

# Theoretische Informatik II

## Einheit 6.4

### Grenzen überwinden



1. Pseudopolynomielle Algorithmen
2. Approximation von  
Optimierungsproblemen
3. Probabilistische Algorithmen
4. 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.  $\mathcal{O}(n^6)$ )
  - **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, ...)
- **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

**Manche Probleme sind gar nicht so schwer wie befürchtet**

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

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

- Es gibt “bessere” Lösungen für *KP*

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

- 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 $KP$

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

- **Betrachte Subprobleme  $KP(k, g)$**  ( $g$  und  $k$  fest)
  - Verwende Gegenstände  $1, \dots, k$  und Maximalgewicht  $g \leq G$
  - Definiere 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 $KP$

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

- **Betrachte Subprobleme  $KP(k, g)$**  ( $g$  und  $k$  fest)
  - Verwende Gegenstände  $1, \dots, k$  und Maximalgewicht  $g \leq G$
  - Definiere 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  $KP$  iterativ**
  - 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  $k \leq n, g \leq G$
  - Laufzeit ist  $\mathcal{O}(n * G)$

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

# PSEUDOPOLYNOMIELLE ALGORITHMEN

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



## 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  $\mathcal{O}(n * (\log G + \log A))$

## 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  $\mathcal{O}(n * (\log G + \log A))$
- **$KP$  ist ein Zahlproblem**
  - $L \subseteq \Sigma^*$  ist **Zahlproblem**, wenn  $MAX(w)$ , die größte in einer Eingabe  $w$  codierte Zahl, nicht durch ein Polynom beschränkt werden kann
  - Weitere Zahlprobleme: *PARTITION, BPP, TSP, MSP, ...*
  - Keine Zahlprobleme: *CLIQUE, VC, IS, SGI, LCS, DHC, HC, GC, ...*

## 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  $\mathcal{O}(n * (\log G + \log A))$
- **$KP$  ist ein Zahlproblem**
  - $L \subseteq \Sigma^*$  ist **Zahlproblem**, wenn  $MAX(w)$ , die größte in einer Eingabe  $w$  codierte Zahl, nicht durch ein Polynom beschränkt werden kann
  - Weitere Zahlprobleme: *PARTITION, BPP, TSP, MSP, ...*
  - Keine Zahlprobleme: *CLIQUE, VC, IS, SGI, LCS, DHC, HC, GC, ...*
- **$KP$  hat pseudopolynomielle Lösung**
  - Algorithmen für ein Zahlproblem  $L \subseteq \Sigma^*$  sind **pseudopolynomiell**, wenn ihre Rechenzeit durch ein Polynom in  $|w|$  und  $MAX(w)$  beschränkt ist

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

$$L_p \equiv \{w \in L \mid \text{MAX}(w) \leq p(|w|)\} \in \mathcal{P}$$

- 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  $L \subseteq \Sigma^*$  pseudopolynomiell lösbar, so ist für jedes Polynom  $p$   
 $L_p \equiv \{w \in L \mid MAX(w) \leq p(|w|)\} \in \mathcal{P}$
  - 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**
  - $L \subseteq \Sigma^*$  **stark  $\mathcal{NP}$ -vollständig**  $\equiv L_p \mathcal{NP}$ -vollständig für ein Polynom  $p$
  - $L$  stark  $\mathcal{NP}$ -vollständig  $\Rightarrow L$  hat keine pseudopolynomielle Lösung

**Einschränkung auf kleine Zahlen löst das  $\mathcal{P}$ - $\mathcal{NP}$  Problem nicht**

- **Viele Probleme haben Optimierungsvarianten**

- $CLIQUE_{opt}$ : bestimme die größte Clique im Graphen  
(Gesucht ist entweder maximale Cliquengröße oder konkrete maximalen Clique)
- $TSP_{opt}$ : 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 hat Laufzeit  $\mathcal{O}(|V| + |E| \log |E|)$

(Anhang)

# 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  $x$  ( $R_A(x) \geq 1$  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**

