2p(n)$ Schritte.

(2) Akzeptiere, wenn M akzeptiert und verwerfe sonst.

Per Konstruktion akzeptiert A niemals ein $w \notin L$

Für $w \in L$ ist $P(M \text{ akzeptiert in } 2p(n) \text{ Schritten}) \geq \frac{1}{2}$