

# Theoretische Informatik



## Einheit 4

### Komplexitätstheorie



1. Komplexitätsmaße
2. Komplexität von Algorithmen (obere Schranken)
3. Komplexität von Problemen (untere Schranken)
4. NP-Vollständigkeit

# KOMPLEXITÄTSTHEORIE

– WAS KANN MIT VERTRETbareM AUFWAND GELÖST WERDEN? –

- **Berechenbarkeit alleine reicht nicht**

- Lösungen müssen effizient sein in praktischen Anwendungen
- Berechenbarkeit/Entscheidbarkeit löst nur die Grundsatzfrage

- **Komplexität: Analyse benötigter Ressourcen**

- Zeitbedarf des Algorithmus Time
- Speicherbedarf des Verfahrens (RAM, Harddisk) Space
- Netzzugriffe, Zugriff auf andere Medien

- **Meßgröße muß unabhängig sein von**

- Konkreter Hardware
- Konkreter Programmiersprache
- Optimierungsfähigkeiten des Compilers
- Auswahl der Testdaten



**Abstrakte Komplexitätsmaße erforderlich**

- **Asymptotisches Verhalten von Algorithmen**

- **Komplexitätsfunktion**: Bedarf abhängig von der Größe der Eingabe
- Abschätzung der Komplexität **großer Probleme**

- **Analyse konkreter Verfahren**

- **Maximaler Verbrauch** im Einzelfall **Worst case**  
Wichtig bei sicherheitskritischen Anwendungen
- **Durchschnittlicher Bedarf** im Langzeitverhalten **Average case**  
Verlangt mathematisch schwierige statistische Analyse

- **Analyse von Problemen**

- Wie effizient ist die **bestmögliche Lösung**? **Untere Schranken**
- Wieviel kann durch **Hardwaresteigerungen** erreicht werden?
- Welche Verbesserung liefert **Parallelität** bzw. **Nichtdeterminismus**
- Welche Probleme sind gleich schwierig? **Komplexitätsklassen**
- **Gibt es Probleme, die nicht effizient lösbar sind?**

# Theoretische Informatik



## Einheit 4.1

### Komplexitätsmaße



1. Zeit- und Platzkomplexität
2. Asymptotische Analyse
3. Praktische Konsequenzen

- **Rechenzeit**  $t_\tau(w)$

- Anzahl der Elementaroperationen von  $\tau$  bis Berechnung terminiert
- Abhängig von konkreter Eingabe  $w$

- **Zeitkomplexität**  $time_\tau(n) = \max\{t_\tau(w) \mid |w|=n\}$

- Maximale Rechenzeit relativ zur Größe  $n$  der Eingabe (worst-case)

- **Speicherbedarf**  $s_\tau(w)$

- Anzahl der Bandzellen, die  $\tau$  während der Berechnung aufsucht

- **Platzkomplexität**  $space_\tau(n) = \max\{s_\tau(w) \mid |w|=n\}$

- Maximaler Speicherbedarf relativ zur Größe  $n$  der Eingabe (worst-case)

Analoge Maße für andere Berechnungsmodelle  
einschließlich nichtdeterministischer Maschinen

# KOMPLEXITÄTSANALYSE EINER TURING-MASCHINE

•  $\tau_1 = (\{s_0\}, \{1\}, \{b, 1\}, \delta_1, s_0, b)$  mit  $\delta_1 =$

| $s$   | $a$ | $s'$  | $a'$ | $P$ |
|-------|-----|-------|------|-----|
| $s_0$ | 1   | $s_0$ | 1    | r   |
| $s_0$ | b   | $s_0$ | 1    | h   |

• Mathematische Analyse:

- Anfangskonfiguration:  $\alpha(1^n) = (s_0, f_n, 0)$ , wobei  $f_n(j) = \begin{cases} 1 & \text{falls } j \in \{0, \dots, n-1\}, \\ b & \text{sonst} \end{cases}$
- Nachfolgekonfigurationen:  $\hat{\delta}(s_0, f_n, j) = \begin{cases} (s_0, f_n, j+1) & \text{falls } j \in \{0, \dots, n-1\}, \\ (s_0, f_{n+1}, n) & \text{falls } j=n \end{cases}$
- Terminierung:  $\min\{j \mid \hat{\delta}^j(s_0, f_n, 0) = (s_0, f_n, j) \wedge \delta(s_0, f_n(j)) = (s_0, b, h)\} = n$



$$t_{\tau_1}(1^n) = n+1 \quad \text{und} \quad time_{\tau_1}(n) = n+1 \quad \text{für alle } n$$

- **Genaue Betrachtungen sind unpraktikabel**

- Zu mühsam bei nichttrivialen Algorithmen
- Zu abhängig von Programmierdetails und Maschinenmodell
- Welches Maschinenmodell sollte der Standard sein?

- **Abschätzung der Komplexität**

- Nur asymptotisches Verhalten auf großen Problemen ist interessant
  - ↳ Einheitskostenmodell: Vereinfachte Zählung von Elementaroperationen
  - ↳ Additive Konstanten werden nicht berücksichtigt
  - ↳ Konstante Faktoren werden nicht berücksichtigt



**Analyse des wesentlichen  
Laufzeitverhaltens/Speicherbedarfs**

## ● Asymptotischer Vergleich von Funktionen

- $f_2$  wächst schneller als  $f_1$ , falls  $f_1(n) \leq f_2(n)$  für alle  $n \in \mathbb{N}$
- $f_2$  wächst asymptotisch schneller als  $f_1$ , falls es ein  $n_0 \in \mathbb{N}$  gibt mit  

$$f_1(n) \leq f_2(n) \text{ für alle } n \geq n_0$$

## ● Ordnung $\mathcal{O}(f)$ einer Funktion

- $\mathcal{O}(f) = \{g : \mathbb{N} \rightarrow \mathbb{N} \mid \exists n_0, c. \forall n \geq n_0. g(n) \leq c * f(n)\}$
- Alternativ:  $\mathcal{O}(f) = \{g : \mathbb{N} \rightarrow \mathbb{N} \mid \exists k, c. \forall n. g(n) \leq k + c * f(n)\}$
- Gängige Schreibweisen
  - $g = \mathcal{O}(f)$  bedeutet  $g \in \mathcal{O}(f)$ ,  $\mathcal{O}(f_1) = \mathcal{O}(f_2)$  bedeutet  $\mathcal{O}(f_1) \subseteq \mathcal{O}(f_2)$
  - $\mathcal{O}(1) \equiv \mathcal{O}(\lambda n. 1)$ ,  $\mathcal{O}(n) \equiv \mathcal{O}(\lambda n. n)$ ,  $\mathcal{O}(n^2) \equiv \mathcal{O}(\lambda n. n^2)$ , ...

## ● Ordnung konkreter Funktionen

- Konstante Funktion:  $g_1(n) = k$  für alle  $n$   $g_1 \in \mathcal{O}(1)$
- Polynome:  $g_2(n) = c_0 + c_1 * n + \dots + c_m * n^m$   $g_2 \in \mathcal{O}(n^m)$
- Logarithmenfunktionen:  $g_3(n) = \log_b n$   $g_3 \in \mathcal{O}(\log_2 n)$
- Fakultätsfunktion:  $g_4(n) = n! = 1 * 2 * \dots * n$   $g_4 \in \mathcal{O}(n^n)$

## ● Asymptotischer Effizienzvergleich

- $\tau_1$  ist schneller als  $\tau_2$ , falls  $time_{\tau_1}(n) \leq time_{\tau_2}(n)$  für alle  $n \in \mathbb{N}$
- $\tau_1$  ist asymptotisch schneller als  $\tau_2$ , falls es ein  $n_0 \in \mathbb{N}$  gibt mit  $time_{\tau_1}(n) \leq time_{\tau_2}(n)$  für alle  $n \geq n_0$

## ● Komplexität $\mathcal{O}(f)$

- $\tau$  hat Zeitkomplexität  $\mathcal{O}(f)$ , falls  $time_{\tau} \in \mathcal{O}(f)$
- $\tau$  hat Platzkomplexität  $\mathcal{O}(f)$ , falls  $space_{\tau} \in \mathcal{O}(f)$

## ● Komplexitätsklassen

- $\tau$  hat konstante (Zeit-)komplexität, falls  $time_{\tau} \in \mathcal{O}(1)$
- $\tau$  hat logarithmische Komplexität, falls  $time_{\tau} \in \mathcal{O}(\log_2 n)$
- $\tau$  hat lineare Komplexität, falls  $time_{\tau} \in \mathcal{O}(n)$
- $\tau$  hat quadratische Komplexität, falls  $time_{\tau} \in \mathcal{O}(n^2)$
- $\tau$  hat kubische Komplexität, falls  $time_{\tau} \in \mathcal{O}(n^3)$
- $\tau$  hat polynomielle Komplexität, falls  $time_{\tau} \in \mathcal{O}(n^k)$  für ein  $k \in \mathbb{N}$
- $\tau$  hat exponentielle Komplexität, falls  $time_{\tau} \in \mathcal{O}(2^{n^k})$  für ein  $k \in \mathbb{N}$
- $\tau$  hat superexponentielle Komplexität, falls  $time_{\tau} \in \mathcal{O}(2^{2^{n^k}})$  für ein  $k \in \mathbb{N}$

⋮

Analoge Klassen für Platzkomplexität

# WIE SCHNELL WÄCHST RECHENZEIT MIT DER GRÖSSE DER EINGABE?

| Rechenzeiten auf 3.3 Ghz Prozessor |              |             |             |              |                |             |     |             |             |
|------------------------------------|--------------|-------------|-------------|--------------|----------------|-------------|-----|-------------|-------------|
| Größe $n$                          | 10           | 20          | 30          | 40           | 50             | 60          | ... | 1000        | 1.000.000   |
| Wachstum                           |              |             |             |              |                |             |     |             |             |
| $\log_2 n$                         | 1ns          | 2ns         |             | 3ns          |                |             |     | 10ns        | 100ns       |
| $n$                                | 3ns          | 6ns         | 9ns         | 12ns         | 15ns           | 18ns        |     | 300ns       | 300 $\mu$ s |
| $n^2$                              | 30ns         | 120ns       | 270ns       | 480ns        | 750ns          | 1.1 $\mu$ s |     | 300 $\mu$ s | 300s        |
| $n^3$                              | 300ns        | 2.4 $\mu$ s | 8.1 $\mu$ s | 19.2 $\mu$ s | 37.5 $\mu$ s   | 64 $\mu$ s  |     | 300ms       | 9.5y        |
| $2^n$                              | 300ns        | 300 $\mu$ s | 300ms       | 300s         | 83.3h          | 9.5y        |     |             |             |
| $3^n$                              | 17.8 $\mu$ s | 1.1s        | 17.3h       | 116y         | 2.500.000.000y |             |     |             |             |

**Wieviel mehr kann man in der gleichen Zeit berechnen,  
wenn Computer um den Faktor 1000 schneller sind?**

|                   | $\log_2 n$              | $n$       | $n^2$   | $n^3$   | $2^n$   | $3^n$  |
|-------------------|-------------------------|-----------|---------|---------|---------|--------|
| Problemsteigerung | 10 <sup>300</sup> -fach | 1000-fach | 31-fach | 10-fach | plus 10 | plus 6 |

- **Große Probleme benötigen polynomielle Lösungen**
  - Exponentielle Algorithmen sind für die Praxis unakzeptabel
  - Auch innerhalb der polynomiellen Komplexität gibt es große Unterschiede
- **Bessere Hardware ist selten eine Lösung**
  - Wenn Algorithmen schlecht sind, nützt die beste Hardware wenig
  - Es lohnt sich, in die Verbesserung von Algorithmen zu investieren
- **Es gibt noch ungeklärte Fragen**
  - Kann Parallelismus signifikante Effizienzsteigerung bewirken?
    - z.B. von exponentieller auf polynomielle Zeit?
  - Was ist der Zusammenhang zwischen Platzbedarf und Laufzeitverhalten
    - Bisher nur grobe Abschätzungen bekannt

# Theoretische Informatik



## Einheit 4.2



## Abschätzung der Komplexität von Algorithmen

1. Suchverfahren
2. Sortieralgorithmen

## Obere Schranken für die Laufzeit

- **Analyse auf Ebene der Algorithmen**

- Algorithmische Elementaroperationen gelten als ein Schritt
- Meist konstanter Expansionsfaktor bei Übersetzung in Maschinensprache
- $+$ ,  $-$ ,  $*$ ,  $/$ , ... Einzelschritte, wenn Zahlengröße beschränkt (z.B. 64-bit)
- Höherer Aufwand bei beliebig großen Zahlen

- **Analyse abstrakter sequentieller Algorithmen**

- Asymptotische Komplexität ist unabhängig von Programmiersprache
- Parallele/nichtdeterministische Maschinen haben evtl. bessere Laufzeit

# SEQUENTIELLE SUCHE

Teste, ob eine Zahl  $x$  in einer Liste  $L$  vorkommt

- **Durchsuche Liste  $L$  von links nach rechts**

```
function searchseq(x,L) ≡  
    found := false;  
    for i = 1 to length(L) do  
        if L[i]=x then found:=true  
    od;  
    return found;
```

Verfahren ist anwendbar auf beliebige Listen

- **Laufzeitanalyse**

- Eine Operation für Initialisierung `found:=false`
  - Je 2 Operationen pro Element von  $L$  in der `for`-Schleife
  - Eine Operation für Ausgabe des Ergebnisses
  - Insgesamt  $2n+2$  Schritte, wenn  $n$  die Größe der Liste  $L$  ist
- ↳ **Sequentielle Suche ist in  $\mathcal{O}(n)$**

# BINÄRE SUCHE

Nur anwendbar, wenn Liste  $L$  geordnet ist

- Teste mittleres Element; suche dann rechts oder links

```
function searchbin(x,L) ≡  
  function searchbound(x,L,left,right) ≡  
    if left>right then return false  
    else  
      mid := (left+right) div 2;  
      if x<L[mid] then searchbound(x,L,left,mid-1)  
      elseif x>L[mid] then searchbound(x,L,mid+1,right)  
      else return true  
    fi;  
  return searchbound(x,L,1,length(L))
```

- Grobe Laufzeitanalyse

- Konstante Anzahl von Operationen pro Aufruf von `searchbound`
- Wie oft wird `searchbound` aufgerufen?

