

# Kapitel 9

## Analyse Randomisierter Algorithmen für Erfüllbarkeitsprobleme

Effiziente Algorithmen, SS 2018

Professor Dr. Petra Mutzel

VO 15/16 am 7./12. Juni 2018

# Übersicht

## I. Effiziente Graphalgorithmen

- ② Starke Zusammenhangskomponenten
- ③ Matching-Probleme
- ④ Maximale Flussprobleme
- ⑤ Amortisierte Analyse
- ⑥ Minimale Schnitte

## II. Approximationsalgorithmen

- ⑦ Rucksackproblem, Bin Packing Problem
- ⑧ Traveling Salesman Problem
- ⑨ **Erfüllbarkeitsprobleme**
- ⑩ Schnittprobleme

# Design-Techniken im Verlauf der Vorlesung

- 1 Die Greedy-Methode: Rucksackproblem (Kap. 7)
- 2 Dynamische Programmierung: Rucksackproblem (FPTAS) (Kap. 7)
- 3 Inkrementelle Algorithmen für Partitionsprobleme: Bin Packing Problem (Kap. 7)
- 4 Spezielle, problemabhängige Verfahren: Traveling Salesman Problem (Kap. 8)
- 5 Zufalls-basierte Verfahren (Kap. 9, Kap. 10)
- 6 LP-basierte Verfahren: MaxSAT (Kap. 9)
- 7 Lokale Suchverfahren: Max Cut (Kap. 10)

## 9.1 Analyse Randomisierter Algorithmen

**Erinnerung** Turingmaschine (TM)  
randomisierte Turingmaschine

**Annahme** Algorithmus und Eingabe **fest**

**klar** bei deterministischer TM  
Rechenweg fest  
**Rechenzeit** = Länge des Rechenweges

**Erinnerung** bei randomisierter TM  
**Baum**  $T$  möglicher Rechenwege  
Kanten **gewichtet** mit Wahrscheinlichkeiten  
**erwartete Rechenzeit**  $E(T) = \sum_{v \text{ Blatt}} \text{Tiefe}(v) \cdot \text{Prob}(v)$   
**erwartete Rechenzeit**  $E(T) = \sum_{v \text{ Knoten}} \text{Prob}(v)$

# Über Zufallsexperimente

in GTI/TIfAI randomisierte TM mit fairen Münzwürfen

hier beinahe beliebige Zufallsexperimente

erforderlich Wahrscheinlichkeitsverteilung  
approximierbar mit polynomiell wenigen Münzwürfen

klar für genaue Rechenzeit Details wichtig

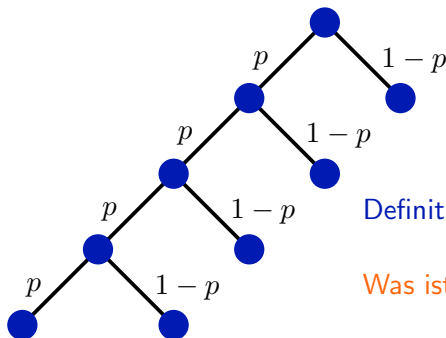
hier Zufallsexperiment in konstanter Zeit

# Ein Miniatur-Beispiel

Sei  $p \in ]0; 1[$  fest.

1. Mit Wahrscheinlichkeit  $p$  setze  $z := 0$   
Sonst setze  $z := 1$
2. If  $z = 0$  Then Weiter bei 1.

Festlegung ein Durchlauf Zeilen 1–2  $\approx 1$  Rechenschritt



Definition  $T = \# \text{Rechenschritte}$

Was ist  $E(T)$ ?

