RANDOMISIERTE ALGORITHMEN II.

Entwurfsmethoden

D. DIE PROBABILISTISCHE METHODE

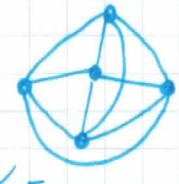
Wonum geht's?

- Wir wollen die Existenz eines bestimmten (mathematischen) Objekts zeigen (z.B. eines Graphen mit bestimmten Eigenschaften).
- Wir werden nicht unbedingt so ein Objekt finden und vorzeigen können.

Gibt es Graphen mit nur sehr kleinen Cliquen und nur sehr kleinen unabhängigen Knotenmengen?

Was bedeutet diese Frage?

eine Clique:



K_5

eine unabhängige Menge:

$\overline{K_5}$

Gibt es große Graphen, die keinen K_5 als Teilgraph enthalten?

→ natürlich können wir einen Graphen mit sehr wenigen (oder keinen) Kanten konstruieren, aber sie werden dann große unabhängige Mengen, insbesondere ein $\bar{K}_5$ enthalten.

Die Frage ist: wie groß kann ein Graph sein, ohne weder ein K_5 noch ein $\bar{K}_5$ zu enthalten?

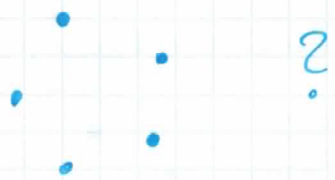
→ Kleines Beispiel: Wir wollen K_3 und $\bar{K}_3$ ausschließen. Wie groß darf unser Graph sein?

→ $n = 4$ Knoten; eine blaue Kante steht für $\{u,v\} \in E$
eine rote Kante bedeutet $\{u,v\} \notin E$

Jede Kante in K_4 soll blau oder rot gefärbt werden, so dass weder ein blaues Dreieck ($= K_3$) noch ein rotes Dreieck ($= \bar{K}_3$) entsteht.

→ für $n = 4$ gibt es viele Lösungen

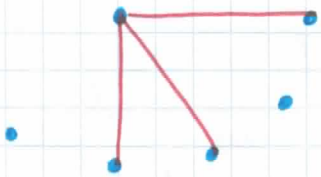
→ $n = 5$



→ $n = 6$ Bei 6 Knoten werden wir scheitern ...
es gibt immer ein K_3 oder ein $\bar{K}_3$ im Graph

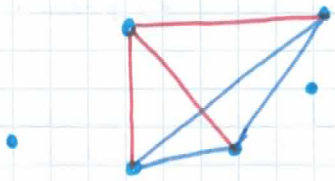
Wamm?

- Ein beliebiger fixierter Knoten hat 5 inzidente Kanten, deshalb haben mindestens 3 inzidente Kanten die gleiche Farbe, o.B.d.A rot



- damit kein rotes Dreieck entsteht, müssen die Kanten zwischen diesen 3 Nachbarn jeweils blau sein

→ so entsteht aber ein blaues Dreieck



Größeres Beispiel:

→ Gibt es Graphen mit 10 Knoten, die weder ein K_5 noch ein $\bar{K}_5$ als Teilgraph enthalten?

→ Wir verwenden die sog. probabilistische Methode um diese Frage zu beantworten:

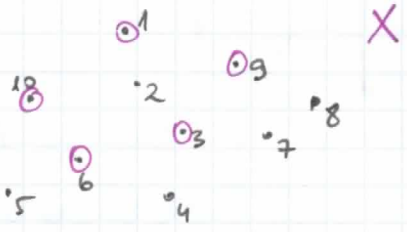
- Wir Betrachten Zufallsgraphen über 10 nummerierten Knoten $V = \{1, 2, 3, \ldots, 10\}$ ((d.h. wir unterscheiden die Knoten voneinander))

- Für jede (Knotenpaar) potentielle Kante $\{u, v\}$ werfen wir eine Münze, und setzen $\{u, v\} \in E$ falls Kopf, bzw. $\{u, v\} \notin E$ falls Zahl, also jeweils mit Prob $= \frac{1}{2}$

R 59.

- wir schätzen die Wahrscheinlichkeit nach oben ab, dass der Zufallsgraph ein K_5 oder ein $\overline{K}_5$ enthält.

- Sei $X \subset \{1,2,3,\dots,10\}$ mit $|X|=5$ eine fixierte Menge. Was ist die Wahrscheinlichkeit, dass X eine Clique wird?
(weitere Kanten sind erlaubt)



X hat $\binom{5}{2} = \frac{5\cdot 4}{2} = 10$ potentielle Kanten; (Knotenpaare)

X wird eine Clique, wenn all diese Kanten einen „Kopf" werfen:

$$\text{Prob}(X \text{ Clique}) = \left(\frac{1}{2}\right)^{10}$$

X wird eine unabhängige Knotenmenge $\overline{K}_5$ wenn all diese Kanten eine „Zahl" werfen mit der Münze.

$$\text{Prob}(X \text{ Unabhängige Menge}) = \left(\frac{1}{2}\right)^{10}$$

$$\text{Prob}(X \text{ Clique oder unabhängig}) = \frac{2}{2^{10}} = \frac{1}{2^9}$$

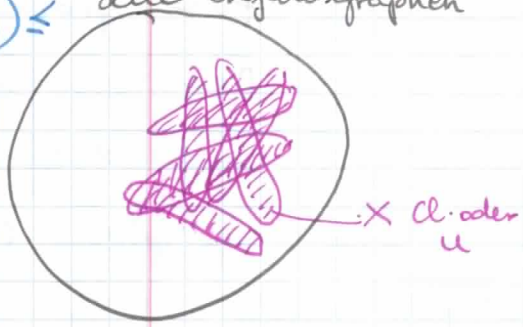
(sich ausschließende Ereignisse)

→ Wie viele Mengen X gibt es mit $|X|=5$?

$$\binom{10}{5} = \frac{10\cdot 9\cdot 8\cdot 7\cdot 6}{5\cdot 4\cdot 3\cdot 2\cdot 1} = \frac{10!}{5!\,5!}$$

Prob (mindestens eine X Clique oder unabhängig) $\leq$ ✓

$$\leq \binom{10}{5} \cdot \frac{1}{2^9} = \frac{10 \cdot 9 \cdot 8 \cdot 7 \cdot 6}{5 \cdot 4 \cdot 3 \cdot 2 \cdot 2^9} = \frac{9 \cdot 7}{2^7} = \frac{63}{128}$$



– also, es existieren Graphen mit 10 Knoten ohne K_5 und $\bar{K}_5$, weil ein zufällig ausgewürfelter Graph mit positiver Wahrscheinlichkeit $\left(> \frac{65}{128}\right)$ ein solcher Graph sein wird!