# BINÄRE SUCHE – ANALYSE

```
function searchbin(x,L) ≡  
  function searchbound(x,L,left,right) ≡  
    if left>right then return false  
    else  
      mid := (left+right) div 2;  
      if x<L[mid] then searchbound(x,L,left,mid-1)  
      elseif x>L[mid] then searchbound(x,L,mid+1,right)  
      else return true  
    fi;  
  return searchbound(x,L,1,length(L))
```

Abstand von **left** und **right** halbiert sich pro Aufruf (mit Abrundung)

Anzahl von Operationen pro Aufruf von **search**<sub>bound</sub> ist eine Konstante  $k$

Abstand zu Beginn ist  $n-1$  ( $n$  ist die Größe der Liste  $L$ )

**search**<sub>bound</sub> terminiert bei Erfolg oder wenn Abstand Null ist

Lösung der Gleichung  $time(n) = k + time(\lfloor n/2 \rfloor)$  ist  $time(n) = k * \log_2 n$



**Binäre Suche ist in  $\mathcal{O}(\log_2 n)$**

- **Ordne Elemente in aufsteigender Reihenfolge**

- Geordnete Listen unterstützen **effizienten Zugriff** auf Elemente
- Eine der häufigsten Operationen in der Programmierung

- **Viele Verfahren bekannt**

- **Insertion Sort**: Einfügen des Listenanfangs in geordnete Teilliste
- **Selection Sort**: Auswahl des jeweils kleinsten Elements als Listenanfang
- **Bubblesort**: Austauschen benachbarter Elemente
- **Quicksort**: Aufteilung nach Größe, Sortieren der entstehenden Teillisten
- **Mergesort**: Aufteilen in Teillisten, Sortieren und Mischen der Teillisten
- **Mergesort (II)**: Identifizieren und Mischen geordneter Teillisten

Auswahl des 'besten' Verfahrens hängt von Größe des Problems ab

# BUBBLESORT

Fortlaufender Vergleich benachbarter Elemente  
Austausch bei falscher Reihenfolge

```
function bubblesort(L) ≡  
  for upper = length(L)-1 downto 1 do  
    for j = 1 to upper do  
      if L[j]>L[j+1] then  
        aux := L[j];  
        L[j] := L[j+1];  
        L[j+1] := aux  
      fi  
    od  
  od
```

## ● Beispiel einer Sortierung mit Bubblesort

|   |   |   |   |   |   |   |
|---|---|---|---|---|---|---|
| 1 | 2 | 5 | 6 | 7 | 8 | 9 |
|---|---|---|---|---|---|---|



Elemente steigen wie Blasen auf, bis sie auf größere treffen

# BUBBLESORT - LAUFZEITANALYSE

```
function bubblesort(L) ≡  
  for upper = length(L)-1 downto 1 do  
    for j = 1 to upper do  
      if L[j]>L[j+1] then  
        aux := L[j];  
        L[j] := L[j+1];  
        L[j+1] := aux  
      fi  
    od  
  od
```

- **Feste Anzahl von Operationen im Schleifenrumpf**
  - Vergleich benachbarter Elemente
  - ggf. **Austausch** unter Verwendung einer Hilfsvariablen
- **Anzahl Schleifen abhängig von Listengröße  $n$** 
  - Innere Schleife wird jeweils genau **upper**-mal durchlaufen
  - Insgesamt  $n-1 + n-2 + \dots + 2 + 1 = n*(n-1)/2$  Durchläufe



**Bubblesort ist in  $\mathcal{O}(n^2)$**

# SORTIEREN SCHNELLER ALS $\mathcal{O}(n^2)$

- **Identifiziere Läufe**, d.h. geordnete Teilfolgen

|   |   |   |   |   |   |   |
|---|---|---|---|---|---|---|
| 9 | 7 | 8 | 2 | 1 | 5 | 6 |
|---|---|---|---|---|---|---|

- **Verschmelze Läufe** zu neuen Läufen

|   |   |   |   |   |   |   |
|---|---|---|---|---|---|---|
| 9 | 7 | 8 | 2 | 1 | 5 | 6 |
| 7 | 8 | 9 | 1 | 2 | 5 | 6 |

– Länge der Läufe wächst – Anzahl halbiert sich

- **Wiederhole bis Folge geordnet**

|   |   |   |   |   |   |   |
|---|---|---|---|---|---|---|
| 7 | 8 | 9 | 1 | 2 | 5 | 6 |
| 1 | 2 | 5 | 6 | 7 | 8 | 9 |

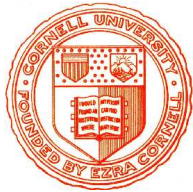
– Liste ist eine einzige (komplett) geordnete Teilfolge ✓

- **Abstrakte Skizze reicht** für Laufzeitanalyse
- **Verschmelzen ist in  $\mathcal{O}(n)$** 
  - Folge wird jeweils komplett durchlaufen
- **Verschmelzen halbiert Anzahl der Läufe**
  - Je **zwei Läufe** werden **zu einem** gemischt
  - Nach maximal  **$\log_2 n$  Verschmelzungen** bleibt ein einziger Lauf übrig  
d.h. die **Liste ist sortiert**



**Sortieren durch Verschmelzen ist in  $\mathcal{O}(n * \log_2 n)$**

# Theoretische Informatik



## Einheit 4.3

### Komplexität von Problemen



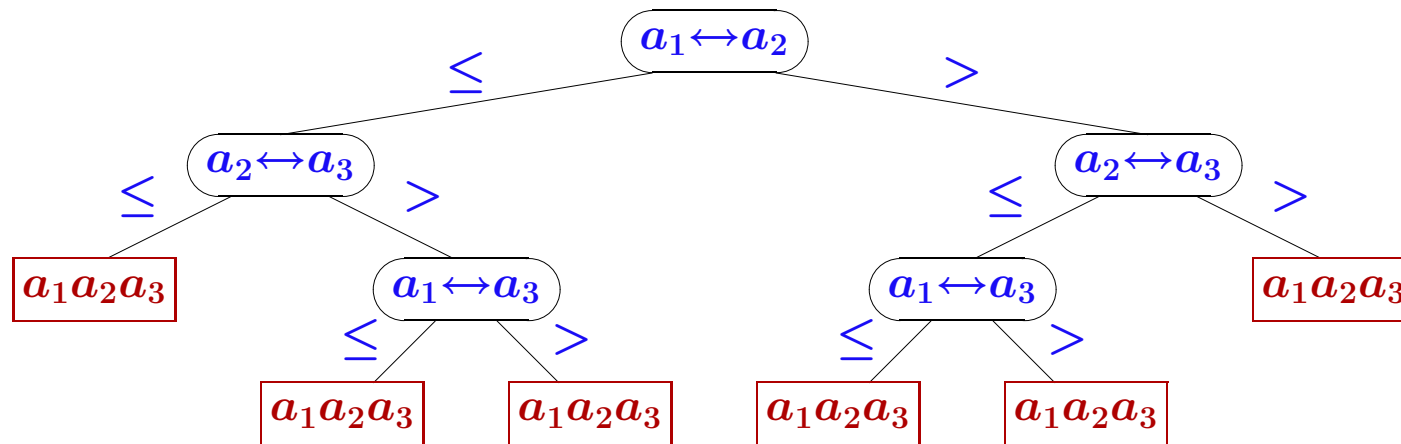
1. Untere Schranken für Komplexität
2. Nichtdeterministische Komplexität
3. Komplexitätsklassen

- **Probleme haben unterschiedlich gute Lösungen**
  - **Suchen**: Lineare Suche  $\mathcal{O}(n)$  — Binärsuche  $\mathcal{O}(\log_2 n)$
  - **Sortieren**: Bubblesort  $\mathcal{O}(n^2)$  — Mergesort  $\mathcal{O}(n \cdot \log_2 n)$
- **Wie effizient kann ein Problem gelöst werden?**
  - Gibt es untere Schranken für die **Komplexität von Lösungen**
- **Wann ist eine Lösung gut genug?**
  - Ist ein Lösungsalgorithmus **optimal** bezüglich Zeit-/Platzbedarf?
- **Nachweis aufwendig**
  - Man muß über **alle möglichen Algorithmen** argumentieren

# KOMPLEXITÄT VON SORTIERVERFAHREN

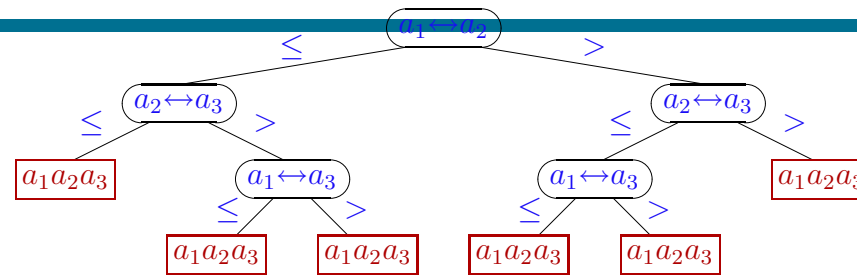
## Schneller als $\mathcal{O}(n \cdot \log_2 n)$ ?

- **Sortiervverfahren müssen Elemente vergleichen**
  - Sonst kann die Anordnung der Elemente nicht garantiert werden
  - Wieviel Vergleiche werden benötigt um  $a_1..a_n$  zu ordnen?
  - Bestimme Anzahl der Vergleiche für den ungünstigsten Fall
- Betrachte **Entscheidungsbaum** von Algorithmen



- Innere Knoten entsprechen den durchgeführten Vergleichen
- Kanten markiert mit Vergleichsergebnis ( $\leq, >$ )
- Blätter sind resultierende Anordnung der Elemente

# KOMPLEXITÄT VON SORTIERVERFAHREN (II)



## ● Algorithmen entsprechen Entscheidungsäumen

- Abarbeitung für konkrete Eingaben entspricht einem Ast im Baum
- Konkrete Laufzeit des Algorithmus entspricht Länge des Astes
- Komplexität des Algorithmus entspricht Tiefe des Entscheidungsbaumes
- ↳ Komplexität von Sortieren  $\equiv$  minimale Tiefe von Entscheidungsäumen

## ● Wie tief ist ein Entscheidungsbaum?

- Jeder Entscheidungsbaum für  $a_1..a_n$  hat  $n!$  Blätter
- Ein binärer Baum der Tiefe  $k$  hat maximal  $2^k$  Blätter
- Jeder Entscheidungsbaum hat mindestens Tiefe  $\log_2 n!$
- $\log_2 n! = \log_2(\prod_{i=1}^n i) = \sum_{i=1}^n \log_2 i \geq \sum_{i=n/2}^n \log_2(n/2) = n/2 * (\log_2 n - 1)$



Sortieren ist in  $\mathcal{O}(n * \log_2 n)$

# KOMPLEXITÄT ANDERER PROBLEMSTELLUNGEN

- **Addition**  $n$ -stelliger Zahlen  $\mathcal{O}(n)$ 
  - Einstellige Addition von rechts nach links mit Übertrag
- **Multiplikation**  $n$ -stelliger Zahlen  $\mathcal{O}(n^2)$ 
  - Jede Stelle muß mit jeder Stelle multipliziert werden
- **Division**  $n$ -stelliger Zahlen  $\mathcal{O}(n^2)$ 
  - Schriftliche Division bestimmt Ergebnis von links nach rechts
- **Berechnung von  $n!$**   $\mathcal{O}(n^2 * (\log_2 n)^2)$ 
  - **Obergrenze:**  $n$ -fache Multiplikation von  $n$  und  $n!$ :  $n * \log_2 n * \log_2(n^n)$
  - **Untergrenze:**  $n/2$ -fach  $n/2 * (n/2)!$ :  $n/2 * \log_2(n/2) * n/4 * (\log_2 n - 2)$
- **Primzahltest** bei  $n$ -stelliger Zahlen  $\mathcal{O}(2^n)$ 
  - Teilbarkeit muß für alle kleineren Zahlen getestet werden
  - Untere Schranke  $\mathcal{O}(2^n)$  **nicht bewiesen**  $\mapsto$  **NP-Vollständigkeit**
  - Ergebnis **gut für offene kryptographische Systeme** (wähle  $n > 200$ )

# WEITERE PROBLEME MIT EXPONENTIELLER KOMPLEXITÄT

- **Travelling Salesman (TSP)**

Gegeben  $n$  Städte, eine Kostentabelle von Kosten  $c_{ij}$  um von Stadt  $i$  nach  $j$  zu reisen und eine Kostenbeschränkung  $B$ . Gibt es eine Rundreise durch alle  $n$  Städte, deren Kosten unter  $B$  liegt?