Bestimmung von  $E(T)$ 

## 1. Möglichkeit Summation über die Blätter

$$\begin{aligned}
E(T) &= \sum_{d=1}^{\infty} d \cdot p^{d-1} \cdot (1-p) = (1-p) \cdot \sum_{d=1}^{\infty} d \cdot p^{d-1} \\
&= (1-p) \cdot \left( \sum_{d=1}^{\infty} \sum_{i=d}^{\infty} p^{i-1} \right) = (1-p) \cdot \left( \sum_{d=1}^{\infty} \sum_{i=d-1}^{\infty} p^i \right) \\
&= (1-p) \cdot \left( \sum_{d=1}^{\infty} \left( \sum_{i=0}^{\infty} p^i - \sum_{i=0}^{d-2} p^i \right) \right) \\
&= (1-p) \cdot \left( \sum_{d=1}^{\infty} \left( \frac{1}{1-p} - \frac{1-p^{d-1}}{1-p} \right) \right) \\
&= \sum_{d=1}^{\infty} p^{d-1} = \sum_{d=0}^{\infty} p^d = \frac{1}{1-p}
\end{aligned}$$

# Bestimmung von $E(T)$

## 2. Möglichkeit Summation über die Knoten

$$\begin{aligned} E(T) &= \sum_{d=1}^{\infty} ((1-p) \cdot p^{d-1} + p^d) = \sum_{d=1}^{\infty} (p^{d-1} - p^d + p^d) \\ &= \sum_{d=0}^{\infty} p^d = \frac{1}{1-p} \end{aligned}$$

klar war sowieso: Rechenzeit = #Versuche bis Ereignis „1 - p“

somit Wartezeit geometrisch verteilt, Parameter 1 - p

also  $E(T) = \frac{1}{1-p}$

**MERKE:** Laufzeit  $E(T)$  ermittelbar, auch wenn Algorithmus theoretisch unendlich lange laufen kann



## 2. Beispiel: Zufällige Wahl einer Teilmenge

**Aufgabe** Ziehe aus  $\{1, 2, \dots, 49\}$   
uniform zufällige sechselementige Teilmenge.

**Beobachtung** 1 ist enthalten mit Wahrscheinlichkeit  
 $1/49 + (48/49)(1/48) + (48/49)(47/48)(1/47) + \dots = 6/49$

1. (seltsame) Idee wenn feststeht, ob 1 enthalten  
 falls 1 enthalten 2 mit W'keit 5/48 enthalten  
 sonst 2 mit W'keit 6/48 enthalten

klar so fortsetzbar  
 $\rightsquigarrow$  Algorithmus 1

# Algorithmus 1 (seltsam)

1.  $n := 49; k := 6; i := 1$
2. Repeat
3.     Mit W'keit  $k/n$ : gib  $i$  aus und setze  $k := k - 1$
4.      $i := i + 1; n := n - 1$
5. Until  $k = 0$

**Korrektheit:** gibt immer 6 Zahlen aus

**Festlegung**  $T$  = Rechenzeit =  $i - 1$  am Ende des Algorithmus

**klar**  $E(T) = \sum_{t=0}^{\infty} t \cdot \text{Prob}(T = t)$

**Beobachtung** gibt Zahlen sortiert aus: die größte als letztes

**Festlegung**  $M$  = größte ausgegebene Zahl

# Algorithmus 1 (seltsam)

1.  $n := 49; k := 6; i := 1$
2. Repeat
3.     Mit W'keit  $k/n$ : gib  $i$  aus und setze  $k := k - 1$
4.      $i := i + 1; n := n - 1$
5. Until  $k = 0$

**Festlegung**  $T$  = Rechenzeit =  $i - 1$  am Ende des Algorithmus

**Festlegung**  $M$  = größte ausgegebene Zahl

$$\begin{aligned}
 E(T) &= \sum_{m=6}^{49} m \cdot \text{Prob}(M = m) = \sum_{m=6}^{49} m \cdot \frac{\binom{m-1}{5} \binom{1}{1}}{\binom{49}{6}} \\
 &= \frac{300}{7} \approx 42,857
 \end{aligned}$$

# Kritische Rückschau

**klar** Problem und Algorithmus verallgemeinerbar  
auf beliebige  $n \in \mathbb{N}$  und  $k \in \{1, 2, \dots, n\}$

**dann**  $E(T)$  für Algorithmus 1 vielleicht **zu groß**  
für sehr große  $n$

**Anmerkung** wenn Zufallsexperimente teuer  
weniger Zufallsexperimente wünschenswert

**hier** Algorithmus 2 für  $n = 49$  und  $k = 6$  zum Vergleich

## Algorithmus 2 deutlich besser

1. Für  $i \in \{1, 2, \dots, 49\}$  setze  $a[i] := 0$ .
2. Für  $k \in \{1, 2, \dots, 6\}$
3.     Repeat
4.         Wähle  $z \in \{1, 2, \dots, 49\}$  uniform zufällig.
5.         Until  $a[z] = 0$
6.          $a[z] := 1$
7. Für  $i \in \{1, 2, \dots, 49\}$
8.     If  $a[i] = 1$  Then Ausgabe  $i$