Es gilt sogar: ein Zufallsgraph wird mit Prob $> \frac{1}{2}$ diese Eigenschaft haben, d.h. die Mehrheit der Graphen wird sogar gut!

Angenommen, dass wir einen beliebigen gegebenen Graphen schnell prüfen können, haben wir sofort einen Las Vegas Algorithmus.

Beachte: ob das gewünschte Objekt (hier: ein Graph mit $|V| = 10$, ohne K_5 und $\bar{K}_5$) existiert, hängt nicht von den Münzwürfen ab! Mathematisch ist die Methode äquivalent mit dem Zusammenzählen aller Graphen und (eine obere Schranke) der schlechten Graphen. Mit Wahrscheinlichkeiten ist die Berechnung einfacher.

Wir haben ein Beispiel mit $n = 10$ und $k = 5$ gesehen.

Unsere Frage allgemein: Sei n beliebig. Wie klein darf k sein, damit noch ein Graph mit n Knoten existiert, der kein K_k und kein $\bar{K}_k$ als Teilgraphen enthält?

R61.

Für große n ist diese Frage sehr schwer. Der genaue Wert von k ist nicht bekannt, und die explizite Konstruktion von solchen Graphen ist noch schwieriger bzw. ungelöst.

Behauptung: Wir wissen aber, dass es solche Graphen gibt für $k = 2 \cdot \log n$. (ziemlich kleine k für n Knoten!)

Beweis: Wir verallgemeinern die obigen Berechnungen für n und $k = 2\log_2 n$. Wir verwenden also wieder die probabilistische Methode:

→ für eine fixierte Knotenmenge $X \subseteq V$ $|X| = k$, die Wahrscheinlichkeit

$$\text{Prob}(X \text{ Clique oder unabhängig}) = 2 \cdot \left(\frac{1}{2}\right)^{\binom{k}{2}}$$

weil es $\binom{k}{2}$ potenzielle Kanten innerhalb X gibt

→ die Anzahl verschiedener Mengen X der Größe k ist $\binom{n}{k}$

→ $\text{Prob}(\text{mindestens ein } X \text{ Clique oder unabhängig}) \leq$

$$\leq \binom{n}{k} \cdot \frac{2}{2^{\binom{k}{2}}} \leq \frac{n^k}{k!} \cdot \frac{2 \cdot 2^{\frac{k}{2}}}{2^{\frac{k^2}{2}}} =$$

(weil $2^{\binom{k}{2}} = 2^{\frac{k(k-1)}{2}} = 2^{\frac{k^2}{2} - \frac{k}{2}} = \frac{2^{\frac{k^2}{2}}}{2^{\frac{k}{2}}}$

und $\binom{n}{k} = \frac{n(n-1)\cdots(n-k+1)}{k!} \leq \frac{n \cdot n \cdot n \cdots n}{k!} = \frac{n^k}{k!}$)

$$= \frac{n^k}{k!} \cdot \frac{2 \cdot 2^{\frac{k}{2}}}{n^k} < \frac{n^k}{k!} \cdot \frac{k!}{n^k} = 1$$

(weil $2^{\frac{k^2}{2}} = 2^{\frac{2 \cdot \log n \cdot 2 \cdot \log n}{2}} = 2^{\log n \cdot 2 \cdot \log n} = n^{2 \log n} = n^k$

und $2 \cdot 2^{\frac{k}{2}} < k!$ für $k \geq 3$)

→ Deshalb gilt

$$\text{Prob}\,(\underline{\text{kein}}\ X \text{ der Größe } k \text{ wird } K_k \text{ oder } \bar{K}_k) > 0$$

⇒ ein zufällig konstruierter Graph wird mit positiver Wahrscheinlichkeit die gewünschte Eigenschaft haben!

□

- Um nur die Existenz nachzuweisen, braucht man nur Prob > 0 zu zeigen. Damit ein (randomisierter) Algorithmus so einen Graphen findet, braucht man effiziente Nachweisbarkeit der Eigenschaft.

[In manchen Fällen kann die probabilistische Methode benutzt werden für den Entwurf eines deterministischen Algorithmus.]
(S. Motwani S. 101)

E. RANDOMISIERUNG IN KONFLIKTSITUATIONEN

Byzantine Agreement (das Problem der Byzantinischen Generäle)

(Gruppenentscheidung treffen trotz störenden Teilnehmern)

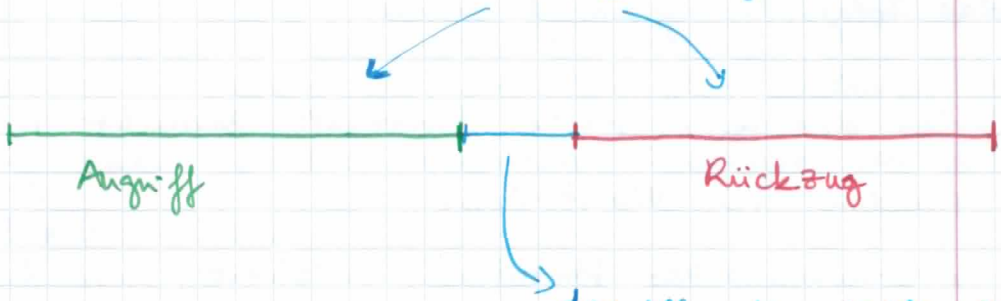
- jede der n Divisionen der Armee von Byzanz wird von einem loyalen oder illoyalen General geführt;
  die Anzahl t der illoyalen Generäle ist höchstens $t < \frac{n}{3}$
- die Generäle müssen sich auf Angriff oder auf Rückzug einigen; sie kommunizieren miteinander direkt über Boten (also immer von einem General zu einem anderen General);
- sie führen eine Abstimmung über mehrere Kommunikationsrunden durch; in einer jeden Runde schickt jeder General zu jedem anderen General sein aktuelles Votum (Angriff oder Rückzug);
- jeder General sieht nur die Voten der anderen Generäle geschickt an ihn selbst; es besteht die Möglichkeit, dass ein (illoyaler) General an verschiedene Generäle unterschiedliche Voten schickt (vor jeder Runde können sich die illoyalen Generäle sogar absprechen: wer was wem schickt ...)
- das Ziel eines jeden loyalen Generals ist: Einigung
  das Ziel eines jeden illoyalen Generals ist: Einigung verhindern

Was soll erreicht werden? (Forderungen)

① Alle loyalen Generäle sollten die selbe Entscheidung treffen
(und das nach möglichst wenigen Runden!)