- **Cliquen-Problem (CLIQUE)**

Gegeben ein Graph  $G = (V, E)$  der Größe  $n$  und eine Zahl  $k \leq n$ . Gibt es in  $G$  eine Clique (vollständig verbundener Teilgraph) der Größe  $k$ ?

- **Erfüllbarkeitsproblem (SAT)**

Ist eine aussagenlogische Formel in KNF der Größe  $n$  erfüllbar?

- **Multiprozessor-Scheduling**

Verteile  $n$  Prozesse derart auf eine Menge von Prozessoren, daß die Ressourcen der Rechner optimal genutzt werden.

- **Binpacking**

Minimiere Anzahl von Verpackungsbehältern, um  $n$  verschieden große Gegenstände zu transportieren.

**Bisher nur durch Testen aller Möglichkeiten lösbar**

## Viele Probleme erscheinen in mehreren Varianten

- **Entscheidungsprobleme**

- Teste ob eine Eingabe  $x_1, \dots, x_n$  eine bestimmte Eigenschaft  $P$  erfüllt
- Suchen in Listen, Primzahltests, Travelling Salesman, Clique ...

- **Berechnungsprobleme**

- Bei Eingabe von  $x_1, \dots, x_n$  berechne ein  $y$ , so daß  $P(x_1, \dots, x_n, y)$  gilt
- Sortieren, Primfaktorzerlegung, Matrixmultiplikation, ...

- **Optimierungsprobleme**

- Bei Eingabe von  $x_1, \dots, x_n$  berechne das beste  $y$  mit  $P(x_1, \dots, x_n, y)$
- Travelling Salesman, Clique, Binpacking, Multiprocessor Scheduling

# VARIANTEN DES CLIQUENPROBLEMS

Gegeben ein Graph  $G = (V, E)$  und eine Zahl  $k \leq |V|$ . Eine  $k$ -Clique von  $G$  ist ein vollständig verbundener Teilgraph  $C = (V_c, E_c)$  mit  $|V_c| = k$

## ● Entscheidungsproblem

- “Gibt es eine Lösung mit einem bestimmten Wert”
- **CLIQUE**: Gibt es in  $G$  eine Clique der Größe  $k$ ?

## ● Berechnungsproblem

- Bestimme eine konkrete Lösung mit einem bestimmten Wert
- **CLIQUE<sub>2</sub>**: Bestimme eine Clique  $C \subseteq G$  der Größe  $k$

## ● Optimierungsprobleme

- Bestimme den Wert einer optimalen Lösung
- **CLIQUE<sub>opt</sub>**: Bestimme das größte  $k$ , so daß  $G$  eine  $k$ -Clique enthält
- Berechne die optimale Lösung
- **CLIQUE<sub>opt2</sub>**: Bestimme eine Clique  $C \subseteq G$  mit maximaler Größe  $k$

## PROBLEM VARIANTEN VON CLIQUE SIND GLEICH SCHWER

- Löse **CLIQUE<sub>opt</sub>** mit **CLIQUE**
  - Beginne mit  $k := |V|$  und teste ob es in  $G$  eine  $k$ -Clique gibt
  - Reduziere  $k$  bis der Test erfolgreich ist und gebe  $k$  aus
  - Zusatzaufwand ist linear in  $|V|$
- Löse **CLIQUE<sub>opt2</sub>** mit **CLIQUE<sub>opt</sub>**
  - Bestimme  $k_{opt}$  für  $G$  und beginne mit  $E_c := E$
  - Wähle Kante  $e \in E$  und teste ob es in  $(V, E_c - \{e\})$  eine  $k_{opt}$ -Clique gibt
  - Ist dies der Fall, so setze  $E_c := E_c - \{e\}$
  - Wiederhole dies iterativ für alle Kanten aus  $E$
  - Die resultierende Menge  $E_c$  und die zugehörigen Ecken bilden die  $k_{opt}$ -Clique
  - Zusatzaufwand ist linear in  $|E|$

Löse analog **CLIQUE<sub>2</sub>** mit **CLIQUE**

Die Umkehrungen sind trivial



Es reicht Entscheidungsprobleme zu analysieren

# VIELE SCHWERE PROBLEME HABEN LEICHTE ERFOLGSTESTS

- **Travelling Salesman:** Für eine gegebene Rundreise  $i_1..i_n$  können die Kosten  $c_{i_1i_2} + \dots c_{i_ni_1}$  in linearer Zeit berechnet und mit der Kostenbeschränkung  $B$  verglichen werden
- **Cliquen-Problem:** Ein gegebener Teilgraph der Größe  $k$  kann in polynomieller Zeit auf Vollständigkeit überprüft werden
- **Erfüllbarkeitsproblem:** Man kann in polynomieller Zeit testen, ob eine gegebene Belegung der Variablen eine Formel erfüllt
- **Binpacking:** Man kann in polynomieller Zeit testen, ob eine gegebene Verteilung der Gegenstände in  $k$  Verpackungsbehälter paßt
- **Zusammengesetztheitstest:** Man kann in quadratischer Zeit testen, ob eine gegebene Zahl Teiler von  $x$  (also  $x$  keine Primzahl) ist

**Nichtdeterministische Maschinen liefern polynomielle Lösung**

# NICHTDETERMINISTISCHE LÖSBARKEIT

## ● Nichtdeterministische Turingmaschine $\tau$

- Komponenten  $S, X, \Gamma, s_0, b$  wie bei normaler Turingmaschine
- Zustandsüberföhrungsfunktion  $\delta: S \times \Gamma \rightarrow 2^{S \times \Gamma \times \{r, l, h\}}$
- Nachfolgekonfigurationsfunktion  $\hat{\delta}: K_\tau \rightarrow 2^{K_\tau}$
- Semantik  $h_\tau: X^* \rightarrow 2^{\Gamma^*}$   
 $h_\tau(w) = \perp$ , wenn  $\hat{\delta}$  unendliche Konfigurationsfolgen ermöglicht
- **Keine Erweiterung des Berechenbarkeitsbegriffs**

## ● Nichtdeterministische Entscheidbarkeit

- $\tau$  entscheidet  $M \subseteq X^*$ , falls  $w \in M \Leftrightarrow 1 \in h_\tau(w)$   
“Es gibt eine akzeptierende Berechnung für  $w$ ”
- Rechenzeit  $nt_\tau(w) = \begin{cases} \text{minimale Länge einer akzeptierenden Berechnung für } w & \text{falls } w \in M \\ 0 & \text{sonst} \end{cases}$
- Zeitkomplexität  $ntime_\tau(n) = \max\{nt_\tau(w) \mid |w|=n\}$
- Platzkomplexität  $nspace_\tau(n)$  analog definiert

## ● Deterministische Turingmaschine mit **Orakel**

- Eingabe des Wortes  $w$  auf erstem Arbeitsband
- **Phase 1**: Orakel generiert Wort  $w'$  auf zweitem Arbeitsband
- **Phase 2**:  $\tau$  verarbeitet  $w$  und  $w'$  deterministisch
- $\tau$  entscheidet  $M \subseteq X^*$ , falls  $\forall w, w'. h_\tau(w, w') \neq \perp$   
und  $w \in M \Leftrightarrow \exists w'. h_\tau(w, w') = 1$
- Rechenzeit  $ot_\tau(w) = \min\{t_\tau(w, w') \mid h_\tau(w, w') = 1\}$  (0 falls  $w \notin M$ )  
Orakel benötigt keine Rechenzeit zum Schreiben
- Zeitkomplexität  $otime_\tau(n) = \max\{ot_\tau(w) \mid |w|=n\}$

## ● NTM's und OTM's sind äquivalent

- OTM kann Berechnungsfolge der NTM raten und deterministisch ausführen
- NTM kann alle Berechnungsfolgen der OTM “parallel” ausführen
- **Rechenzeit ist identisch**

# KOMPLEXITÄT VON (ENTSCHEIDUNGS-)PROBLEMEN

## ● Zeitkomplexität

- Eine Menge  $M$  hat **Zeitkomplexität**  $\mathcal{O}(f)$ , falls es eine DTM  $\tau$  mit  $time_{\tau} \in \mathcal{O}(f)$  gibt, die  $M$  entscheidet.
- **DTIME**( $f$ ) =  $\{M \mid M \text{ hat Zeitkomplexität } \mathcal{O}(f)\}$
- Eine Menge  $M$  hat **nichtdeterministische Zeitkomplexität**  $\mathcal{O}(f)$ , falls es eine NTM  $\tau$  mit  $ntime_{\tau} \in \mathcal{O}(f)$  gibt, die  $M$  entscheidet.
- **NTIME**( $f$ ) =  $\{M \mid M \text{ hat nichtdeterministische Zeitkomplexität } \mathcal{O}(f)\}$

## ● Platzkomplexität

- Eine Menge  $M$  hat **Platzkomplexität**  $\mathcal{O}(f)$ , falls es eine DTM  $\tau$  mit  $space_{\tau} \in \mathcal{O}(f)$  gibt, die  $M$  entscheidet.
- **DSPACE**( $f$ ) =  $\{M \mid M \text{ hat Platzkomplexität } \mathcal{O}(f)\}$
- Eine Menge  $M$  hat **nichtdeterministische Platzkomplexität**  $\mathcal{O}(f)$ , falls es eine NTM  $\tau$  mit  $nspace_{\tau} \in \mathcal{O}(f)$  gibt, die  $M$  entscheidet.
- **NSPACE**( $f$ ) =  $\{M \mid M \text{ hat nichtdeterministische Platzkomplexität } \mathcal{O}(f)\}$

**Konkrete Analysen werden mit abstrakten deterministischen oder nichtdeterministischen Algorithmen durchgeführt**

# WICHTIGE KOMPLEXITÄTSKLASSEN

- $\mathcal{P} = \bigcup_k DTIME(n^k)$  : **Effiziente Lösbarkeit**
  - Menge der in polynomieller Zeit lösbaren Probleme
- $\mathcal{NP} = \bigcup_k NTIME(n^k)$  :
  - Menge der nichtdeterministisch in polynomieller Zeit lösbaren Probleme
- **Weitere Zeitkomplexitätsklassen**
  - $EXPTIME = \bigcup_k DTIME(2^{n^k})$
  - $NEXPTIME = \bigcup_k NTIME(2^{n^k})$
  - $LOGTIME = DTIME(\log_2 n)$
  - $NLOGTIME = NTIME(\log_2 n)$
- **Platzkomplexitätsklassen**
  - $LOGSPACE = DSPACE(\log_2 n)$
  - $NLOGSPACE = NSPACE(\log_2 n)$
  - $PSPACE = \bigcup_k DSPACE(n^k)$
  - $NPSPACE = \bigcup_k NSPACE(n^k)$
  - $EXPSPACE = \bigcup_k DSPACE(2^{n^k})$

# BEISPIELE FÜR KOMPLEXITÄTSKLASSEN

- $\mathcal{P}$ :

- Arithmetische Operationen, Sortieren, Matrixmultiplikation, ...