# TYPEN VON APPROXIMATIONSALGORITHMEN

- **Polynomielles Approximationsschema** (gut)
  - Algorithmus kann optimale Lösung beliebig genau approximieren
  - Algorithmus  $A$  ist **parametrisiert** in gewünschter Güte  $1+\epsilon$
  - $A$  berechnet für alle  $x$  und alle  $\epsilon > 0$  eine Lösung  $A_\epsilon(x)$  mit  $R_{A_\epsilon}(x) \leq 1+\epsilon$
  - Rechenzeit ist für jedes (feste)  $\epsilon$  **polynomiell** in  $|x|$
  - Rechenzeit kann bei kleinem  $\epsilon$  sehr hoch werden
- **Echt polynomielles Approximationsschema** (besser)
  - $A$  kann optimale Lösung beliebig genau **und effizient** approximieren
  - $A$  berechnet für alle  $x$  und alle  $\epsilon > 0$  eine Lösung  $A_\epsilon(x)$  mit  $R_{A_\epsilon}(x) \leq 1+\epsilon$
  - Rechenzeit ist **polynomiell** in  $|x|$  und  $\epsilon^{-1}$
- **Approximation mit additivem Fehler  $k$**  (fast ideal)
  - Approximationsfehler wird mit wachsendem Optimum beliebig gering
  - Für alle  $x$  gilt  $|OPT(x) - W(x, A(x))| \leq k$
  - Rechenzeit ist **polynomiell** in  $|x|$

- **Echt polynomielle Approximationsschemata für ganzzahlige Probleme liefern pseudopolynomielle Lösungen**

Bedingung:  $OPT_L(w)$  beschränkt durch Polynom  $p$  in  $|x|$  und  $MAX(x)$

– Sei  $A$  echt polynomielle Approximationsschemata für  $L$

– Für Eingabe  $x$  wähle  $\epsilon = (p(|w|, MAX(w))+1)^{-1}$  und berechne  $A_\epsilon(x)$

– Wegen  $R_{A_\epsilon}(x) \leq 1 + \epsilon$  folgt  $|W(x, A(x)) - OPT_L(x)|$

$$\leq |(1+\epsilon)*OPT_L(x) - OPT_L(x)| = \epsilon*OPT_L(x)$$

$$\leq \epsilon*p(|w|, MAX(w)) < 1$$

– Da  $L$  ganzzahlig ist, muß  $A_\epsilon(x)$  optimal sein

–  $A$  ist pseudopolynomiell, da Rechenzeit Polynom in  $p(|w|, MAX(w))+1$

- **Echt polynomielle Approximationsschemata für ganzzahlige Probleme liefern pseudopolynomielle Lösungen**

Bedingung:  $OPT_L(w)$  beschränkt durch Polynom  $p$  in  $|x|$  und  $MAX(x)$

– Sei  $A$  echt polynomielle Approximationsschemata für  $L$

– Für Eingabe  $x$  wähle  $\epsilon = (p(|w|, MAX(w))+1)^{-1}$  und berechne  $A_\epsilon(x)$

– Wegen  $R_{A_\epsilon}(x) \leq 1 + \epsilon$  folgt  $|W(x, A(x)) - OPT_L(x)|$

$$\leq |(1 + \epsilon) * OPT_L(x) - OPT_L(x)| = \epsilon * OPT_L(x)$$

$$\leq \epsilon * p(|w|, MAX(w)) < 1$$

– Da  $L$  ganzzahlig ist, muß  $A_\epsilon(x)$  optimal sein

–  $A$  ist pseudopolynomiell, da Rechenzeit Polynom in  $p(|w|, MAX(w))+1$

- **Stark  $\mathcal{NP}$ -harte ganzzahlige Probleme haben keine echt polynomielle Approximationsschemata**

– Falls  $P \neq \mathcal{NP}$  und  $OPT_L(w)$  beschränkt durch ein  $p(|w|, MAX(w))$