Illustration: Warum können sie nicht sofort in einer Runde eine Mehrheitsentscheidung (der loyalen Generäle) treffen?

Was wenn die Voten der loyalen Generäle zB. so verteilt sind:



die Illoyalen sagen manchen Anderen „Angriff" und manchen Anderen „Rückzug"

⇒ die „Mehrheitsentscheidung" könnte für verschiedene loyale Generäle unterschiedlich aussehen — und das nach beliebig vielen Runden! ① wäre nicht erfüllt

---

Es wird in ① nicht gefordert, dass die getroffene Entscheidung die tatsächliche Mehrheit (innerhalb der Loyalen) widerspiegelt! Man verlangt nur irgendeine einheitliche Entscheidung der Loyalen. Für sowas gäbe es aber ein triviales Protokoll: z.B. ~~stimme immer für A~~

triff die Entscheidung „Angriff" nach einer Runde!

ABER: (sowas nennen wir keine Gruppenentscheidung / Abstimmung)

R 65.

Damit die Entscheidung die Meinungen der Generäle (einigermaßen) wiedergibt, wird noch gefordert, dass

(2.) Wenn alle logalen Generäle mit der selben Meinung starten, dann sollen sie sich für diese Meinung entscheiden (als Endergebnis).

Die Forderungen (1) + (2) scheinen bescheiden und leicht erfüllbar, aber für <u>deterministische</u> Protokolle kann das Folgende gezeigt werden:

<u>Theorem</u>: Jedes deterministische Protokoll braucht mindestens $t+1$ Runden im Worst-Case.
Ein deterministisches Protokoll existiert, das höchstens $t+1$ Runden benötigt.

<u>Formalisierung des Problems für verteilte Systeme</u> (distributed systems)
(<u>kein</u> Routingproblem!)

- gegeben ist ein Netzwerk von n Prozessoren, wobei $t < \frac{n}{8}$ Prozessoren fehlerhaft sind (und versuchen die Berechnung bösartig zu stören)
- die Prozessoren starten mit jeweils einem Eingabebit

(1.) die fehlerfreien Prozessoren müssen sich am Ende auf dasselbe Bit einigen

(2.) wenn alle fehlerfreien Prozessoren dasselbe Eingabebit besitzen, müssen sie sich letztendlich auf dieses Bit einigen

Die Regel der Kommunikation

- die Kommunikation geht in Runden
- in jeder Runde schickt jeder Prozessor irgendein/sein (aktuelles) Votum an jeden anderen Prozessor (also ein Bit)
  - er kann (wird) in unterschiedlichen Runden unterschiedliche Voten haben (natürlich)
  - er kann innerhalb einer Runde unterschiedliche Bits an unterschiedliche Empfänger ausschicken
- vor jeder Runde können sich die fehlerhaften Prozessoren auf eine beliebig komplexe Strategie von Voten in dieser Runde einigen
- die fehlerfreien Prozessoren kennen die fehlerhaften Prozessoren und ihre Strategren nicht.

Warum wird sogar bösartige Kollaboration der fehlerhaften Prozessoren angenommen? → Ein Protokoll, das sich gegen koordinierte Angriffe wehren kann, kann sich wahrscheinlich auch gegen zufällige Fehler wehren.

Ein randomisiertes Protokoll für Byzantine Agreement

- das Eingabebit für Prozessor i ist b_i
- in jeder Runde s steht ein globales Zufallsbit $\Gamma(s)$ zur Verfügung (also, dasselbe Bit für jeden Prozessor)
- nur die Aktionen der fehlerfreien Prozessoren werden bestimmt
- $v_i(s)$ bezeichne das Votum von Prozessor i in Runde s (die fehlerfreien schicken jedem dasselbe Votum $v_i(s)$ in Runde s)

R 67.

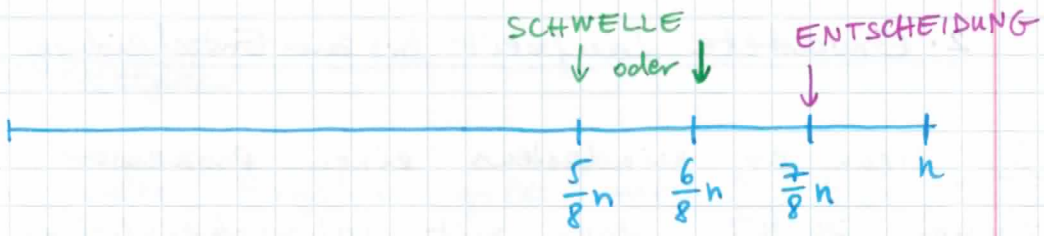
Protokoll für Prozessor i

- setze $v_i(1) = b_i$ (das Eingabebit ist das erste Votum)
- $s := 1$
- WIEDERHOLE

  - schicke $v_i(s)$ an alle Prozessoren
  - erhalte das globale Zufallsbit $T(s)$ und setze $\text{SCHWELLE} = \begin{cases} \frac{5}{8}n & \text{falls } T(s)=0 \\ \frac{6}{8}n & \text{falls } T(s)=1 \end{cases}$
  - empfange $v_j(s)$ von allen anderen Prozessoren $j \neq i$ und sei Bit_i die Mehrheit der Stimmen, und Stimmen_i die Anzahl der Stimmen mit Bit_i (diese hängen vom Empfänger i ab!)
  - IF $\text{Stimmen}_i \geq \text{SCHWELLE}$ setze $v_i(s+1) = \text{Bit}_i$
    ELSE setze $v_i(s+1) = 0$
  - $s := s+1$

BIS $\text{Stimmen}_i \geq \frac{7}{8}n$ (endgültige Entscheidung für Prozessor i)
danach setze $v_i(s+r) := v_i(s) = \text{Bit}_i$

Beobachtung: Die Anzahl der erhaltenen 1Bits (0Bits) von Prozessor i und Prozessor j $(j \neq i)$ kann sich in einer Runde höchstens um t unterscheiden (und $t < \frac{n}{8}$).
Es gilt auch, dass $|\text{Stimmen}_i - \text{Stimmen}_j| \leq t$.
↓
(nicht ganz trivial da sie sich auf unterschiedliche Bits beziehen können)
((jedoch nur in irrelevanten Fällen))



Analyse 1. Korrektheit des Protokolls

Behauptung 1. Wenn in irgendeiner Runde alle fehlerfreie Prozessoren dasselbe Bit schicken, dann entscheiden sich alle in dieser Runde. (Insbesondere wird die Forderung ② erfüllt: wenn alle mit demselben Bit starten, entscheiden sie sich für dieses Bit.)