**Beobachtung** Laufzeit  $\Omega(n)$

**Festlegung**  $Z = \# \text{Zufallsexperimente}$

**Beobachtung** für  $k$  im E-Wert  $\frac{49}{49-(k-1)}$  Zufallsexp.

$$E(Z) = \sum_{k=1}^6 \frac{49}{49-(k-1)} = 1 + \frac{49}{48} + \frac{49}{47} + \frac{49}{46} + \frac{49}{45} + \frac{49}{44} \approx 6,33$$

## Algorithmus 3 elegant und gut

1. Für  $i \in \{1, 2, \dots, 49\}$  setze  $a[i] := i$ .
2. Für  $k \in \{1, 2, \dots, 6\}$ 
  3. Wähle  $z \in \{k, k + 1, \dots, 49\}$  gemäß Gleichverteilung zufällig.
  4. Vertausche  $a[k]$  und  $a[z]$ .
5. Für  $k \in \{1, 2, \dots, 6\}$ 
  6. Ausgabe  $a[i]$

**Beobachtung**    Rechenzeit  $\Theta(n)$

**Rechenzeit**    Anzahl Zufallsexperimente =  $k$

## 9.2 Approximationsalgorithmen für MAX- $k$ -SAT

**SAT**    **Eingabe**     $m$  Klauseln  $c_1, c_2, \dots, c_m$   
                               über  $n$  Variablen  $x_1, x_2, \dots, x_n$   
                               Klausel: Disjunktion über Literale  
                               (z. B.  $c_j = x_3 \vee \overline{x_5} \vee \overline{x_7} \vee x_9$ )

**zulässige Lösungen**

Belegungen  $b \in \{0, 1\}^n$  der  $n$  Variablen

**Bewertung** Anzahl durch  $b$  erfüllter Klauseln

klar

- $\text{SAT} \in \mathcal{NPO}$
- zugehöriges Entscheidungsproblem NP-vollständig

# MAXSAT und MAX- $k$ -SAT

**Festlegung** Optimierungsvariante von SAT heißt **MAXSAT**

**Festlegung** Klausel, in der keine Variable mehrfach vorkommt, heißt **reduziert**

**Festlegung** MAXSAT-Instanz mit ausschließlich reduzierten Klauseln heißt **reduziert**

**ab jetzt** nur noch reduzierte Instanzen

**Festlegung** Anzahl Literale einer Klausel heißt **Länge**

**Festlegung** MAXSAT-Instanz, in der alle Klauseln Länge =  $k$  haben, heißt **MAX- $k$ -SAT-Instanz**



# MAX- $k$ -SAT approximieren

## Fakten

- $k$ -SAT NP-vollständig für  $k \in \mathbb{N} \setminus \{1, 2\}$
- 2-SAT  $\in P$
- MAX- $k$ -SAT NP-schwierig für  $k \in \mathbb{N} \setminus \{1\}$
- MAX-2-SAT NP-schwierig

## Algorithmus 9.1

1. Für  $i \in \{1, 2, \dots, n\}$
2. Mit W'keit  $1/2$  setze  $b[i] := 0$  sonst setze  $b[i] := 1$ .
3. Ausgabe  $b$

## Theorem 9.2

Algorithmus 9.1 hat Laufzeit  $\Theta(n)$  und erfüllt im Durchschnitt  $(1 - 2^{-k}) \cdot m$  aller Klauseln einer reduzierten MAX- $k$ -SAT-Instanz mit  $m$  Klauseln.