↳  $TSP$  kann kein echt polynomielles Approximationsschema haben

# POSITIVE ERGEBNISSE

- **Knapsack** hat echt polynomielle Approximationsschemata
  - FPTAS Algorithmus berechnet  $A_\epsilon(x)$  in Laufzeit  $\mathcal{O}(|x|^3 * \epsilon^{-1})$   
(en.wikipedia.org)
- **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  $x$ , also  $R_A^\infty = 11/9$   
(en.wikipedia.org)
- **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)

**Bessere Approximationen sind jeweils 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  $x$  (Folie 14)
- **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  $x$  (Folie 15)
- **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 asymptotische worst-case Güte**
  - Es gibt keinen polynomiellen Algorithmus  $A_{TSP}$  mit  $R_{A_{TSP}}^\infty < \infty$  (Folie 16)

**Nachweise verwenden verschiedenartige 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

– 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
- 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

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

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

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

**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

$R_{A_{TSP}}^\infty = r$  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

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

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

- 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

## “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
- **Laufzeit abhängig vom Pivotelement**
  - $\mathcal{O}(n \cdot \log_2 n)$ , 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**  $\mathcal{O}(n \cdot \log_2 n)$
  - Im Durchschnitt schneller als deterministische  $\mathcal{O}(n \cdot \log_2 n)$ -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

- **$Prob[M \downarrow w]$ : Wahrscheinlichkeit der Akzeptanz von  $w$**

- 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
  - Entscheidungsalgorithmus mit polynomieller Laufzeit
  - Wahrscheinlichkeit für korrekte Antwort ist mindestens  $1/2$   
d.h.  $w \in L \Rightarrow \text{Prob}[M \downarrow w] \geq 1/2$  und  $w \notin L \Rightarrow \text{Prob}(M \not\downarrow w) > 1/2$
- **BPP: Bounded error Probabilistic Polynomial**
  - Entscheidungsalgorithmus mit polynomieller Laufzeit
  - Wahrscheinlichkeit für korrekte Antwort ist mindestens  $1/2 + \epsilon$
- **RP: Random Polynomial**
  - Entscheidungsalgorithmus mit polynomieller Laufzeit
  - Wahrscheinlichkeit für Akzeptanz ist mindestens  $1/2$ , wenn  $w \in L$   
es reicht, daß Wahrscheinlichkeit für Akzeptanz mindestens  $\epsilon$  für ein  $\epsilon > 0$  ist
  - Algorithmus akzeptiert mit Sicherheit nicht, wenn  $w \notin L$
- **ZPP: Zero error PP** Las-Vegas-Algorithmen
  - Entscheidungsalgorithmus, bei dem die Antwort immer korrekt ist
  - Laufzeit nur im Erwartungswert polynomiell (in extrem seltenen Fällen länger)

Alternative Definitionen verwenden Erkenntnis  $ZPP = RP \cap \text{co-RP}$  (Folie 29)

## 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 verwerfe

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

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

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

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

### • Wahrscheinlichkeit korrekter Akzeptanz $> 1/2$

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

## 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  $w \in L \Rightarrow \text{Prob}[M \downarrow w] \geq 1 - 2^{-k}$   
und  $w \notin L \Rightarrow \text{Prob}(M \not\downarrow w) = 1$
  - 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  $t > \frac{k-1}{-\log(1-4\epsilon^2)}$   
 $w \in L \Rightarrow \text{Prob}(M^t \downarrow w) > 1 - 2^{-k}$  und  $w \notin L \Rightarrow \text{Prob}(M^t \not\downarrow w) > 1 - 2^{-k}$
  - Aufwendige Analyse (siehe Wegener 75–77 für Details)
- **Keine Aussagen für  $PP$  Algorithmen möglich**

## ● 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

## ● 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  $(x^e)^d \bmod n = x$

- **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  $\mathcal{O}(\log n^3)$

- **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 ist  $n$  keine Primzahl. Halte ohne zu akzeptieren
2. Ansonsten wähle  $a \in \{1 \dots n-1\}$  zufällig
3. Ist  $\text{ggT}(n, a) \neq 1$  oder  $\left(\frac{a}{n}\right) = 0$  dann ist  $n$  keine Primzahl.  
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  $n$   
Ansonsten ist  $n$  keine Primzahl. Halte ohne zu akzeptieren