(Beweis: es gibt $n - t \geq \frac{7}{8}n$ fehlerfreie Prozessoren, und somit wird $\text{Stimmen}_i \geq \frac{7}{8}n$ für alle, und sie legen sich auf dieses Bit fest.)

Behauptung 2. Wenn sich mindestens einer in Runde s entscheidet (festlegt), dann entscheiden sich alle spätestens in Runde $s+1$ für dasselbe Bit. (Insbesondere wird Forderung ① erfüllt.)

(Beweis: Wenn $\text{Stimmen}_i \geq \frac{7}{8}n$, dann $\text{Stimmen}_j \geq \frac{6}{8}n \geq \text{SCHWELLE}$ für alle andere (für den selben Bit, d.h. $\text{Bit}_j = \text{Bit}_i$), und sie alle schicken in der nächsten Runde Bit_i, und alle legen sich fest laut Behauptung 1.)

2. Erwartete Laufzeit bis zur Entscheidung

Behauptung 3. Wenn es mindestens einen Prozessor i gibt mit Stimmen$_i \geq \frac{6}{8}n$, dann mit Wahrscheinlichkeit $\geq \frac{1}{2}$ schicken alle in der nächsten Runde dasselbe Bit aus (und treffen somit eine Entscheidung).

Beweis: dann ist für jeden Prozessor j

Stimmen$_j \geq \frac{5}{8}n$, und mit $p = \frac{1}{2}$ wird

SCHWELLE $= \frac{5}{8}n$, also schicken alle in der nächsten Runde dasselbe Bit (Mehrheitsbit).

Behauptung 4. Wenn es **keinen** Prozessor gibt mit Stimmen$_i \geq \frac{6}{8}n$, dann mit Wahrscheinlichkeit $\geq \frac{1}{2}$ schicken alle in der nächsten Runde Bit$_i = 0$ aus (und treffen somit eine Entscheidung).

Beweis: Wenn SCHWELLE $= \frac{6}{8}n$, wird Stimmen$_i <$ SCHWELLE und 0 wird geschickt.

$\Rightarrow$ in beiden Fällen (also in **jeder** Runde) haben wir Wahrscheinlichkeit $\geq \frac{1}{2}$, dass spätestens in der **nächsten** Runde eine Entscheidung fällt. Es folgt:

Theorem: Die erwartete Anzahl der Runden bis zur Entscheidung ist höchstens 3.

Beweis: Nenne es „Erfolg" wenn in der **nächsten** Runde alle dasselbe Bit schicken. Laut Behauptungen 3-4, hat Erfolg Wahrscheinlichkeit $\geq \frac{1}{2}$ in jeder Runde

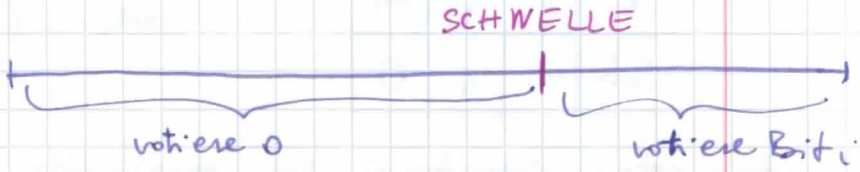
$$E[\text{Laufzeit}] = E[\underset{\text{eintritt}}{\text{Erfolg}}] + 1 = E^{\frac{1}{2}}_{\text{geom}} + 1 = 2 + 1 = 3.$$

Bemerkungen:

→ Das selbe Ergebnis gilt mit $\frac{n}{2}+t+1$, $\frac{n}{2}+2t+1$, $\frac{n}{2}+3t+1$,
statt $\frac{5}{8}n$, $\frac{6}{8}n$, $\frac{7}{8}n$.

→ Warum war hier ein randomisiertes Protokoll erfolgreicher als deterministische Protokolle?

Jeder Prozessor votiert in der nächsten Runde mit Bit_i falls $Stimme_i \geq$ SCHWELLE, und mit 0 falls $Stimme_i <$ SCHWELLE



- Nehmen wir $Bit_i = 1$ an.
Eine einheitliche Entscheidung wird verhindert, wenn für viele i gilt $Stimme_i >$ SCHWELLE und für viele i $Stimme_i <$ SCHWELLE
- die fehlerhaften können diese Situation ggf. erreichen, falls sie den Wert SCHWELLE kennen
- Wenn SCHWELLE zufällig gewählt wird nachdem die Prozessoren ihre Stimmen $v_i(s)$ rausgeschickt haben, kennen sie SCHWELLE nicht, und können die obige Situation mit Prob $\geq \frac{1}{2}$ nicht erzeugen

(Wenn die fehlerhaften Prozessoren die Folge $\tau(s)$ $s=1,2,\ldots$ von Zufallsbits von vornherein kennen würden, und die Eingabe b_i entsprechend gewählt wird, können sie eine Einigung verhindern (s. Aufgabe 32. im Skript))

R 71.

Weitere Erklärung zur Randomisierung in Konfliktsituationen

Beispiel 1. ein 2-Personen Spiel: Stein-Papier-Schere

1 Runde wird gespielt von Alice und Bob

Die folgende Gewinntabelle zeigt die „Auszahlung" an Alice in Abhängigkeit von den gewählten Strategien der einzelnen Spieler