# Beweis von Theorem 9.2

Laufzeit ✓

Definiere ZV  $X = \#$ durch  $b$  erfüllte Klauseln

Definiere ZV  $X_i = \begin{cases} 1 & b \text{ erfüllt } c_i \\ 0 & \text{sonst} \end{cases}$

klar  $X = \sum_{i=1}^m X_i$

$$\begin{aligned} E(X) &= E\left(\sum_{i=1}^m X_i\right) = \sum_{i=1}^m E(X_i) = \sum_{i=1}^m \text{Prob}(X_i = 1) \\ &= \sum_{i=1}^m \text{Prob}(b \text{ erfüllt } c_i) \end{aligned}$$

# Erfüllen einer MAX- $k$ -SAT-Klausel

Wir haben  $X = \#$ durch  $b$  erfüllte Klauseln

$$E(X) = \sum_{i=1}^m \text{Prob}(b \text{ erfüllt } c_i)$$

Betrachte MAX- $k$ -SAT-Klausel  $c_i$

klar  $c_i$  hat Länge  $k$

Beobachtung  $c_i$  enthält  $k$  verschiedene Variablen  
weil  $c_i$  reduziert

Betrachte alle  $2^k$  Teilbelegungen für  $c_i$

Beobachtung genau 1 Belegung erfüllt  $c_i$  nicht

also  $\text{Prob}(b \text{ erfüllt } c_i) = \frac{2^k - 1}{2^k} = 1 - \frac{1}{2^k}$

also  $E(X) = (1 - 2^{-k}) \cdot m$



# Vom randomisierten Algorithmus zur Approximation

**Erinnerung**  $c$ -Approximation  
liefert deterministisch in Polynomialzeit  
Lösung mit Güte  $\leq c$

Können wir aus Algorithmus 9.1  $c$ -Approximation machen?

**Begriff** systematische Überführung randomisiert  $\rightsquigarrow$  deterministisch  
heißt **Derandomisierung**

# Derandomisierung von Algorithmus 9.1

**Erinnerung**  $b$  wird von links nach rechts belegt  
also  $p_i := \text{Prob}(b[i] = 1)$  immer  $\in \{0, 1/2, 1\}$

$p_i = 1/2$  wenn  $b[i]$  noch nicht gesetzt

**Definition**  $I_i^+ := \{j \in \{1, \dots, n\} \mid x_j \text{ kommt in } c_i \text{ vor}\}$   
 $I_i^- := \{j \in \{1, \dots, n\} \mid \overline{x_j} \text{ kommt in } c_i \text{ vor}\}$

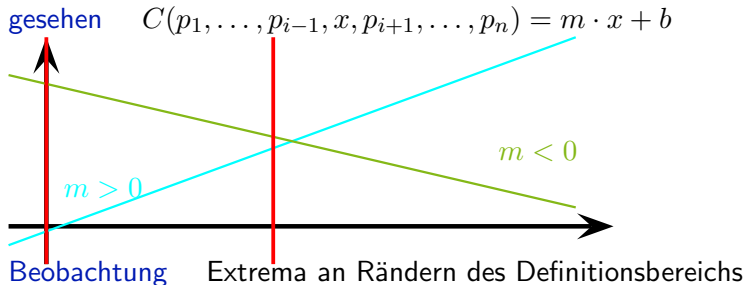
**damit**  $C_i(p_1, \dots, p_n) := \mathbb{E}(X_i) = 1 - \prod_{j \in I_i^+} (1 - p_j) \cdot \prod_{j \in I_i^-} p_j$

**Erinnerung** ZV  $X_i = \begin{cases} 1 & b \text{ erfüllt } c_i \\ 0 & \text{sonst} \end{cases}$

**Beobachtung** gilt auch nach Festlegung von  $p_j \in \{0, 1\}$

**Definition**  $C(p_1, \dots, p_n) := \mathbb{E}(X) = \sum_{i=1}^m C_i(p_1, \dots, p_n)$

**klar**  $C, C_i$  ist für jede **einzelne**  $p_j$  **lineare Funktion in**  $p_j$

Schrittweise „Optimierung“ von  $C$ 

## Algorithmus 9.3

1. Für  $i \in \{1, 2, \dots, n\}$  setze  $p_i := 1/2$ .
2. Für  $i \in \{1, 2, \dots, n\}$
3.     If  $C(p_1, \dots, p_{i-1}, 0, p_{i+1}, \dots, p_n)$   
         $> C(p_1, \dots, p_{i-1}, 1, p_{i+1}, \dots, p_n)$
4.     Then  $p_i := 0$  Else  $p_i := 1$ .
5. Für  $i \in \{1, 2, \dots, n\}$  setze  $b[i] := p_i$
6. Ausgabe  $b$

# Über Algorithmus 9.3

## Theorem 9.4

Algorithmus 9.3 berechnet zu einer reduzierten MAX- $k$ -SAT-Instanz mit  $m$  Klauseln über  $n$  Variablen in Zeit  $\Theta(n \cdot m)$  eine Belegung, die mindestens  $(1 - 2^{-k}) \cdot m$  Klauseln erfüllt.