- $\mathcal{NP}$ :

- Travelling Salesman, Cliques-Problem, Erfüllbarkeitsproblem
- Multiprozessor-Scheduling, Binpacking, Zusammengesetztheitstest

- ***EXPTIME***:

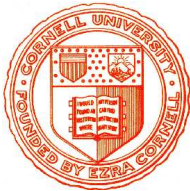
- Travelling Salesman, Cliques-Problem, Erfüllbarkeitsproblem
- Multiprozessor-Scheduling, Binpacking, Primzahltest

- **Komplexitätsklassenhierarchie**

$$\begin{aligned} LOGTIME &\subseteq NLOGTIME \subseteq LOGSPACE \subseteq NLOGSPACE \\ &\subseteq \mathcal{P} \subseteq \mathcal{NP} \subseteq PSPACE \subseteq NPSPACE \\ &\subseteq EXPTIME \subseteq NEXPTIME \subseteq EXPSPACE \subseteq \dots \end{aligned}$$

- Es wird vermutet, daß alle Inklusionen echt sind

# Theoretische Informatik



## Einheit 4.4

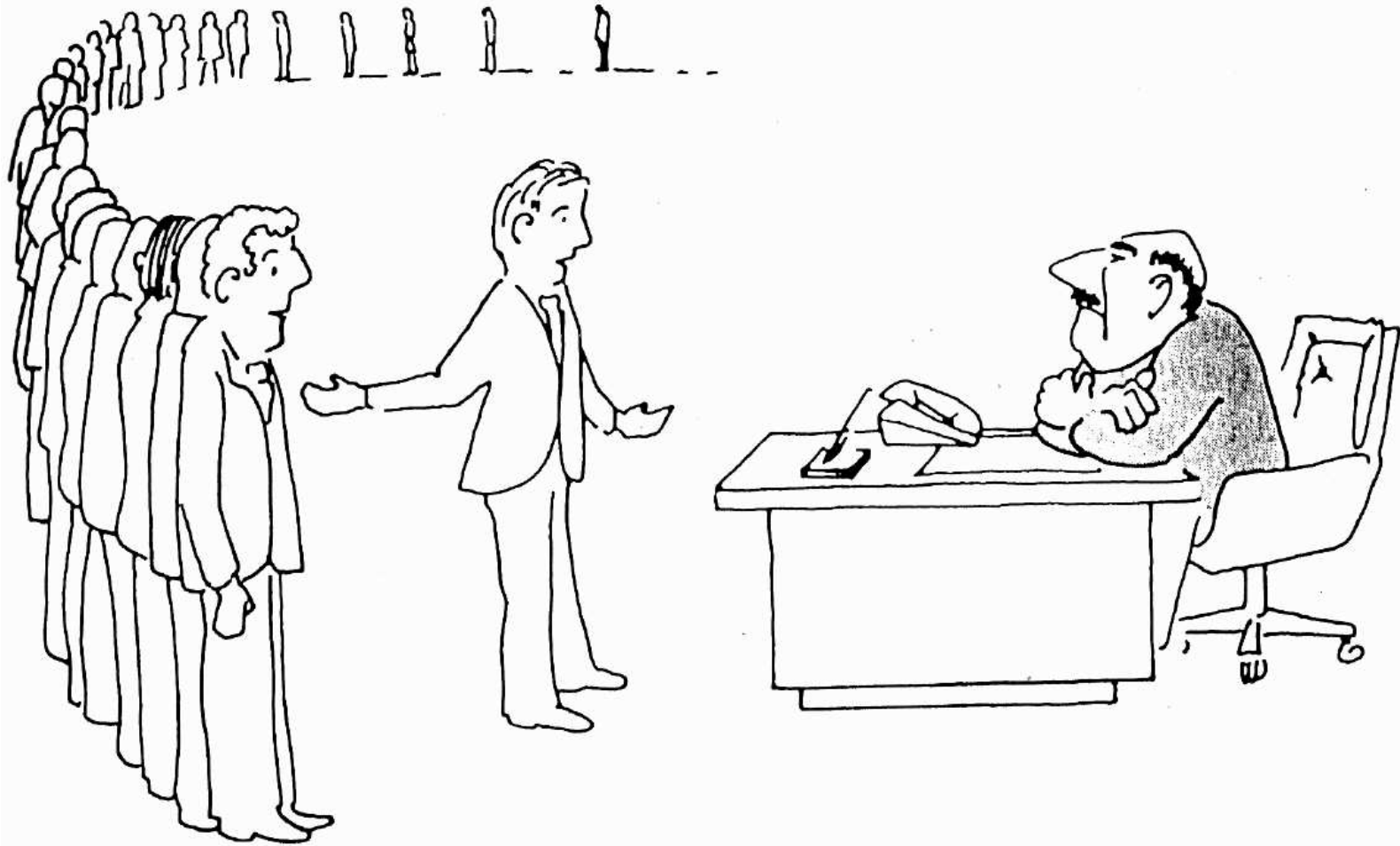
### NP-Vollständigkeit



1. Reduzierbarkeit und Vollständigkeit von Klassen
2. Der Satz von Cook
3. NP-vollständige Probleme

# DAS $\mathcal{P}$ - $\mathcal{NP}$ PROBLEM

WAS TUN, WENN EIN PROBLEM NICHT EFFEKTIV LÖSBAR ZU SEIN SCHEINT?



“I can’t find an efficient algorithm, but neither can all these famous people.”

**Vielleicht der einzig mögliche Weg**

# DAS $\mathcal{P}$ - $\mathcal{NP}$ PROBLEM

Gilt  $\mathcal{P}=\mathcal{NP}$  oder  $\mathcal{P}\neq\mathcal{NP}$  ?

- **Eines der wichtigsten offenen Probleme der TI**
  - Sind nichtdeterministisch lösbare Probleme effizient lösbar?
  - Seit mehr als 30 Jahren ungeklärt, möglicherweise unlösbar
- **Mehr als 1000 algorithmische Probleme betroffen**
  - Suchprobleme (Travelling Salesman, ...)
  - Reihenfolgenprobleme (Scheduling, Binpacking, ...)
  - Graphenprobleme (Clique, Vertex cover, ...)  $\mapsto$  Operations Research
  - Logische Probleme (Erfüllbarkeit, ...)  $\mapsto$  Model Checking, Hardwareverifikation
  - Zahlenprobleme (Primzahltest, ...)  $\mapsto$  Kryptographie, IT Sicherheit
- **Indizien sprechen gegen  $\mathcal{P}=\mathcal{NP}$** 
  - Zu viele  $\mathcal{NP}$ -Probleme ohne bekannte polynomielle Lösung
  - Mehr als 1000 äquivalente Probleme in der ‘schwersten Teilklasse’ von  $\mathcal{NP}$

# WIE ANALYSIERT MAN DIE FRAGE “ $\mathcal{P}=\mathcal{NP}$ ODER $\mathcal{P}\neq\mathcal{NP}$ ”?

- **Untersuche die “schwierigsten”  $\mathcal{NP}$ -Probleme**
  - Kann man eines davon effizient lösen?
  - Wenn **ja**, dann gilt  $\mathcal{P}=\mathcal{NP}$
  - Wenn **nein**, dann gibt es ein Beispiel für  $\mathcal{P}\neq\mathcal{NP}$
- **Was heißt “ $M$  ist schwierigstes  $\mathcal{NP}$ -Problem”?**
  - Jedes andere  $\mathcal{NP}$ -Problem  $M'$  ist leichter als  $M$
  - Lösungen für  $M'$  können in Lösungen für  $M$  umgewandelt werden
  - Transformation der Lösung ist effizient



**Polynomielle Reduktion**

- **$M \subseteq X^*$  polynomiell reduzierbar auf  $M' \subseteq Y^*$** 
  - Es gibt eine in polynomieller Zeit berechenbare totale Funktion  $f: X^* \rightarrow Y^*$  mit  $M = f^{-1}(M')$
  - Schreibweise:  **$M \leq_p M'$**
- Reduzierbarkeit  $\equiv$  geringere Komplexität
  - $M \leq_p M' \wedge M' \in \mathcal{P} \Rightarrow M \in \mathcal{P}$
  - $M \leq_p M' \wedge M' \in \mathcal{NP} \Rightarrow M \in \mathcal{NP}$

**Beweis:**

- $\chi_M(x)=1 \Leftrightarrow x \in M \Leftrightarrow f(x) \in M' \Leftrightarrow \chi_{M'}(f(x))=1 \Leftrightarrow (\chi_{M'} \circ f)(x)=1$
- $\chi_{M'} \circ f$  ist in polynomieller Zeit berechenbar, wenn dies für  $\chi_{M'}$  gilt

# WICHTIGE GRAPHENTHEORETISCHE DEFINITIONEN

- Ein (ungerichteter) *Graph* ist ein Paar  $G = (V, E)$ , wobei  $V$  endliche Menge und  $E \subseteq \{ \{v, v'\} \mid v, v' \in V \wedge v \neq v' \}$ .  
Ein Graph ist darstellbar als Liste  $v_1, \dots, v_n, \{v_{i_1}, v'_{i_1}\}, \dots, \{v_{i_m}, v'_{i_m}\}$ .
- Ein Graph  $H = (V_H, E_H)$  ist genau dann *Subgraph* des Graphen  $G = (V, E)$  ( $H \sqsubseteq G$ ), wenn alle Ecken und Kanten von  $H$  auch Ecken bzw. Kanten in  $G$  sind:  
 $(V_H, E_H) \sqsubseteq (V, E) : \Leftrightarrow V_H \subseteq V \wedge E_H \subseteq E$
- $H = (V_H, E_H)$  ist *isomorph* zu  $G = (V, E)$  (kurz:  $H \cong G$ ), wenn die Graphen durch Umbenennung (bijektive Abbildung  $h : V \rightarrow V_H$ ) ineinander überführt werden können:  
 $(V_H, E_H) \cong (V, E) : \Leftrightarrow \exists h : V \rightarrow V_H, (h \text{ bijektiv} \wedge E_H = \{ \{h(u), h(v)\} \mid \{u, v\} \in E \})$
- Die *Größe*  $|G|$  eines Graphen  $G = (V, E)$  ist die Anzahl  $|E|$  seiner Kanten.
- Der *Komplementärgraph* des Graphen  $G = (V, E)$  ist der Graph  $G^c = (V, E^c)$  mit  $E^c = \{ \{v, v'\} \mid v, v' \in V \} - E$ .
- Eine *Clique* der Größe  $k$  im Graphen  $G = (V, E)$  ist eine vollständig verbundene Knotenmenge  $V' \subseteq V$  mit  $|V'| = k$ . (Dabei heißt *vollständig verbunden*:  $\forall v, v' \in V'. v \neq v' \Rightarrow \{v, v'\} \in E$ )
- Eine *Knotenüberdeckung* (Vertex cover) des Graphen  $G = (V, E)$  ist eine Knotenmenge  $V' \subseteq V$  mit der Eigenschaft  $\forall \{v, v'\} \in E. v \in V' \vee v' \in V'$
- Ein *Hamilton'scher Kreis* im Graphen  $G = (V, E)$  ist ein Kreis, der nur aus Kanten aus  $E$  besteht und jeden Knoten genau einmal berührt.  
(D.h. eine Permutation  $\pi : \{1..n\} \rightarrow \{1..n\}$  mit  $\forall i < n. \{v_{\pi(i)}, v_{\pi(i+1)}\} \in E \wedge \{v_{\pi(n)}, v_{\pi(1)}\} \in E$ )
- Ein *gerichteter Graph* ist ein Paar  $G = (V, E)$ , wobei  $V$  endliche Menge und  $E \subseteq V \times V$ .  
Ein *Hamilton'scher Kreis* im gerichteten Graphen  $G = (V, E)$  ist ein gerichteter Kreis, der nur aus Kanten aus  $E$  besteht und jeden Knoten genau einmal berührt.

## ● Cliquen Problem

- Gegeben ein Graph  $G = (V, E)$  der Größe  $n$  und eine Zahl  $k \leq n$ .
- Gibt es in  $G$  eine Clique der Größe  $k$ ?

$$\textbf{CLIQUE} = \{ (G, k) \mid G = (V, E) \text{ Graph} \wedge (\exists V_c \subseteq V. |V_c| \geq k \\ \wedge V_c \text{ is Clique in } G) \}$$

## ● Vertex Cover Problem

- Gegeben ein Graph  $G = (V, E)$  der Größe  $n$  und eine Zahl  $k \leq n$ .
- Gibt es eine Teilmenge  $V' \subseteq V$  mit höchstens  $k$  Elementen,  
so daß aus jeder Kante in  $G$  mindestens eine Ecke in  $V'$  liegt?

$$\textbf{VC} = \{ (G, k) \mid G \text{ Graph} \wedge (\exists V' \subseteq V. |V'| \leq k \\ \wedge V' \text{ ist Knotenüberdeckung von } G) \}$$

# REDUZIERBARKEIT: $CLIQUE \leq_p VERTEX COVER$

## ● Analyse der Eigenschaften

$V'$  Knotenüberdeckung von  $G$

$$\Leftrightarrow \forall \{v, v'\} \in E. v \in V' \vee v' \in V'$$

$$\Leftrightarrow \forall \{v, v'\} \in E. \{v, v'\} \cap V' \neq \emptyset$$

$$\Leftrightarrow \forall v, v' \in V - V'. \{v, v'\} \notin E$$

$$\Leftrightarrow \forall v, v' \in V - V'. v \neq v' \Rightarrow \{v, v'\} \in E^c := \{ \{v, v'\} \subseteq V \mid v \neq v' \} - E$$

$$\Leftrightarrow V - V' \text{ ist Clique im Komplementgraphen } G^c = (V, E^c)$$

## ● Transformation der Probleme (Vertausche $G$ und $G^c$ )

$(G, k) \in CLIQUE$

$$\Leftrightarrow G \text{ hat Clique } V_c \text{ der Mindestgröße } k$$

$$\Leftrightarrow G^c \text{ hat Knotenüberdeckung } V' = V - V_c \text{ der Maximalgröße } |V| - k$$

$$\Leftrightarrow (G^c, |V| - k) \in VC$$

Wähle  $f(G, k) := (G^c, |V| - k)$

$f$  in polynomieller Zeit berechenbar und  $CLIQUE = f^{-1}(VC)$  ✓

- **$\mathcal{NP}$ -hart**: nicht leichter als  $\mathcal{NP}$ 
  - $M' \subseteq X^*$  ist  **$\mathcal{NP}$ -hart**, wenn  $M \leq_p M'$  für alle  $M \in \mathcal{NP}$
- **$\mathcal{NP}$ -vollständig**: das Schwierigste in  $\mathcal{NP}$ 
  - $M \subseteq X^*$  ist  **$\mathcal{NP}$ -vollständig**, wenn  $M \in \mathcal{NP}$  und  $M$   $\mathcal{NP}$ -hart
  - Schreibweise:  **$M \in \mathcal{NPC}$**