reine Strategien für Bob

mögliche reine Strategien für Alice

A \ B	St	P	Sch
St	0	−1	1
P	1	0	−1
Sch	−1	1	0

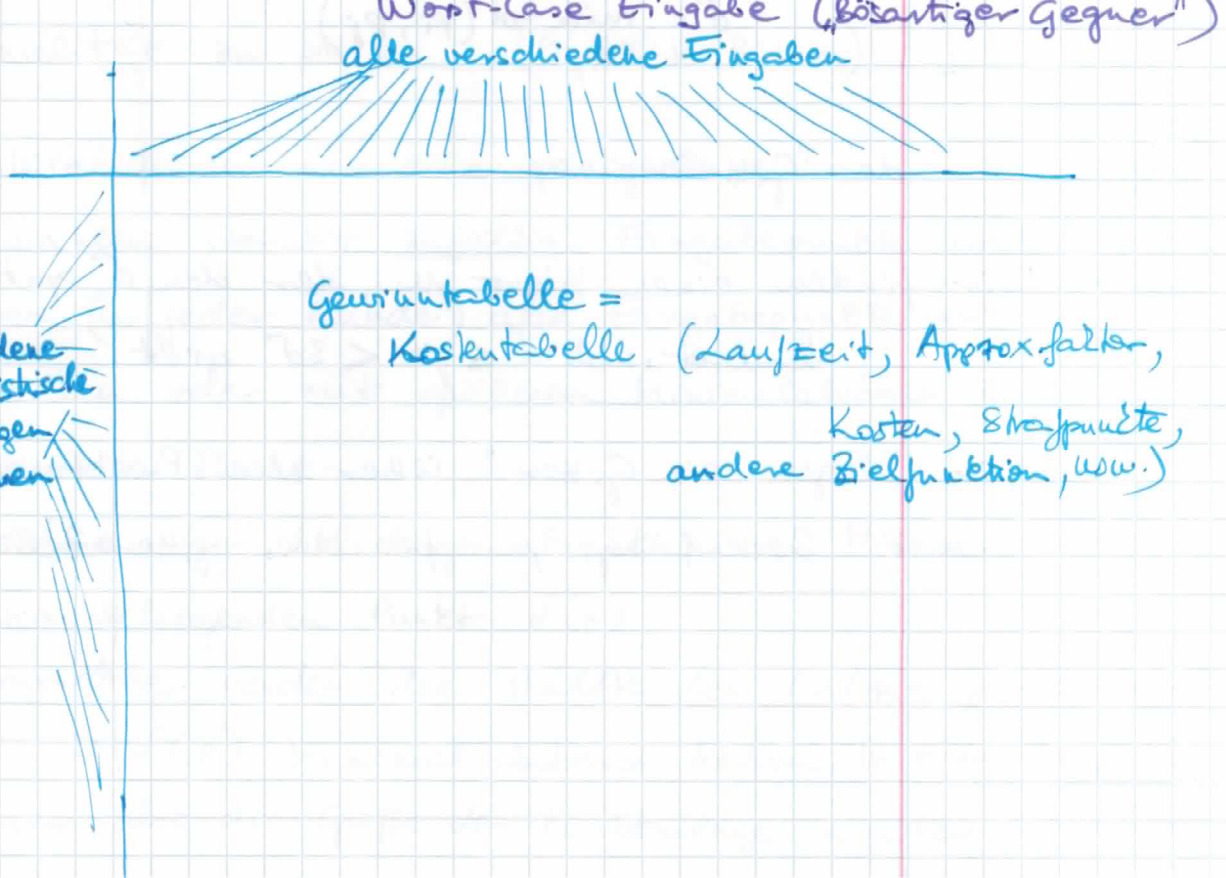
- Gegen jede reine Strategie von Alice (z.B. „Schere") hat Bob eine Gegenstrategie (z.B. „Stein") so dass er gewinnt, d.h. die Auszahlung an Alice −1 wird.
- Gegen die folgende sog. gemischte Strategie: „Stein, Papier oder Schere mit jeweils Prob = $\frac{1}{3}$" ergibt jede reine oder gemischte Strategie von Bob die erwartete Auszahlung 0. („gemischt" bedeutet aber nicht unbedingt „gleichverteilt"!)
- Wir haben angenommen, dass der Gegner (Bob) die gewählte Strategie von Alice kennt, und dazu seine beste Strategie auswählt, was in diesem Fall (da dies ein sog. Nullsummenspiel ist) jeweils die schlechteste Möglichkeit für Alice ist, also „worst-case".
- Jedoch wenn Alice's Strategie darin besteht, zufällig gleichverteilt eine Zeile der Tabelle auszuwählen, und die Kenntnis dieser Strategie von Bob der Kenntnis der Wahrscheinlichkeiten $(\frac{1}{3}, \frac{1}{3}, \frac{1}{3})$ entspricht, kann er im Worst-case nur erwarteten Gewinn 0 (statt −1) für Alice erzwingen.

Beispiel 2. **Die Analyse eines randomisierten Algorithmus**

kann auch als ein 2-Personen Spiel betrachtet werden. Ein randomisierter Algorithmus A wählt zufällig aus einer Menge von deterministischen Berechnungen (nach gegebener Verteilung). Ein Gegner B wählt eine worst-case Eingabe, also eine die die (erwartete) Laufzeit / Approximationsfaktor / Speicherbedarf, usw. maximiert. Da A zufällig wählt, und B nur die Wahrscheinlichkeitsverteilung über die deterministischen Berechnungen kennt, ist der erwartete Gewinn von B weniger als wenn A ein deterministischer Algorithmus (eine konkrete Berechnung) wäre.

Fazit: Wenn unsere Strategie ist, (nach geeigneter Verteilung) zufällig eine reine Strategie auszuwählen, dann ist unser erwarteter Gewinn gegen einen bösartigen Gegner (also Worst-Case), höher.

→ gegen Gegner sind randomisierte Strategien besser!



F. STICHPROBEN -(Random Sampling)

Grundidee: Eine Stichprobe (also zufällig ausgewählte Elemente) aus einer Population repräsentiert die Population wohl.

Das Closest-Pair Problem

Eingabe: n Punkte in der Ebene

$$P = \{p_1\ p_2\ p_3\ \cdots\ p_n\} \subset \mathbb{R}^2$$

Ausgabe: ein Paar (p_i, p_j) nächstliegender Punkte

Es gibt deterministische Algorithmen für dieses Problem mit $O(n \cdot \log n)$ Laufzeit. Der folgende randomisierte Algorithmus findet ein „Closest-Pair" in linearer erwarteter Laufzeit.

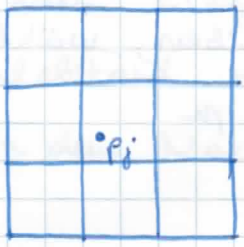
Bezeichne δ den (unbekannten) Minimalabstand, d.h.

$$\delta = \min_{i \neq j} d(p_i, p_j)$$

Idee 1: das Gitter

- Wir finden einen Wert μ, der den δ gut approximiert, konkret: s.d. $\delta \leq \mu < 3\delta$ gilt.
- Wir „legen ein Gitter" über die Punktmenge P mit Seitenlänge μ für die Gitter-Zellen.

- Da $\delta \leq p$, ein Closest-Pair befindet sich in der selben oder in benachbarten Zellen



→ für jeden Eingabepunkt brauchen wir nur die Eingabepunkte in den entsprechenden 9 (eigene und benachbarte) Zellen für ihre Distanz von p_j zu prüfen.