Algorithmus 9.3 ist eine  $\left(1 + \frac{1}{2^k - 1}\right)$ -Approximation für MAX- $k$ -SAT.

## Beweis der Laufzeit

- Zeilen 1, 5, 6 Zeit  $\Theta(n)$
- **Preprocessing**  $C = m \cdot (1 - 2^{-k})$ ,  $C_j = 1 - 2^{-k}$  Zeit  $\Theta(m)$
- **Preprocessing** je Klausel Inzidenzvektor über Variablen  
Zeit  $\Theta(m \cdot n)$
- **Zeile 3**  $i$ -Schleife über  $\{1, \dots, n\}$  Schleife über Klauseln mit Preprocessing Test und Anpassung in Zeit  $\Theta(1)$   
gesamt Zeit  $\Theta(n \cdot m)$  ✓

# Approximationsgüte von Algorithmus 9.3

initial  $C = m \cdot (1 - 2^{-k})$  ✓

für jedes  $i$  alle  $p_j$  mit  $j \neq i$  fest  
also  $C$  lineare Funktion in  $p_i$

darum Maximum von  $C$  wird von  $p_i \in \{0, 1\}$  erreicht

klar  $p_i$  passend gesetzt

also  $C$  kann nicht kleiner werden

also  $b \in \{0, 1\}^n$  und  $C \geq m \cdot (1 - 2^{-k})$

Zur Approximationsgüte Haben  $\text{OPT} \leq m$ , deswegen

$$\text{Güte} = \frac{\text{OPT}}{C} \leq \frac{m}{m \cdot (1 - 2^{-k})} = \frac{1}{1 - 2^{-k}} = \frac{2^k}{2^k - 1} = 1 + \frac{1}{2^k - 1}$$





## 9.3 Randomisiertes Runden mittels ILP

**Einschub: Ganzzahlige lineare Programmierung (ILP):**  
 siehe Extra-Folien zur Linearen Programmierung (ppt.pdf)

**Eingabe**    lineare Zielfunktion über  $x_1, x_2, \dots, x_n$   
                $m$  lineare Ungleichungen über  $x_1, x_2, \dots, x_n$   
               Koeffizienten  $\in \mathbb{Z}$

**zulässige Lösungen**    Belegungen  $\in \mathbb{Z}^n$   
                               alle linearen Ungleichungen gleichzeitig erfüllt

**Bewertung**    Zielfunktion, maximieren

**klar**    ganzzahlige lineare Optimierung  $\in \mathcal{NPO}$

**Behauptung**    MAXSAT als ganzzahliges lineares Programm  
                       formulierbar

# MAXSAT als ganzzahliges lineares Programm

**Variable** für MAXSAT-Variable  $x_i$  ILP-Variable  $y_i$

für Klausel  $c_j$  ILP-Variable  $z_j$

**Idee**  $z_j$  entspricht ZV  $X_j$

**Zielfunktion**  $\max \sum_{j=1}^m z_j$

**Nebenbedingungen**

$$y_i \leq 1, y_i \geq 0 \text{ für alle } i$$

$$z_j \leq 1, z_j \geq 0 \text{ für alle } j$$

$$\sum_{i \in I_j^+} y_i + \sum_{i \in I_j^-} (1 - y_i) \geq z_j \text{ für alle } j$$

$$y_i \in \mathbb{Z}, z_j \in \mathbb{Z} \text{ für alle } i, j$$

**Beobachtung** polynomielle Reduktion  
 direkte Entsprechung von Belegungen  $x_i \leftrightarrow y_i$   
 direkte Entsprechung Zielfunktion  $\leftrightarrow$  #erfüllte Klauseln