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

1. Ist  $n$  gerade dann ist  $n$  keine Primzahl. Halte ohne zu akzeptieren
2. Ansonsten wähle  $a \in \{1 \dots n-1\}$  zufällig
3. Ist  $\text{ggT}(n, a) \neq 1$  oder  $\left(\frac{a}{n}\right) = 0$  dann ist  $n$  keine Primzahl.  
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  $n$   
Ansonsten ist  $n$  keine Primzahl. Halte ohne zu akzeptieren

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

- Für jede Primzahl gilt  $a^{(n-1)/2} \equiv \left(\frac{a}{n}\right) \pmod n$  (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 ist  $n$  keine Primzahl. Halte ohne zu akzeptieren
2. Ansonsten wähle  $a \in \{1 \dots n-1\}$  zufällig
3. Ist  $\text{ggT}(n, a) \neq 1$  oder  $\left(\frac{a}{n}\right) = 0$  dann ist  $n$  keine Primzahl.  
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  $n$   
Ansonsten ist  $n$  keine Primzahl. Halte ohne zu akzeptieren

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

- Für jede Primzahl gilt  $a^{(n-1)/2} \equiv \left(\frac{a}{n}\right) \pmod n$  (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 14/15



# ANHANG

# DER KRUSKAL ALGORITHMUS

## Bestimme einen MWST in einem Graphen $G$

- **MWST: aufspannender Baum mit minimalem Gewicht**
  - Ein Baum **spannt einen Graphen auf**, wenn jeder Knoten von  $G$  von der Wurzel des Baums aus erreichbar ist

# DER KRUSKAL ALGORITHMUS

## Bestimme einen MWST in einem Graphen $G$

- **MWST: aufspannender Baum mit minimalem Gewicht**
  - Ein Baum **spannt einen Graphen auf**, wenn jeder Knoten von  $G$  von der Wurzel des Baums aus erreichbar ist
- **Erzeuge Zusammenhangskomponenten in  $G$** 
  - Initialwert ist  $\{v\}$  für jeden Knoten  $v \in V$  ( $Z := \{\{v\} \mid v \in V\}$ )
  - Betrachte eine **neue Kante**  $e \in E$  mit **geringstem Gewicht**  
Falls  $e$  Knoten aus verschiedenen Zusammenhangskomponenten verbindet, füge  $e$  dem MWST hinzu und **vereine die beiden Komponenten**
  - Wiederhole dies, bis alle Knoten in einer Komponente sind oder alle Kanten betrachtet wurden

## Bestimme einen MWST in einem Graphen $G$

- **MWST: aufspannender Baum mit minimalem Gewicht**
  - Ein Baum **spannt einen Graphen auf**, wenn jeder Knoten von  $G$  von der Wurzel des Baums aus erreichbar ist
- **Erzeuge Zusammenhangskomponenten in  $G$** 
  - Initialwert ist  $\{v\}$  für jeden Knoten  $v \in V$  ( $Z := \{\{v\} \mid v \in V\}$ )
  - Betrachte eine **neue Kante**  $e \in E$  mit **geringstem Gewicht**  
Falls  $e$  Knoten aus verschiedenen Zusammenhangskomponenten verbindet, füge  $e$  dem MWST hinzu und **vereine die beiden Komponenten**
  - Wiederhole dies, bis alle Knoten in einer Komponente sind oder alle Kanten betrachtet wurden
- **Implementierbar mit Laufzeit  $\mathcal{O}(|V| + |E| \log |E|)$** 
  - Liste der Kanten muß zuerst nach Gewicht sortiert werden
  - Zusammenhangskomponenten müssen mit Pointern repräsentiert werden
  - Turingmaschine würde Laufzeit  $\mathcal{O}((|V| + |E|)^4)$  benötigen HMU §10.1.2

# 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  $RP$

# 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

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

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

(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

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

# HIERARCHIE PROBABILISTISCHER SPRACHKLASSEN