- Da $p < 3\delta$, und δ ist der Minimalabstand, passen nur konstant-viele Eingabepunkte in eine Zelle mit Seitenlänge p

⇒ jeder Punkt p_j hat konstant-viele andere Punkte p_i (wir werden sehen, in unserer Implementierung höchstens 36) in einer Zelle so dass $\delta = d(p_i, p_j)$ möglich ist.

⇒ angenommen, wir haben eine p gefunden, brauchen wir nur $O(n)$ Punktpaare / Distanzwerte nach Minimalität zu prüfen ($9 \cdot 36 \cdot n$ Paare)

Idee 2: Wie finden wir eine geeignete p?

- Wir wählen iterativ zufällig Eingabepunkte, und entfernen (in jeder Runde) alle Eingabepunkte mit dem selben oder mit größerem Minimalabstand als der gewählte Punkt.
- Bezeichne für beliebigen Punkt p den Minimalabstand vom nächstliegenden Punkt $\delta(p)$
- in Erwartung werden die Hälfte der (übrig gebliebenen) Punkte $\geq \delta(p)$ Minimalabstand haben. In Erwartung halbieren wir also die Größe der Punktmenge in jeder Iteration.

R75.

(die $\delta(p)$ des zufällig gewählten Punktes p ist eine Stichprobe aus allen $\delta(\,)$ Werten)

- Falls keine Punkte übrig bleiben, wählen wir die letzte $\delta(p)$ als geeignete μ

- In Erwartung haben wir
  - $\log n$ Runden
  - $\underbrace{O(n) + O(\frac{n}{2}) + O(\frac{n}{4}) + \dots + O(1)}_{O(n)}$ Schritte insgesamt

Der randomisierte Algorithmus

1. Sei $P_0 = P$; $i = 0$;

2. WHILE $P_i \neq \emptyset$ DO

   a.) - wähle $p \in P_i$ zufällig

   - bestimme $\delta_i = \min_{\substack{p_j \in P_i \\ p_j \neq p}} d(p, p_j) = \delta(p)$

   b.) - betrachte ein Gitter mit Seitenlänge $\frac{\delta_i}{3}$

   - ein Punkt heißt <u>isoliert</u> wenn er keinen anderen Punkt in seiner oder in einer Nachbarzelle hat

   - entferne alle isolierten Punkte aus P_i und sei P_{i+1} die restliche Punktmenge
     (alle Punkte mit Mindestabstand δ_i werden entfernt, und manche andere Punkte auch)

   c.) $i := i + 1$

3. Sei $i = i-1$ und somit δ_i der zuletzt berechnete Minimalabstand. Jetzt betrachte das Gitter mit Seitenlänge $\mu = \delta_i$.

Für jeden Eingabepunkt in $P_0 = P$ bestimme den Minimalabstand von anderen Punkten in der selben Zelle und in den Nachbarzellen.

Korrektheit:

Beobachtung 1. Im Schritt 2.b. werden alle Punkte aus P_i mit Mindestabstand $\geq 2 \cdot \frac{\sqrt{2}}{3} \cdot \delta_i = \frac{\sqrt{8}}{3} \delta_i$ aus P_i entfernt, insbesondere alle Punkte mit Mindestabstand $\geq \delta_i$.

Warum?

Mindestabstand $\geq \frac{\sqrt{8}}{3} \delta_i$ passt nicht in eine benachbarte Zelle

→ so ein Punkt wird isolierter Punkt

Beobachtung 2. Im Schritt 3. gilt schon $\frac{\delta_i}{3} < \delta \leq \delta_i$, weil in einem $\frac{\delta_i}{3}$-Gitter alle Punkte isoliert waren (oder jeweils in einem früheren, größeren Gitter). Deshalb $\delta \leq \delta_i < 3\delta$ für die zuletzt berechnete $\delta_i = \mu$.

Beobachtung 3. In eine Zelle der Länge μ passen 36 kleine Zellen der Länge $\frac{\mu}{6}$; jede so kleine Zelle enthält maximal einen Eingabepunkt, weil $\frac{\mu}{6} < \frac{\delta}{2}$ und jedes Punktpaar hat Distanz $\geq \delta$. Jeder Eingabepunkt enthält $\leq 9 \cdot 36$ andere Punkte in der selben oder in benachbarten Zellen der Länge μ.

$\frac{\sqrt{2}}{2} \delta < \delta$

R 77.

Laufzeitanalyse:

- Schritt 3. $9 \cdot 36 \cdot n = O(n)$ (Beobachtung 3.)
- Schritt 2.a $O(|P_i|)$
- Schritt 2.b. $O(|P_i|)$

(Jeder Punkt wird in seine Zelle „gehasht", dann für jeden Punkt die betroffenen 9 Zellen geprüft ob leer.)

Erwartete Laufzeit: $O(n) + O\left(\sum_{i=0}^{\infty} E[|P_i|]\right)$

Was ist $\sum_i E[|P_i|]$?

Intuitiv: P_i wird in Erwartung halbiert in jeder Runde, „deshalb" wird $E[|P_i|] = \frac{n}{2^i}$

$$\sum_{i=1}^{\infty} \frac{n}{2^i} = n$$

Korrekter Beweis:

1. In jeder Iteration werden mindestens die Hälfte der Punkte (isoliert und) entfernt in Erwartung.

Warum? Seien $q_1\, q_2 \ldots q_m$ die Punkte in P_i so dass $\sigma(q_1) \leq \sigma(q_2) \leq \ldots \leq \sigma(q_m)$

Wenn zufällig q_k gewählt wird, dann werden mindestens die Punkte $q_k\, q_{k+1} \ldots q_m$, also $m-k+1$ Punkte entfernt. (Beobachtung 1.)