**klar** ILP NP-schwierig **Wo ist denn da der Sinn?**

# Ein anderes Problem: Lineare Programmierung (LP)

**Eingabe**    lineare Zielfunktion über  $x_1, x_2, \dots, x_n$   
                $m$  lineare Ungleichungen über  $x_1, x_2, \dots, x_n$   
               Koeffizienten  $\in \mathbb{Z}$

**zulässige Lösungen**    Belegungen  $\in \mathbb{R}^n$   
                               alle linearen Ungleichungen gleichzeitig erfüllt

**Bewertung**    Zielfunktion, maximieren

**Fakt**    lineare Programmierung  $\in \text{P}$

**Definition: kanonische LP-Relaxierung:**    LP, das man erhält, wenn man in einem  $\{0/1\}$ -ILP alle Binärvariablen durch Variable im Bereich  $[0, 1]$  ersetzt.

**Lösung**    statt  $\in \{0, 1\}^n$  nun  $\in [0, 1]^n$

**Beobachtung**    Lösungsraum größer, weniger Einschränkungen  
                       heißt **Relaxierung**

# Randomisiertes Runden

## Was nützt uns die Lösung des relaxierten Problems?

Idee Belegungen geben Hinweise

Wert für  $y_i$  nahe 0  $\rightsquigarrow$  vermutlich besser  $x_i = 0$  setzen

Wert für  $y_i$  nahe 1  $\rightsquigarrow$  vermutlich besser  $x_i = 1$  setzen

### Algorithmus 9.5

1. Formuliere zur MAX- $k$ -SAT-Instanz das lineare Programm.
2. Berechne optimale Lösung  $\hat{y}_1, \hat{y}_2, \dots, \hat{y}_n, \hat{z}_1, \hat{z}_2, \dots, \hat{z}_m$  dazu.
3. Für  $i \in \{1, 2, \dots, n\}$
4. Mit W'keit  $\hat{y}_i$  setze  $b[i] := 1$  sonst setze  $b[i] := 0$ .
5. Ausgabe  $b$

# Über randomisiertes Runden

## Theorem 9.6

- Algorithmus 9.5 berechnet zu einer reduzierten MAX- $k$ -SAT-Instanz mit  $m$ -Klauseln über  $n$  Variablen in Polynomialzeit eine Belegung, die im Erwartungswert mindestens  $(1 - (1 - 1/k)^k) \cdot \text{OPT}$  Klauseln erfüllt, wenn OPT die maximal gleichzeitig erfüllbare Anzahl der Klauseln angibt.
- Algorithmus 9.5 berechnet zu einer MAXSAT-Instanz mit  $m$ -Klauseln über  $n$  Variablen in Polynomialzeit eine Belegung, die im Erwartungswert mindestens  $(1 - e^{-1}) \cdot \text{OPT}$  Klauseln erfüllt, wenn OPT die maximal gleichzeitig erfüllbare Anzahl der Klauseln angibt.
- Jede einzelne Klausel  $c_j$  mit  $k$  Literalen wird mit Wahrscheinlichkeit mindestens  $(1 - (1 - \frac{1}{k})^k) \cdot \hat{z}_j$  erfüllt.

# Zwei Hilfsaussagen (ohne Beweis)

## Lemma 9.7

$$\forall a_1, a_2, \dots, a_k \in \mathbb{R}_0^+ : \prod_{i=1}^k a_i \leq \left( \frac{\sum_{i=1}^k a_i}{k} \right)^k$$

## Lemma 9.8

$$\forall x \in [0, 1] : \forall k \in \mathbb{N} : 1 - \left(1 - \frac{x}{k}\right)^k \geq \left(1 - \left(1 - \frac{1}{k}\right)^k\right) \cdot x$$