# KONSEQUENZEN VON $\mathcal{NP}$ -VOLLSTÄNDIGKEIT

- Alle  $\mathcal{NP}$ -vollständigen Probleme sind äquivalent
  - $M, M' \in \mathcal{NPC} \Rightarrow M' \leq_p M \wedge M \leq_p M'$
- $\mathcal{NP}$ -Vollständigkeit ist leicht nachweisbar, wenn ein  $\mathcal{NP}$ -vollständiges Problem bekannt ist
  - $M \in \mathcal{NPC} \Leftrightarrow M \in \mathcal{NP} \wedge \exists M' \in \mathcal{NPC}. M' \leq_p M$
  - $M \in \mathcal{NPC} \Leftrightarrow \exists M' \in \mathcal{NPC}. M' \leq_p M \wedge M \leq_p M'$
- $\mathcal{NP}$ -vollständige Probleme entscheiden “ $\mathcal{P} = \mathcal{NP}$ ”
  - $\mathcal{P} = \mathcal{NP} \Leftrightarrow \exists M \in \mathcal{NPC}. M \in \mathcal{P}$
  - Ist  $\mathcal{P} = \mathcal{NP}$  dann sind alle  $\mathcal{NP}$ -vollständigen Probleme in  $\mathcal{P}$
  - $\mathcal{P} \neq \mathcal{NP} \Leftrightarrow \exists M \in \mathcal{NPC}. M \notin \mathcal{P}$
  - Ist  $\mathcal{P} \neq \mathcal{NP}$  dann sind alle  $\mathcal{NP}$ -vollständigen Probleme nicht in  $\mathcal{P}$

## WIE ZEIGT MAN $\mathcal{NP}$ -VOLLSTÄNDIGKEIT?

- **Codiere Berechnung einer NTM in einem Problem  $L$** 
  - Wenn Berechnung positiv ausfällt, soll Problem lösbar sein
  - Wenn Berechnung negativ ausfällt, soll Problem unlösbar sein
  - Ergibt  $M \leq_p L$  für das von der NTM entschiedene Problem  $M$
- **Problem  $L$  muß **alle** polynomiellen NTMs codieren**
  - Ergibt  $M \leq_p L$  für jedes  $M \in \mathcal{NP}$
- **Welches Problem ist ausdrucksstark genug?**
  - Codiere mögliche Zustandsübergänge durch logische Formeln
  - Problem: Können Zustandsübergänge so kombiniert werden,  
daß Berechnung mit Ergebnis 1 codiert wird?
  - Erfüllbarkeitsproblem der (Aussagen-)logik ist Kandidat für  $\mathcal{NPC}$

# DAS ERFÜLLBARKEITSPROBLEM

Ist eine aussagenlogische Formel in KNF erfüllbar?

Gegeben  $m$  Klauseln  $k_1, \dots, k_m$  über  $n$  Variablen  $x_1, \dots, x_n$ .

Gibt es eine Belegung  $a_1, \dots, a_n \in \{0, 1\}$

der Variablen  $x_1, \dots, x_n$ , welche alle Klauseln erfüllt?

- **Klausel** über den Variablen  $x_1, \dots, x_n$ 
  - Disjunktion einiger **Literale** der Form  $x_i$  bzw.  $\neg x_i$
- **Belegung**  $a_1, \dots, a_n \in \{0, 1\}$  **erfüllt** Klausel  $k_j$ 
  - Auswertung von  $k_j$  unter  $a_1, \dots, a_n$  ergibt den Boole'schen Wert 1
- **SAT** =  $\{k_1, \dots, k_m \mid k_i \text{ Klausel über } x_1, \dots, x_n$   
 $\wedge \exists a_1, \dots, a_n \in \{0, 1\}. \forall j \leq m. a_1, \dots, a_n \text{ erfüllt } k_j\}$

Codierbar als Teilmenge der Sprache der Aussagenlogik

## BEISPIELE VON FORMELN IN KNF

$$(\underline{x_1} \vee x_2) \wedge (x_1 \vee \underline{x_2} \vee \underline{x_3}) \wedge \underline{x_3} \quad \text{erfüllbar}$$

– Setze  $x_3=0$ ,  $x_2=1$ ,  $x_1$  beliebig, z.B.  $x_1=0$

– Auswertung:  $(\underline{0}+1) * (0+\underline{1}+\underline{0}) * \underline{0} = (1+1) * (0+0+1) * 1 = 1 * 1 * 1 = 1$

$$x_1 \wedge \underline{x_1} \quad \text{nicht erfüllbar}$$

– Jede Belegung ergibt den Wert 0

$$(x_1 \vee x_2) \wedge (\underline{x_1} \vee \underline{x_2}) \quad \text{erfüllbar, Belegung: } (1,0)$$

$$(x_1 \vee x_2) \wedge (x_1 \vee \underline{x_2}) \wedge (\underline{x_1} \vee x_2) \wedge (\underline{x_1} \vee \underline{x_2}) \quad \text{nicht erfüllbar}$$

$$(x_1 \vee \underline{x_2} \vee x_3) \wedge (\underline{x_1} \vee x_2 \vee \underline{x_4}) \wedge (\underline{x_1} \vee \underline{x_2} \vee \underline{x_3}) \quad \text{erfüllbar, Belegung: } (1,1,0,0)$$

# LÖSUNGsalgorithmen für das Erfüllbarkeitsproblem

$$SAT = \{k_1..k_m \mid k_i \text{ Klausel über } x_1..x_n \wedge \exists a_1..a_n \in \{0,1\}. \forall j \leq m. a_1..a_n \text{ erfüllt } k_j\}$$

## ● Deterministisch

- Werte Klauseln für alle möglichen Belegungen der Variablen aus bis erfüllende Belegung gefunden ist
- Es gibt  $2^n$  möglichen Belegungen von  $x_1, ..x_n$
- Auswertung linear in Größe der Formel  $\mathcal{O}(m * n)$
- Laufzeit ist in  $\mathcal{O}(2^n)$

## ● Nichtdeterministisch

- Rate eine erfüllende Belegung der Variablen (falls es eine gibt)
- Prüfe Belegung durch Auswertung der Formel
- Polynomielle Laufzeit



$$SAT \in \mathcal{NP}$$

# DER SATZ VON COOK

## *SAT* ist $\mathcal{NP}$ -vollständig

- **Gegeben:** NTM  $\tau$ , die ein Problem in polynomieller Zeit löst
- **Ziel:** Codiere Berechnung von  $\tau$  bei Eingabe  $w$  durch Formel in KNF, die genau dann erfüllbar ist, wenn  $h_\tau(w) = 1$ 
  - Codierung muß in polynomieller Zeit (relativ zu  $|w|$ ) berechenbar sein
- **Vorgehen:** Beschreibe mögliche Konfigurationsübergänge von  $\tau$  durch Klauseln
  - Codiere Zustand, Kopfposition und Bandzellen durch Literale
  - Es werden nur polynomiell viele Literale und Klauseln benötigt
  - Formel ist erfüllbar, wenn Konfigurationsübergänge zu Berechnung zusammengesetzt werden können

- Zeige  $L \in \mathcal{NP}$ :

- Beschreibe, welchen Lösungsvorschlag das Orakel generiert
- Beschreibe, wie Lösungsvorschlag überprüft wird
- Zeige, daß das Prüfverfahren polynomiell ist

- Zeige  $\exists L' \in \mathcal{NPC}. L' \leq_p L$  (anstatt  $\forall L' \in \mathcal{NP}. L' \leq_p L$ )

- Wähle ein ähnliches, bekanntes Problem  $L' \in \mathcal{NPC}$
- Beschreibe Transformationsfunktion  $f$ , welche Eingaben aus der Sprache für  $L'$  in Worte der Sprache für  $L$  umwandelt
- Zeige für alle  $x$ :  $x \in L' \Leftrightarrow f(x) \in L$  (also  $L' = f^{-1}(L)$ )
- Zeige, daß  $f$  in polynomieller Zeit berechnet werden kann

# ERFÜLLBARKEITSPROBLEM MIT 3 LITERALEN PRO KLAUSEL

## 3SAT

$$= \{k_1, \dots, k_m \mid \forall i \leq m. \exists z_{ij} \in \{x_1, \underline{x_1}, \dots, x_n, \underline{x_n}\}. k_i = z_{i1} \vee z_{i2} \vee z_{i3} \\ \wedge \exists a_1, \dots, a_n \in \{0,1\}. \forall j \leq m. a_1, \dots, a_n \text{ erfüllt } k_j \}$$

## ● 3SAT $\in \mathcal{NP}$

– Wie  $SAT \in \mathcal{NP}$ : Rate Belegung der Variablen und werte Klauseln aus

## ● SAT $\leq_p$ 3SAT:

– **Normalisierung** der Klauseln  $k_1, \dots, k_m$  über  $x_1, \dots, x_n$ .

Ersetze Klausel  $k_i$  durch äquivalente Menge von Dreierklauseln

• Ersetze einelementige Klauseln  $k_i = z$  durch  $z \vee z \vee z$

• Ersetze zweielementige Klauseln  $k_i = z \vee z'$  durch  $z \vee z \vee z'$

• Übernahme dreielementige Klauseln unverändert

• Ersetze Klauseln  $k_i = z_1 \vee z_2 \dots \vee z_j$  durch  $j-2$  neue Klauseln mit neuen Variablen  $y_{i,l}$ :  $(z_1 \vee z_2 \vee y_{i,1}) \wedge (\underline{y_{i,1}} \vee z_3 \vee y_{i,2}) \wedge \dots (\underline{y_{i,j-3}} \vee z_{j-1} \vee z_j)$

– Normalisierung der Klauseln möglich in **polynomieller Zeit**

–  $k_i$  erfüllbar genau dann wenn normalisierte Klauselmenge erfüllbar

– Für die Transformation  $f$  gilt:  $\forall F. F \in SAT \Leftrightarrow f(F) \in 3SAT$

# DAS CLIQUEN PROBLEM IST $\mathcal{NP}$ -VOLLSTÄNDIG

$$\textbf{CLIQUE} = \{ (G, k) \mid G = (V, E) \text{ Graph} \wedge (\exists V_c \subseteq V. |V_c| \geq k \wedge V_c \text{ is Clique in } G) \}$$

## ● $\textbf{CLIQUE} \in \mathcal{NP}$ :

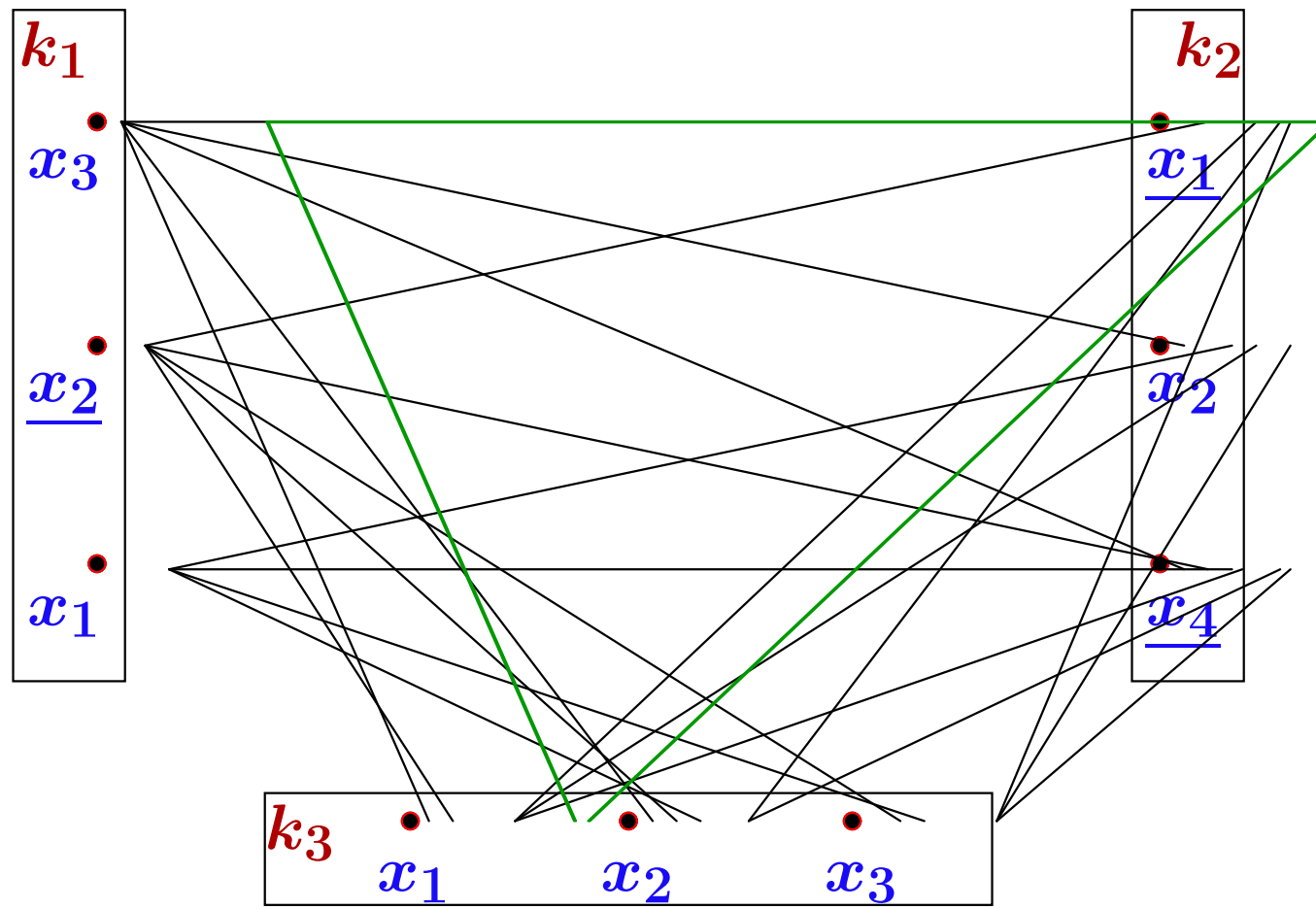
- Rate eine Kantenmenge  $V_c \subseteq V$
- Prüfe  $|V_c| \geq k$  *maximal  $|V_c|$  Schritte*
- Prüfe:  $\forall v \neq v' \in V_c. \{v, v'\} \in E$  *maximal  $|V_c|^2 * |E| \leq |V|^4$  Schritte*