Die erwartete Anzahl entfernter Punkte ist:

$$\sum_{k=1}^{m} \frac{1}{m}(m-k+1) = \frac{1}{m}\left(m + m-1 + m-2 + \cdots + 1\right) =$$

$$= \frac{1}{m} \cdot \frac{m \cdot (m+1)}{2} = \frac{m+1}{2}$$

Die erwartete Zahl verbleibender Punkte ist $m - \frac{m+1}{2} = \frac{m-1}{2}$

2. $\mathbb{E}[|P_i|] \leq \frac{n}{2^i}$

Beweis durch Induktion: $|P_0| = |P| = n = \frac{n}{2^0}$ ✓

Induktionsschritt: Angenommen $\mathbb{E}[|P_i|] \leq \frac{n}{2^i}$, wir zeigen

$$\mathbb{E}[|P_{i+1}|] \leq \frac{n}{2^{i+1}}$$

$$\mathbb{E}[|P_{i+1}|] = \sum_{m=0}^{\infty} \frac{m-1}{2} \cdot \text{Prob}(|P_i| = m) \leq \frac{1}{2} \sum_{m=0}^{\infty} \text{Prob}(|P_i| = m) \cdot m \leq$$

Satz vom totalen Erwartungswert

$\mathbb{E}[|P_i|]$

$$\leq \frac{1}{2} \cdot \mathbb{E}[|P_i|] \leq \frac{1}{2} \cdot \frac{n}{2^i} = \frac{n}{2^{i+1}} \qquad \square$$

$\Leftrightarrow$

Induktionsannahme

DAS YAO-PRINZIP (Siehe Borodin, Motwani.. Moore-Mertens)

Untere Schranken für randomisierte Algorithmen

Die folgenden Aussagen über randomisierte Algorithmen betreffen in dieser Form Las Vegas Algorithmen, d.h. Algorithmen die in jedem Fall ein korrektes Ergebnis liefern.

→ Was bedeutet „untere Schranke"?

Für einen konkreten Algorithmus A:

> Algorithmus A ist „mindestens so schlecht" im Worst-Case;
> er hat Laufzeit / Approximationsfaktor / Wettbewerbsfaktor /
> Kosten / ... mindestens

Beispiel: LIST Scheduling hat Wettbewerbsfaktor $\geq 2 - \frac{1}{m}$

Least Frequently Used hat Wettbewerbsfaktor $\geq$

Bubble Sort hat Laufzeit $\Omega(n^2)$
im Worst Case

Wie beweist man sowas? Für konkrete Algorithmen fast immer mit einer Worst-Case Instanz für diesen Algorithmus (siehe LIST oder Bubble Sort), evtl. mit einer Folge unendlich vieler Instanzen (siehe LIST für WF ≥ 2, oder LFU)

Hier betrachten wir allgemeinere untere Schranken:

Allgemeine untere Schranken für „alle" Algorithmen haben diese Form:

- Jeder polynomialzeit, deterministische online randomisierte Algorithmus hat im Worst Case mindestens ..... Laufzeit/Kosten/WF. usw.

Beispiel 1. Jeder Polynomialzeit-Algorithmus für BINPACKING benötigt im Worst-Case OPT + 1 Behälter (falls P ≠ NP).
(Man reduziert das NP-vollständige PARTITION Problem auf BINPACKING)

Bei online Algorithmen wird eine Instanz abhängig von Algorithmus definiert:

2. Jeder online Algorithmus für BINPACKING hat Wettbewerbsfaktor $\geq \frac{4}{3}$
(man definiert eine Worst-Case Eingabe deren Länge vom Verhalten des Algorithmus abhängt)

3. Jeder deterministische online Algorithmus hat Wettbewerbsfaktor $\geq k$
(wir haben eine Worst-Case Eingabe definiert mit $k+1$ verschiedenen Seiten, die immer die Seite anfordert, die der Algorithmus eben ausgelagert hat – also die Eingabe wird abhängig vom Algorithmus definiert.)

Im Folgenden beweisen wir eine untere Schranke für den Wettbewerbsfaktor von randomisierten online Paging-Algorithmen; der Beweis verwendet das sog. **Yao-Prinzip**

Theorem 1: Jeder randomisierte online Algorithmus für Paging, mit k Seiten im Cache, besitzt einen Wettbewerbsfaktor $\alpha \geq \sum_{i=1}^{k} \frac{1}{i} = H_k$.

Beachte: Somit ist der Marking-Algorithmus mit $\alpha = 2H_k$ fast optimal.

Beweis:

I. Wenn wir jetzt genau so wie im deterministischen Fall vorgehen würden, dann sollten wir für einen beliebigen randomisierten Algorithmus R eine Eingabefolge σ (Seiten-Anfragen) definieren, so dass für diese Eingabefolge der Algorithmus R in Erwartung mindestens $H_k \cdot OPT(\sigma)$ Seitenfehler hat. Der Algorithmus R ist eine Verteilung über (alle) deterministischen Algorithmen (wie bekannt).

Statt eine Instanz ⟶ für jede Verteilung über det. Algorithmen darf man eine Verteilung über Instanzen ⟶ für alle deterministische Algorithmen

II. Wir wenden stattdessen Yao's Prinzip an, und definieren eine Wahrscheinlichkeitsverteilung Π über mögliche Eingabefolgen, so dass jeder deterministischer Algorithmus D in Erwartung $\geq H_k \cdot OPT$ Seitenfehler hat, (wobei OPT den Erwartungswert der optimalen Anzahl von Seitenfehlern über alle Eingabefolgen bedeutet).

$$OPT = E_{\Pi}[OPT(\sigma)] = \sum_{\sigma} \Pi(\sigma) \cdot OPT(\sigma)$$

↓ Wahrscheinlichkeit der Folge σ als Eingabe.

Beachte, dass im zweiten Fall (II) die Erwartung über die Verteilung Π der Eingaben genommen wird; Im ersten Fall (I) wird die Erwartung über die gewürfelten Zufallsbits von R genommen, also über die Verteilung $p(r)$, wobei $p(r)$ ist die Wahrscheinlichkeit dass durch die Zufallswahlen von R der deterministische Algorithmus (Berechnung) D_r laufen wird.

Das Yao-Prinzip kann für diesen Fall so formuliert werden:

Theorem 2: Wenn für irgendeine Verteilung Π über aller möglichen Eingabefolgen gilt, dass für jeden deterministischen Algorithmus D,

$$\frac{E_{\Pi}[D(\sigma)]}{E_{\Pi}[OPT(\sigma)]} \geq \alpha$$

dann hat jeder randomisierte online Algorithmus (für das Paging Problem) Wettbewerbsfaktor mindestens α.

Wir setzen den Beweis von Theorem 1 so fort:

Wir definieren eine Verteilung Π über Eingaben wie folgt:

Wir brauchen genau $k+1$ verschiedene Seiten

$$\{p_0, p_1, p_2, \ldots, p_k\}$$

Sei n die Länge der Eingabefolge fixiert (und groß genug).