## Beweis von Theorem 9.6

Betrachte Klausel  $c_j$ O. B. d. A.  $c_j = x_1 \vee x_2 \vee \cdots \vee x_k$ Beobachtung  $\text{Prob}(c_j \text{ nicht erfüllt}) = \prod_{i=1}^k (1 - \hat{y}_i)$ also  $\text{Prob}(c_j \text{ erfüllt}) = 1 - \prod_{i=1}^k (1 - \hat{y}_i)$ aus LP  $\sum_{i=1}^k \hat{y}_i \geq \hat{z}_j$ also  $\sum_{i=1}^k (1 - \hat{y}_i) \leq k - \hat{z}_j$ mit Lemma 9.7  $\prod_{i=1}^k (1 - \hat{y}_i) \leq \left( \frac{\sum_{i=1}^k (1 - \hat{y}_i)}{k} \right)^k \leq \left( \frac{k - \hat{z}_j}{k} \right)^k = \left( 1 - \frac{\hat{z}_j}{k} \right)^k$

# Abschätzung für Klausel $c_j$

Wir haben  $\text{Prob}(c_j \text{ erfüllt}) = 1 - \prod_{i=1}^k (1 - \hat{y}_i)$

$$\prod_{i=1}^k (1 - \hat{y}_i) \leq \left(1 - \frac{\hat{z}_j}{k}\right)^k$$

also  $\text{Prob}(c_j \text{ erfüllt}) = 1 - \prod_{i=1}^k (1 - \hat{y}_i) \geq 1 - \left(1 - \frac{\hat{z}_j}{k}\right)^k$

$$\geq \left(1 - \left(1 - \frac{1}{k}\right)^k\right) \cdot \hat{z}_j \text{ (mit Lemma 9.8)}$$

Fakt  $1 - \left(1 - \frac{1}{k}\right)^k \geq 1 - e^{-1}$



# Abschätzung für alle Klauseln

Wir haben  $\text{Prob}(c_j \text{ erfüllt}) \geq \left(1 - \left(1 - \frac{1}{k}\right)^k\right) \cdot \hat{z}_j$

$$1 - \left(1 - \frac{1}{k}\right)^k \geq 1 - e^{-1}$$

$$\begin{aligned} E(\# \text{erfüllte Klauseln}) &= \sum_{j=1}^m \text{Prob}(c_j \text{ erfüllt}) \\ &\geq \sum_{j=1}^m \left(1 - \left(1 - \frac{1}{k}\right)^k\right) \cdot \hat{z}_j \\ &\geq \sum_{j=1}^m (1 - e^{-1}) \cdot \hat{z}_j = (1 - e^{-1}) \cdot \sum_{j=1}^m \hat{z}_j \end{aligned}$$

MAX- $k$ -SAT und MAXSAT

Wir haben

$$\begin{aligned}
 & \mathbb{E}(\text{\#erfüllte Klauseln}) \\
 & \geq \left(1 - \left(1 - \frac{1}{k}\right)^k\right) \cdot \sum_{j=1}^m \hat{z}_j \\
 & \geq (1 - e^{-1}) \cdot \sum_{j=1}^m \hat{z}_j
 \end{aligned}$$

Erinnerung LP ist Relaxierung

deshalb

$$\sum_{j=1}^m \hat{z}_j \geq \text{OPT}$$

also

$$\mathbb{E}(\text{\#erfüllte Klauseln}) \geq (1 - e^{-1}) \cdot \text{OPT}$$



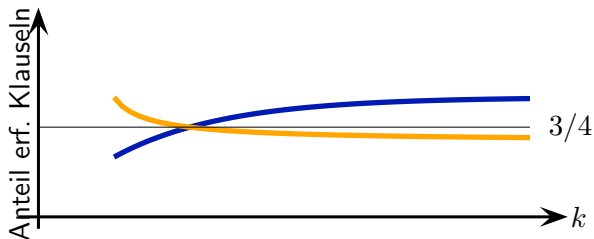
## 9.4 Kombination beider Algorithmen

Zwei randomisierte Algorithmen für MAX- $k$ -SAT  
im Erwartungswert

**Algorithmus 9.1** erfüllt  $\geq (1 - 2^{-k}) \cdot \text{OPT}$  Klauseln  
 $\hat{=}$  Güte  $\left(1 + \frac{1}{2^k - 1}\right)$

**Algorithmus 9.5** erfüllt  $\geq (1 - (1 - 1/k)^k) \cdot \text{OPT}$  Klauseln  
 $\hat{=}$  Güte  $\left(1 + \frac{(1 - 1/k)^k}{1 - (1 - 1/k)^k}\right)$

Welcher Algorithmus ist besser?