## ● $3\text{SAT} \leq_p \textbf{CLIQUE}$

- Gegeben  $F = (k_1, \dots, k_m)$  mit  $k_i = z_{i1} \vee z_{i2} \vee z_{i3}$  und  $z_{ij} \in \{x_1, \dots, \underline{x_n}\}$
- Konstruiere Graphen  $G_F := (V, E)$  mit  
 $V := \{v_{ij} \mid 1 \leq i \leq m, 1 \leq j \leq 3\}$  und  $E := \{ \{v_{ij}, v_{i'j'}\} \mid i \neq i' \wedge z_{ij} \neq \underline{z_{i'j'}} \}$
- Setze  $f(F) := (G_F, m)$
- $f$  ist in polynomieller Zeit berechenbar

# CODIERUNG EINER FORMEL ALS CLIQUENPROBLEM

$F = (k_1, k_2, k_3)$  mit  $k_1 = x_1 \vee \underline{x_2} \vee x_3$   $k_2 = \underline{x_1} \vee x_2 \vee \underline{x_4}$   $k_3 = \underline{x_1} \vee \underline{x_2} \vee \underline{x_3}$



Gibt es in dem Graphen eine 3-Clique?

# KORREKTHEIT DER TRANSFORMATION

Gegeben  $F = (k_1, \dots, k_m)$  mit  $k_i = z_{i1} \vee z_{i2} \vee z_{i3}$  und  $z_{ij} \in \{x_1, \dots, x_n\}$

Setze  $f(F) := (G_F, m)$  mit  $G_F := (V, E)$ , wobei

$V := \{v_{ij} \mid 1 \leq i \leq m, 1 \leq j \leq 3\}$  und  $E := \{ \{v_{ij}, v_{i'j'}\} \mid i \neq i' \wedge z_{ij} \neq \underline{z_{i'j'}} \}$

Es sei  $f(F) \in \text{CLIQUE}$

Dann hat  $G_F$  eine  $m$ -Clique  $V_c$

Per Konstruktion von  $E$  enthält  $V_c$  aus jedem der Blöcke  $b_i := \{v_{ij} \mid 1 \leq j \leq 3\}$  genau einen Knoten und keine zwei Knoten in  $V_c$  sind komplementär ( $z_{ij} \neq \underline{z_{i'j'}}$ )

Eine Belegung der zugehörigen  $z_{ij}$  mit 1 erfüllt alle Klauseln  $k_i$  von  $F$

Also gilt  $F \in 3SAT$

Gilt umgekehrt  $F \in 3SAT$ , so gibt es eine erfüllende Belegung der  $z_{ij}$

Wähle aus jeder Klausel  $k_i$  ein Literal mit dem Wert 1

Die zugehörigen Knoten bilden eine  $m$ -Clique in  $G_F$

Also gilt  $f(F) \in \text{CLIQUE}$



$$3SAT \leq_p \text{CLIQUE}$$

# VERTEX COVER PROBLEM IST $\mathcal{NP}$ -VOLLSTÄNDIG

$VC = \{ (G, k) \mid G \text{ Graph} \wedge (\exists V' \subseteq V. |V'| \leq k \wedge V' \text{ ist Knotenüberdeckung von } G) \}$

- $VC \in \mathcal{NP}$ :

- Rate eine Kantenmenge  $V' \subseteq V$
- Prüfe  $|V'| \leq k$
- Prüfe:  $\forall \{v, v'\} \in E. v \in V' \vee v' \in V'$

*maximal  $|V'|$  Schritte*

*maximal  $|V'| * |E| \leq |V|^3$  Schritte*

- $CLIQUE \leq_p VC$

- Bereits beweisen

# WEITERE $\mathcal{NP}$ -VOLLSTÄNDIGE GRAPHENPROBLEME

## ● Independent Set

$CLIQUE \leq_p IS$

- Gegeben ein Graph  $G = (V, E)$  der Größe  $n$  und eine Zahl  $k \leq n$ .
- Gibt es in  $G$  eine unabhängige Knotenmenge der Größe  $k$ ?

$$IS = \{ (G, k) \mid G = (V, E) \text{ Graph} \\ \wedge \exists V_i \subseteq V. |V_i| \geq k \wedge \forall u, v \in V_i. \{u, v\} \notin E \}$$

## ● Subgraph Isomorphism

$CLIQUE \leq_p SGI$

- Gegeben zwei Graphen  $G_1 = (V_1, E_1)$  und  $G_2 = (V_2, E_2)$ .
- Gibt es einen Subgraphen  $H$  von  $G_1$ , der isomorph zu  $G_2$  ist?

$$SGI = \{ (G_1, G_2) \mid G_1, G_2 \text{ Graphen} \wedge \exists H \text{ Graph. } H \subseteq G_1 \wedge H \cong G_2 \}$$

## ● Largest Common Subgraph

$SGI \leq_p LCS$

- Gegeben Graphen  $G_1 = (V_1, E_1)$  und  $G_2 = (V_2, E_2)$  und eine Zahl  $k \leq |G_1|$
- Gibt es isomorphe Subgraphen  $H_1$  von  $G_1$  und  $H_2$  von  $G_2$  der Größe  $k$ ?

$$LCS = \{ (G_1, G_2, k) \mid G_1, G_2 \text{ Graphen} \wedge k \leq |G_1| \wedge \exists H_1, H_2 \text{ Graphen.} \\ H_1 \subseteq G_1 \wedge H_2 \subseteq G_2 \wedge H_1 \cong H_2 \wedge |H_1| \geq k \}$$

## WEITERE $\mathcal{NP}$ -VOLLSTÄNDIGE GRAPHENPROBLEME

### ● Directed Hamiltonian Circuit

$$3SAT \leq_p DHC$$

- Gegeben ein gerichteter Graph  $G$
- Gibt es in  $G$  einen Hamilton'schen Kreis?

$$DHC = \{ G \mid G = (V, E) \text{ gerichteter Graph} \\ \wedge \exists \pi: \{1..n\} \rightarrow \{1..n\}. \pi \text{ Hamilton'scher Kreis in } G \}$$

### ● Hamiltonian Circuit

$$DHC \leq_p HC$$

- Gegeben ein ungerichteter Graph  $G = (V, E)$ .
- Gibt es in  $G$  einen Hamilton'schen Kreis?

$$HC = \{ G \mid G = (V, E) \text{ Graph} \wedge \exists \pi: \{1..n\} \rightarrow \{1..n\}. \\ \pi \text{ Hamilton'scher Kreis in } G \}$$

### ● Travelling Salesman Problem

$$HC \leq_p TSP$$

- Gegeben  $n$  Städte, Reisekostentabelle  $c_{ij}$ , Kostenbeschränkung  $B$
- Gibt es eine Rundreise durch alle  $n$  Städte, deren Kosten unter  $B$  liegt?

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

## ● **Knapsack: (Rucksack-Bepackung)** $3SAT \leq_p KP$

- Gegeben  $n$  Objekte mit Gewichten  $g_1, \dots, g_n$  und Nutzwerten  $a_1, \dots, a_n$
- Rucksack mit Gewichtsschranke  $G$ , Minimalnutzwert  $A$ .
- Gibt es eine Bepackung mit Mindestnutzen  $A$  und Maximalgewicht  $G$ ?

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

## ● **Partitionsproblem** $KP \leq_p PART$

- Gegeben  $n$  Objekte mit Wert  $b_1, \dots, b_n$ .
- Gibt es eine Aufteilung der Objekte in zwei gleichwertige Stapel?

$$PART = \{ b_1, \dots, b_n \mid b_i \in \mathbb{N} \wedge \exists I \subseteq \{1..n\}. \sum_{i \in I} b_i = \sum_{i \in \{1..n\} - I} b_i \}$$

## ● **Binpacking** $PART \leq_p BPP$

- Gegeben  $n$  Objekte der Größe  $a_1, \dots, a_n$  und  $k$  Behälter der Größe  $b$
- Kann man alle Objekte in den Behältern unterbringen?

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

## WEITERE $\mathcal{NP}$ -VOLLSTÄNDIGE PROBLEME

### ● Graph Coloring

$$3SAT \leq_p GC$$

- Gegeben ein Graph  $G = (V, E)$  der Größe  $n$  und eine Zahl  $k \leq n$ .
- Gibt es eine Färbung von  $V$  mit  $k$  verschiedenen Farben, so daß verbundene Knoten verschiedene Farben haben?

$$GC = \{ (G, k) \mid G=(V, E) \text{ Graph} \wedge \exists f_V: V \rightarrow \{1..k\}. \forall \{u, v\} \in E. f_V(u) \neq f_V(v) \}$$

### ● Multiprozessor-Scheduling

$$MSP \hat{=} BPP$$

- Gegeben  $n$  Prozesse  $j_i$  mit Laufzeit  $t(j_i)$ ,  $m$  Prozessoren, Deadline  $t_D$ .
- Gibt es eine Verteilung der Prozesse auf die Prozessoren, so daß bei Startzeit  $t_0$  alle Prozesse vor der Zeit  $t_D$  beendet sind?

### ● Integer Linear Programming

$$3SAT \leq_p ILP$$

- Gegeben eine  $k \times k$  Matrix  $A$  und einen Vektor  $\vec{b} \in \mathbb{Z}^k$
- Gibt es ein  $\vec{x} \in \mathbb{Z}^k$ , welches das lineare Ungleichungssystem  $A * \vec{x} \geq \vec{b}$  löst?

### ● Zusammengesetztheit (vermutlich nicht $\mathcal{NP}$ -vollständig)

- Gegeben eine  $n$ -stellige Zahl  $x \in \mathbb{N}$
- Gibt es zwei natürliche Zahlen  $p$  und  $q$  mit  $x = p * q$ ?

# NP-VOLLSTÄNDIGKEIT DES RUCKSACKPROBLEMS

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

## ● $KP \in NP$ :

- Rate Menge von Gegenständen Kantenmenge  $J \subseteq \{1..n\}$
- Prüfe  $\sum_{i \in J} g_i \leq G$  und  $\sum_{i \in J} a_i \geq A$  *maximal  $2|J|$  Schritte*

## ● Zeige $3SAT \leq_p KP$

- Gegeben  $F = (k_1, \dots, k_m)$  mit  $k_i = z_{i1} \vee z_{i2} \vee z_{i3}$  und  $z_{ij} \in \{x_1, \dots, \underline{x_n}\}$
- Konstruiere Rucksackproblem  $f(F) \equiv (g_1, ..g_{2m+2n}, a_1, ..a_{2m+2n}, G, A)$   
wobei die  $a_j$  und  $g_j$   $m + n$ -stellige Zahlen sind, welche die Anzahl  
der Vorkommen von Literalen in den Klauseln codieren
  - $a_j, j \leq n$ : Stelle  $i \leq m$  ist Anzahl der  $x_j$  in  $k_i$ , Stelle  $m+j$  ist 1, sonst 0
  - $b_j \equiv a_{n+j}, j \leq n$ : Stelle  $i \leq m$  ist Anzahl der  $\bar{x}_j$  in  $k_i$ , Stelle  $m+j$  ist 1, sonst 0
  - $c_i \equiv a_{2n+i}, i \leq m$ : Stelle  $m+i$  ist 1, sonst 0
  - $d_i \equiv a_{2n+m+i}, i \leq m$ : Stelle  $m+i$  ist 2, sonst 0
  - $g_j = a_j$  für alle  $j$       $A \equiv G = \underbrace{4 \dots 4}_{m\text{-mal}} \underbrace{1 \dots 1}_{n\text{-mal}}$
- $f$  ist in polynomieller Zeit berechenbar