Sei die Verteilung Π so, dass jede n-lange Folge mit den $k+1$ Seiten gleichwahrscheinlich ist (d.h. Folge von n Seiten, jeweils aus $\{p_0 p_1 \dots p_k\}$)

→ die Anzahl aller möglichen Folgen ist: $(k+1)^n$

jede dieser Folgen hat also Wahrscheinlichkeit $\frac{1}{(k+1)^n}$

→ nach einer beliebigen Teilfolge $\sigma_1 \sigma_2 \dots \sigma_{i-1}$ die nächste Seite σ_i wird $p_0, p_1, p_2 \dots$ oder p_k jeweils mit Wahrscheinlichkeit $\frac{1}{k+1}$

→ Sei D ein beliebiger deterministischer online Algorithmus. Wie hoch ist die <u>erwartete Anzahl</u> von Seitenfehlern von D?

Die Prob. eines Strafpunkts für jede neue Anfrage ist $\frac{1}{k+1}$; deshalb ist die Antwort $E_\Pi[D(\sigma)] = \frac{n}{k+1}$

(Mit Indikatorvariablen: sei $X_i = \begin{cases} 1 & \text{falls } \sigma_i \text{ Seitenfehler} \\ 0 & \text{sonst} \end{cases}$

$$E[X_i] = 1 \cdot \frac{1}{k+1} + 0 \cdot \frac{k}{k+1} = \frac{1}{k+1}$$

Anzahl aller Fehler: $D(\sigma) = \sum_{i=1}^{n} X_i$ und $E_\Pi[D(\sigma)] = \sum_{i=1}^{n} E_\Pi[X_i] = \frac{n}{k+1}$)

96.

→ Jetzt betrachten wir die Seitenfehler von OPT, oder oBdA. vom Longest-Forward-Distance (LFD) Algorithmus. OPT kennt die zukünftigen Anforderungen (offline Optimum)

Nach einem Seitenfehler, wann erhält OPT seinen nächsten Seitenfehler?

→ erst wenn alle $k+1$ verschiedene Seiten wieder gefragt wurden!

Wir unterteilen die Eingabefolge $\sigma = \sigma^0\sigma^1\sigma^2\ldots\sigma^i\ldots$ so dass σ^i längstmöglich ist (nach σ^{i-1}) so dass nur k verschiedene Seiten gefragt werden.

(Beispiel $k=4$ $\overbrace{p_1 p_2 p_2 p_3 p_1 p_1 p_2 p_4 p_1 p_2 p_4 p_3}^{\sigma^0}\,\big|\,\overbrace{p_0 p_2 p_3 \ldots}^{\sigma^1\longrightarrow}$)

OPT erhält 1 Strafpunkt in jeder Teilfolge ab σ^1, jeweils für die erste Anfrage der Teilfolge. Die Frage ist somit, wieviele Teilfolgen es gibt in Erwartung. Wir stellen die Frage Anders:

~~Wieviele Teilfolgen gibt es in Erwartung?~~

Wie lang ist eine Teilfolge σ^i in Erwartung?

Nach einem Seitenfehler, die erwartete Folgenlänge, bis alle anderen k Seiten wieder gefragt werden:

$$\frac{k+1}{k} + \frac{k+1}{k-1} + \frac{k+1}{k-2} + \ldots + \frac{k+1}{3} + \frac{k+1}{2} + \frac{k+1}{1} =$$

$$= (k+1)\left(\frac{1}{k} + \frac{1}{k-1} + \frac{1}{k-1} + \ldots + \frac{1}{3} + \frac{1}{2} + 1\right) = (k+1)H_k$$

(Siehe

$\Rightarrow$ es gibt in Erwartung $\frac{n}{(k+1)\cdot H_k}$ Teilfolgen σ^i

$\Rightarrow$ OPT hat in Erwartung $\leq \frac{n}{(k+1)H_k}$ Seitenfehler.

$$E_\pi[OPT(\sigma)] \leq \frac{n}{(k+1)H_k}$$

$\rightarrow$ Es gilt

$$\frac{E_\pi[D(\sigma)]}{E_\pi[OPT(\sigma)]} \geq \frac{\frac{n}{k+1}}{\frac{n}{(k+1)\cdot H_k}} = H_k$$

für diese Verteilung π über Eingaben.

$\rightarrow$ Laut Theorem 2 hat somit jede randomisierte online Strategie für Paging Wettbewerbsfaktor $\geq H_k$

□

Bemerkung 1. Wir können den Algorithmus-Designer und den bösartigen Gegner als zwei Spieler auffassen. Der Algorithmus-Designer sucht „die beste Verteilung über deterministische Algorithmen", während der Gegner die (für ihn/uns) schlimmste Verteilung über Eingaben sucht. Hier erkennt man eine offensichtliche Dualität, und das Yao-Prinzip folgt tatsächlich aus dem Min-Max Theorem der Spieltheorie.*

Bemerkung 2. Es gibt viele Varianten des Yao-Theorems (endlich-unendlich, Minimierungs-Maximierungsprobleme) und im Fall einer Anwendung soll man die genaue Bedingungen beachten.

Eine grobe Begründung für das YAO-Prinzip:

Im Fall von randomisierten Strategien der beiden Spieler (Alg. Designer vs. Bösartiger Gegner) gelten:

(1) Der Spieler der als zweiter seine Strategie wählt, kann genauso gut eine deterministische Strategie wählen.

(weil jede optimale randomisierte Strategie dieses Spielers eine Verteilung über optimalen deterministischen Strategien ist – alle mit dem gleichen erwarteten Zielwert –.

(2) (minimax Theorem) Bei randomisierten Strategien der optimale erreichbare Zielwerte (erwartete Zielwerte) sind gleich, egal welcher Spieler zuerst wählt, also ob a oder b zuerst gewählt wird.

wer wählt zuerst

$$\min_{a} \max_{b} \; a^T \cdot M \cdot b \;=\; \max_{b} \min_{a} \; a^T \cdot M \cdot b$$

- $\min_a$: Verteilungen über det. Algorithmen
- $a^T \cdot M \cdot b$: erwartete Kosten (Laufzeit, Wettbewerbsfaktor usw.)
- $\max_b$: Verteilungen über Eingabe-Instanzen

Yao Prinzip: Die (erwarteten) Zielwerte (z.B. Laufzeiten) sind gleich für

die schlechteste Instanz → für die beste Verteilung über det. Algorithmen

und für

den besten det. Algorithmus → für die schlechteste Verteilung über Instanzen

Warum?

Der erwartete Spielwert ist gleich für

die schlechteste Instanz → für die beste Verteilung über det Algorithmen

|| wegen (1)

die schlechteste Verteilung über Instanzen → für die beste Verteilung über det Algorithmen

|| wegen (2)

die beste Verteilung über det. Algorithmen → für die schlechteste Verteilung über Instanzen

|| wegen (1)

der beste deterministische Algorithmus → für die schlechteste Verteilung über Instanzen