# Eine naheliegende Idee

## Algorithmus 9.9

1. Berechne Belegung  $a_1$  mit Algorithmus 9.1.
2. Berechne Belegung  $a_2$  mit Algorithmus 9.5.
3. Gib die Belegung aus, die mehr Klauseln erfüllt.

Nützt das etwas?

Sei  $A_1 = \# \text{erf. Klauseln durch } a_1$   
 $A_2 = \# \text{erf. Klauseln durch } a_2$

klar Algorithmus 9.9 erfüllt  $\max\{A_1, A_2\}$  Klauseln

Ziel  $E(\max\{A_1, A_2\})$  abschätzen

# $E(\max\{A_1, A_2\})$ abschätzen

**Beobachtung**  $\max\{A_1, A_2\} \geq \frac{A_1 + A_2}{2}$

**Betrachte** Klausel  $c_j$  mit Länge  $l_j$

**Erinnerung**  $\text{Prob}(a_1 \text{ erfüllt } c_j) = 1 - 2^{-l_j}$

**Erinnerung**  $\text{Prob}(a_2 \text{ erfüllt } c_j) \geq 1 - (1 - 1/l_j)^{l_j}$

**also**  $E(A_1) \geq \sum_{j=1}^m (1 - 2^{-l_j}) \geq \sum_{j=1}^m (1 - 2^{-l_j}) \cdot \hat{z}_j$

$$E(A_2) \geq \sum_{j=1}^m \left(1 - (1 - 1/l_j)^{l_j}\right) \cdot \hat{z}_j$$

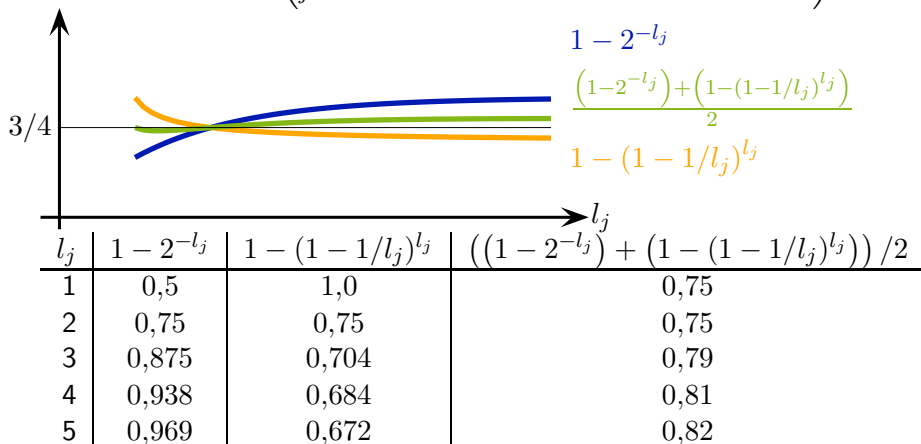
**zusammen**

$$\begin{aligned} & E\left(\frac{A_1 + A_2}{2}\right) \\ &= \frac{1}{2} \cdot (E(A_1) + E(A_2)) \\ &\geq \frac{1}{2} \cdot \left( \sum_{j=1}^m \left( (1 - 2^{-l_j}) + (1 - (1 - 1/l_j)^{l_j}) \right) \cdot \hat{z}_j \right) \end{aligned}$$

# Ergebnis für MAXSAT

Wir haben 
$$E\left(\frac{A_1 + A_2}{2}\right) = \frac{1}{2} \cdot (E(A_1) + E(A_2))$$

$$\geq \frac{1}{2} \cdot \left( \sum_{j=1}^m \left( (1 - 2^{-l_j}) + \left(1 - (1 - 1/l_j)^{l_j}\right) \right) \cdot \hat{z}_j \right)$$



# Zusammenfassung

## Theorem 9.10

Algorithmus 9.9 berechnet zu einer MAXSAT-Instanz, in der höchstens OPT Klauseln gleichzeitig erfüllt werden können, in Polynomialzeit eine Belegung, in der im Erwartungswert mindestens  $(3/4) \cdot \text{OPT}$  Klauseln gleichzeitig erfüllt sind.



**Fakt** Es gibt **kein PTAS**, wenn  $P \neq NP$ .