# CODIERUNG EINER FORMEL ALS RUCKSACKPROBLEM

$$F = (k_1, k_2, k_3) \text{ mit } k_1 = x_1 \vee \underline{x_2} \vee x_3 \quad k_2 = \underline{x_1} \vee x_2 \vee \underline{x_4} \quad k_3 = \underline{x_1} \vee \underline{x_2} \vee \underline{x_3}$$

$$A = 444\,1111$$

$$a_1 = 100\,1000 \quad b_1 = 011\,1000 \quad c_1 = 100\,0000 \quad d_1 = 200\,0000$$

$$a_2 = 010\,0100 \quad b_2 = 101\,0100 \quad c_2 = 010\,0000 \quad d_2 = 020\,0000$$

$$a_3 = 100\,0010 \quad b_3 = 001\,0010 \quad c_3 = 001\,0000 \quad d_3 = 002\,0000$$

$$a_4 = 000\,0001 \quad b_4 = 010\,0001$$

(1, 1, 0, 0) ist erfüllende Belegung

$$a_1 + a_2 + b_3 + b_4 + c_1 + c_3 + d_1 + d_2 + d_3 = A$$

## KORREKTHEIT DER TRANSFORMATION

- $a_j$ : Stelle  $i \leq m$  ist Anzahl der  $x_j$  in  $k_i$ , Stelle  $m+j$  ist 1, sonst 0
- $b_j$ : Stelle  $i \leq m$  ist Anzahl der  $\bar{x}_j$  in  $k_i$ , Stelle  $m+j$  ist 1, sonst 0
- $c_i$ : Stelle  $m+i$  ist 1, sonst 0       $d_i$ : Stelle  $m+i$  ist 2, sonst 0
- $g_j = a_j$  für alle  $j$        $A \equiv G = \underbrace{4 \dots 4}_{m\text{-mal}} \underbrace{1 \dots 1}_{n\text{-mal}}$

Ist  $F \in 3SAT$ , so gibt es eine erfüllende Belegung der  $x_j$

Für  $j \leq n$  wähle  $a_j$  falls  $x_j = 1$  und  $b_j$  sonst

$\mapsto$  In der Summe haben alle Stellen  $m+j$  den Wert 1

$\mapsto$  Da  $k_i$  erfüllt wird, haben die Stellen  $i \leq n$  einen Wert aus  $\{1..3\}$

Die Stellen  $i \leq n$  können mit  $c_i$  und  $d_i$  zu 4 ergänzt werden Also  $f(F) \in KP$

Gilt  $f(F) \in KP$ , so gibt es eine Bepackung die genau den Wert  $A$  ergibt

Die Bepackung enthält für  $j \leq n$  entweder  $a_j$  (wähle  $x_j := 1$ ) oder  $b_j$  ( $x_j := 0$ )

Wegen  $c_i + d_i = 3$  ist jede Stelle  $i \leq m$  der Summe der  $a_j$  und  $b_j$  mindestens 1

Also kommt in jeder Klausel  $k_i$  mindestens ein Literal mit dem Wert 1 vor

Damit erfüllt die Belegung die Formel  $F$ , also  $F \in 3SAT$



$$3SAT \leq_p KP$$

# $\mathcal{NP}$ -VOLLSTÄNDIGKEIT DES FÄRBBARKEITSPROBLEMS

$$GC = \{ (G, k) \mid G=(V, E) \text{ Graph} \wedge \exists f_V: V \rightarrow \{1..k\}. \forall \{u, v\} \in E. f_V(u) \neq f_V(v) \}$$

## Zeige $3SAT \leq_p GC$

- Gegeben  $F = (k_1, \dots, k_m)$  mit  $k_i = z_{i1} \vee z_{i2} \vee z_{i3}$  und  $z_{ij} \in \{x_1, \dots, \underline{x}_n\}$
- Konstruiere Färbungsproblem  $f(F) \equiv (G, 3)$  wie folgt

- Teilgraph für Codierung der Variablenbelegung

Wähle  $V_{var} = \{u, x_1, \dots, \underline{x}_n\}$

und  $E_{var} = \{\{u, x_1\}, \{u, \underline{x}_1\}, \{x_1, \underline{x}_1\}, \dots, \{u, x_n\}, \{u, \underline{x}_n\}, \{x_n, \underline{x}_n\}\}$

Bei 3-Färbbarkeit erhalten  $x_i$  und  $\underline{x}_i$  verschiedene Farben aus 0 oder 1

- Teilgraph für Codierung der Klauseln

Wähle  $V_k = \{v, a_1, b_1, c_1, y_1, z_1, \dots, a_m, b_m, c_m, y_m, z_m\}$

und  $E_k = \{\{v, y_1\}, \{v, z_1\}, \{a_1, y_1\}, \{a_1, z_1\}, \{b_1, y_1\}, \{c_1, z_1\}, \{b_1, c_1\}, \dots, \{v, y_m\}, \dots, \{b_m, c_m\}, \{u, v\}\}$

Knoten  $v$  erhält Farbe 0 oder 1

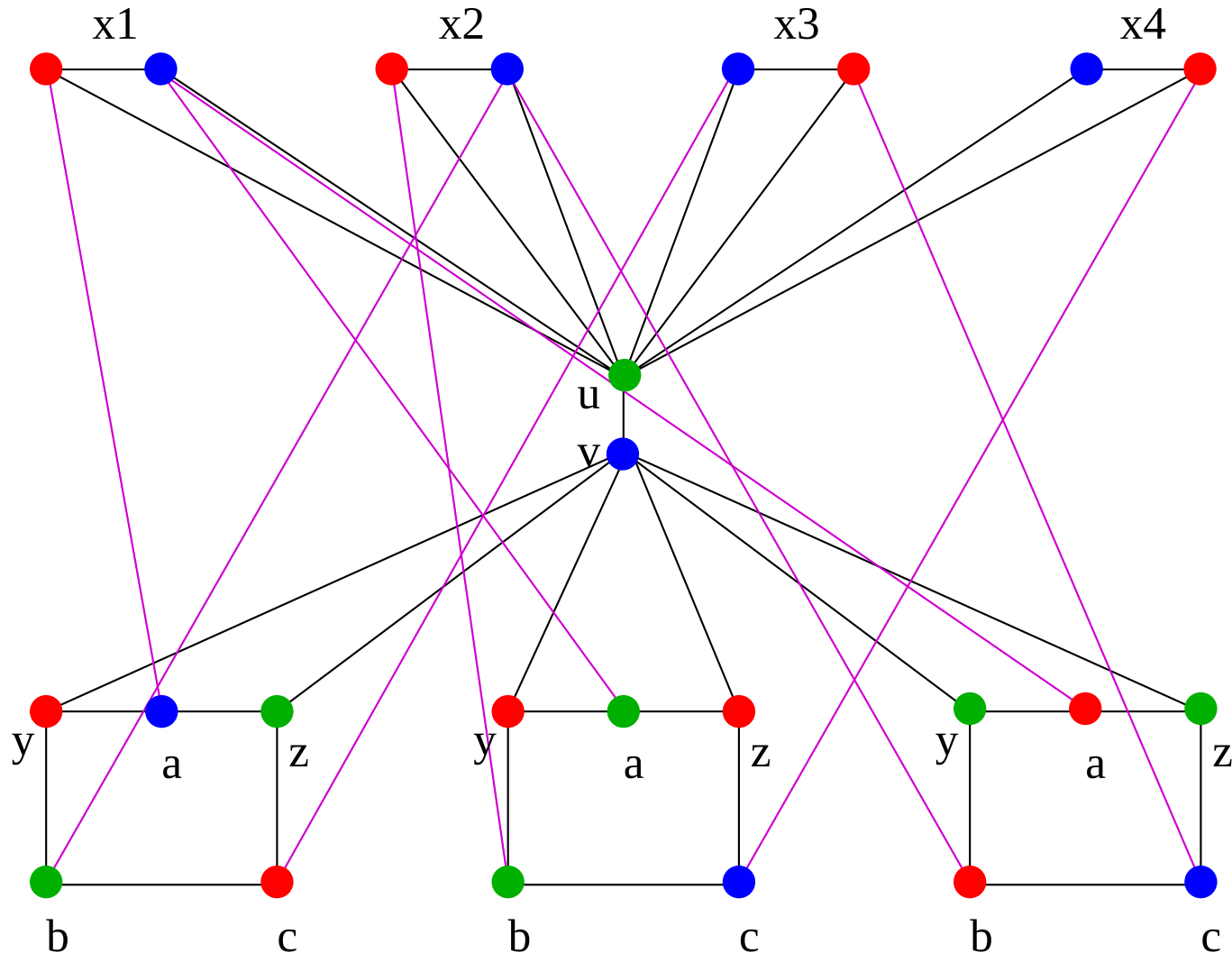
- Kanten zur Codierung der Klauselliterale

$E_{lit} = \{ \{a_1, z_{11}\}, \{b_1, z_{12}\}, \{c_1, z_{13}\}, \dots, \{a_m, z_{m1}\}, \{b_m, z_{m2}\}, \{c_m, z_{m3}\} \}$

- $G=(V_{var} \cup V_k, E_{var} \cup E_k \cup E_{lit})$  ist in polynomieller Zeit berechenbar

# CODIERUNG EINER FORMEL ALS FÄRBUNGSPROBLEM

$F = (k_1, k_2, k_3)$  mit  $k_1 = x_1 \vee \underline{x_2} \vee x_3$   $k_2 = \underline{x_1} \vee x_2 \vee \underline{x_4}$   $k_3 = \underline{x_1} \vee \underline{x_2} \vee \underline{x_3}$



# KORREKTHEIT DER TRANSFORMATION

- Ist  $F \in 3SAT$ , so gibt es eine erfüllende Belegung der  $x_j$   
Wähle  $f_v(x_i), f_v(\underline{x_i}) \in \{0,1\}$  entsprechend,  $f_v(u) = 2$  und  $f_v(v) = 0$ .  
Da jedes  $k_i$  erfüllbar ist, kann einer der  $a_i, b_i, c_i$  die Farbe 0 erhalten  
Da jedes  $k_i$  erfüllbar ist, kann einer der  $a_i, b_i, c_i$  die Farbe 0 erhalten  
Die anderen 4 Knoten bilden eine Kette und werden abwechselnd gefärbt  
Also  $f(F) \in GC$
- Ist  $f(F) \in GC$  dann ist o.B.d.A.  $f_v(u) = 2$  und  $f_v(v) = 0$   
Wähle Belegung der  $x_i$  entsprechend der Färbung von  $x_i$   
Wäre Klausel  $k_i$  nicht erfüllt, so müßte die Farbe der  $a_i, b_i, c_i$  1 oder 2 sein  
Wegen  $f_v(b_i) \neq f_v(c_i)$  und  $f_v(v) = 0$  wäre dann  $\{f_v(y_i), f_v(z_i)\} = \{1, 2\}$   
Dies widerspricht der Färbbarkeit, da  $a_i$  ebenfalls mit 1 oder 2 gefärbt ist.  
Also  $F \in 3SAT$



$$3SAT \leq_p GC$$

## JENSEITS VON $\mathcal{NP}$ -VOLLSTÄNDIGKEIT

- **$co\text{-}\mathcal{NP}$** : Probleme mit Komplement in  $\mathcal{NP}$

- Die Menge der gültigen Formeln ist in  $co\text{-}\mathcal{NP}$  (Komplement von  $SAT$ )
- Das Primzahlproblem liegt in  $co\text{-}\mathcal{NP}$
- Das Primzahlproblem liegt auch in  $\mathcal{NP}$
- Ist ein  $co\text{-}\mathcal{NP}$  Problem  $L$   $\mathcal{NP}$ -vollständig, so gilt  $\mathcal{NP} = co\text{-}\mathcal{NP}$   
Es würde folgen:  $L' \leq_p L \in \mathcal{NP}$  für jedes  $L' \in co\text{-}\mathcal{NP}$
- Das Zusammengesetztheitsproblem ist vermutlich nicht  $\mathcal{NP}$ -vollständig sondern liegt zwischen  $\mathcal{P}$  und  $\mathcal{NPC}$  (sofern

- Es gibt  **$PSPACE$ -vollständige** Probleme

In-Place Acceptance

- Gegeben DTM  $\tau$  und  $w \in X^*$ : Gilt  $h_\tau(w) = 1$  und  $s_\tau(w) \leq |w|$ ?

QBF: Ist eine gegebene quantifizierte boole'sche Formel wahr?

Wie kann man unhandhabbare Probleme angehen?

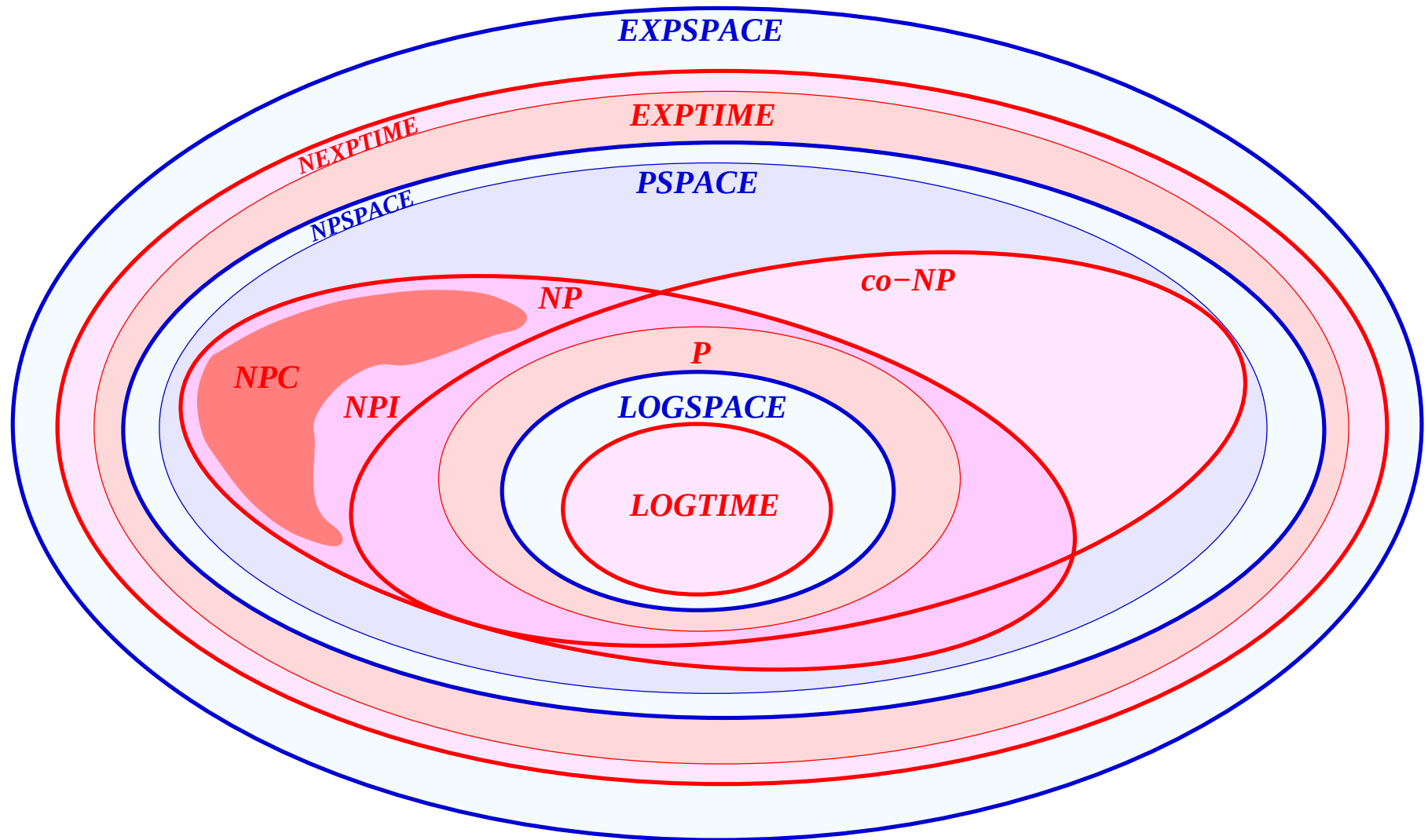
## ● Zeitkomplexitätsklassen

|                   |                                                                                       |
|-------------------|---------------------------------------------------------------------------------------|
| <i>LOGTIME</i>    | in logarithmischer Zeit lösbar                                                        |
| <i>NLOGTIME</i>   | nichtdeterministisch in logarithmischer Zeit lösbar                                   |
| $\mathcal{P}$     | in polynomieller Zeit lösbar                                                          |
| $\mathcal{NP}$    | nichtdeterministisch in polynomieller Zeit lösbar                                     |
| $\mathcal{NPC}$   | $\mathcal{NP}$ -vollständige Probleme                                                 |
| $\mathcal{NPI}$   | $\mathcal{NP}$ -unvollständige Probleme: $\mathcal{NP} - \mathcal{NPC} - \mathcal{P}$ |
| $co-\mathcal{NP}$ | Komplement in $\mathcal{NP}$                                                          |
| <i>EXPTIME</i>    | in exponentieller Zeit lösbar                                                         |
| <i>NEXPTIME</i>   | nichtdeterministisch in exponentieller Zeit lösbar                                    |

## ● Platzkomplexitätsklassen

|                  |                                                                |
|------------------|----------------------------------------------------------------|
| <i>LOGSPACE</i>  | mit logarithmischem Platzverbrauch lösbar                      |
| <i>NLOGSPACE</i> | nichtdeterministisch mit logarithmischem Platzverbrauch lösbar |
| <i>PSPACE</i>    | mit polynomielltem Platzverbrauch lösbar                       |
| <i>NPSPACE</i>   | nichtdeterministisch mit polynomielltem Platzverbrauch lösbar  |
| <i>EXPSPACE</i>  | mit exponentiellem Platzverbrauch lösbar                       |

# SPRACHKLASSENHIERARCHIE



# WICHTIGE VERTRETER VERSCHIEDENER KLASSEN

- **Isomorphie ungerichteter Graphen**  $\mathcal{NP}$
- **Zuverlässigkeit von Netzwerken**  $\mathcal{NP}$ -hart, vermutlich nicht in  $\mathcal{NP}$ 
  - Wahrscheinlichkeit für fehlerfreie Verbindung zwischen zwei Knoten
- **Minimale äquivalente Schaltkreise**  $\mathcal{NP}$ -hart, nicht in  $\mathcal{NP}$  (“ $\Sigma_2$ ”)
  - Bestimme optimale Größe einer Schaltung
- **Quantifizierte boole’sche Formeln**  $PSPACE$ -vollständig
  - Gültigkeit aussagenlogischer Formeln mit boole’schen Quantoren
- **Strategische Spiele**  $PSPACE$ -vollständig
  - Details in Garey/Johnson Seite 254ff
- **TSP\*: Bestimmung aller Rundreisen mit gegebenen Kosten**
  - Unrealistische Problemstellung: zu viele Lösungen  $EXSPACE$

# Theoretische Informatik



## Einheit 4.5

### Grenzen überwinden



1. Pseudopolynomielle Algorithmen
2. Approximationsalgorithmen
3. Probabilistische Algorithmen

# WIE KANN MAN “UNLÖSBARE” PROBLEME ANGEHEN?

- **Künstliche Intelligenz**

- Heuristische Lösung unentscheidbarer Probleme (ohne Erfolgsgarantie)
- Theorembeweisen, Programmverifikation und -synthese (unvollständig)

- **Approximierende und probabilistische Algorithmen**

- Effiziente Bestimmung von nahezu optimalen Lösungen
- Z.B. Primzahltest mit geringem Fehler (logarithmisch statt linear)

- **Selbstorganisation statt vorformulierter Lösungen**

- Lernverfahren, Neuronale Netze, genetische Algorithmen, ...

Suche nach neuen Wegen liefert tieferes Verständnis der Materie

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

### ● Was unterscheidet *CLIQUE* von *KP*?

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

### ● Es gibt “bessere” Lösungen für *KP*

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

- 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 $KP$

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

## ● Betrachte Subprobleme $KP(k, g)$

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

## ● Löse Rucksackproblem $KP$

- Es gilt  $(g_1..g_n, a_1..a_n, G, A) \in KP \Leftrightarrow N(n, G) \geq A$
- Gleichungen beschreiben rekursiven Algorithmus für  $N(n, G)$
- Tabellarischer Algorithmus bestimmt alle  $N(k, g)$  mit  $k \leq n$  und  $g \leq G$
- Laufzeit ist  $\mathcal{O}(n * G)$



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

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

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

## STARKE $\mathcal{NP}$ -VOLLSTÄNDIGKEIT

- **Pseudopolynomiell  $\hat{=}$  effizient bei kleinen Zahlen**

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

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

- Der Reduktionsbeweis  $HC \leq_p TSP$  zeigt  $HC \leq_p TSP_n$
- Eine Restriktion von  $TSP$  auf kleine Zahlen bleibt  $\mathcal{NP}$ -vollständig

- **$TSP$  ist stark  $\mathcal{NP}$ -vollständig**

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



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

- **Viele Probleme haben Optimierungsvariante**

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

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

**Wie effizient kann man eine optimale Lösung annähern?**

- **Optimierungsproblem  $M \subseteq X^*$**

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

- **Approximationsalgorithmus  $A$  für  $M \subseteq X^*$**

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

# APPROXIMATIONSSCHEMATA FÜR DAS RUCKSACKPROBLEM

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

## ● Beliebiger multiplikativer Fehler

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

## ● Kein konstanter additiver Fehler möglich

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

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

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

**Beweistechnik: Multiplikation des Problems, nachträgliche Division des Fehlers**

# APPROXIMATIONSSCHEMATA FÜR BINPACKING

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

## ● Asymptotische Güte 11/9 erreichbar

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

## ● Keine absolute Güte besser als 3/2 (falls $\mathcal{P} \neq \mathcal{NP}$ )

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

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

# APPROXIMATION DES TRAVELING SALESMAN PROBLEMS

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

- $R_A^\infty = 3/2$  erreichbar bei Dreiecksungleichung
- Keine endliche Grenze für multiplikativen Fehler

– Direkte Verbindung ist kürzer als ein Umweg:  $\forall i, j, k. c_{i,j} \leq c_{i,k} + c_{k,j}$

– Es gibt keinen polynomiellen Algorithmus  $A$  mit  $R_A^\infty = r$  für ein  $r \in \mathbb{N}$

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

- Transformiere einen Graphen  $G = (V, E)$  in  $w = c_{12}, \dots, c_{n-1,n}, |V|$  mit  $c_{ij} = 1$  falls  $\{i, j\} \in E$  und  $c_{ij} = r|V| + 1$  sonst
- Dann  $G \in HC \Rightarrow OPT_{TSP}(w) = |V|$  und  $G \notin HC \Rightarrow OPT_{TSP}(w) > (r+1) * |V|$
- Für große Graphen:  $A(w) \leq r * OPT_{TSP}(w)$  also  $G \in HC \Leftrightarrow A(w) \leq r * |V|$   
(Für kleine Graphen verwende den exponentiellen Entscheidungsalgorithmus)

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

# TRAVELLING SALESMAN MIT DREIECKSUNGLEICHUNG

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

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

## • Laufzeit des Algorithmus ist $O(n^3)$

## • Güte des Algorithmus ist $R_A^{\infty} \leq 3/2$ (aufwendig)

## “Approximation” einer Entscheidung

- **Verhalten gesteuert durch Zufallszahlen**

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

- **Anwendungen**

- Primzahltest in linearer Zeit
- Optimierung von Quicksort auf  $\mathcal{O}(n \cdot \log n)$  (Bestimmung Pivotelement)

- **Wie weist man gut Eigenschaften nach?**

- Einfaches Modell für probabilistische Algorithmen formulieren
- Eigenschaften abstrakter probabilistischer Sprachklassen analysieren

## ● Probabilistische Turingmaschine

- Struktur:  $\tau = (S, X, \Gamma, \delta, s_0, b)$
- Zustandsüberföhrungsfunktion:  $\delta: S \times \Gamma \rightarrow (S \times \Gamma \times \{r, l, h\})^2$   
Jede Alternative wird mit Wahrscheinlichkeit  $1/2$  ausgewählt
- Ausgabe:  $h_\tau(w) \in \{0, 1, ?\}$  (Akzeptieren – Verwerfen – keine Aussage)
- Rechenzeit: maximale Rechenzeit aller möglichen Rechenwege
- PTM: polynomiell zeitbeschränkte probabilistische Turingmaschine

## ● Abstrakteres Modell: Probabilistische Algorithmen

- Programme mit zufälligen Entscheidungen
- Abstrakte Komplexität wie bisher

Was kann man mit polynomiell zeitbeschränkten probabilistischen Algorithmen erreichen?

# WICHTIGE PROBABILISTISCHE SPRACHKLASSEN

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

# PROBABILISTISCHER PRIMZAHLTEST FÜR $n \geq 3$

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

## **RP-Algorithmus**

- Korrekte Ausgabe, falls  $n$  Primzahl
- Fehlerwahrscheinlichkeit unter  $1/2$ , falls  $n$  keine Primzahl

**Rechenzeit  $\leq 6 * \log n$**

- **$k$ -fache Iteration von  $RP$  Algorithmen verringert die Wahrscheinlichkeit einer falschen Antwort auf  $2^{-k}$** 
  - Ist  $\tau$  die  $k$ -fache **statistisch unabhängige** Iteration einer PTM für  $L \in RP$ , so gilt
$$\forall w \in L. Prob( h_\tau(w)=1 ) > 1-2^{-k} \wedge \forall w \notin L. Prob( h_\tau(w)=0 ) = 1$$
- **$t$ -fache Iteration eines  $BPP$  Algorithmus für  $t > \frac{k}{-\log(1-4\epsilon^2)}$  verringert die Wahrscheinlichkeit der falschen Antwort auf  $2^{-k}$** 
  - Sei  $\tau^t$  die  $(2t+1)$ -fache **statistisch unabhängige** Iteration einer PTM  $\tau$  für  $L \in BPP$ , die genau dann akzeptiert, wenn  $\tau$  mindestens  $t+1$ -mal akzeptiert, so gilt für  $t > \frac{k-1}{-\log(1-4\epsilon^2)}$ 

$$\forall w. Prob( h_{\tau^t}(w)=\chi_L(w) ) > 1-2^{-k}$$
- **$\mathcal{P} \subseteq ZPP \subseteq RP \subseteq BPP \subseteq PP$   
 $RP \subseteq \mathcal{NP} \subseteq \mathcal{NP} \cup co\text{-}\mathcal{NP} \subseteq PP$**

# SPRACHKLASSENHIERARCHIE

