Online Algorithmen

Rolf Klein
Mitschrift von Martin Köhler

Inhaltsverzeichnis

23.04.03

1 Einführung:

Problem: Tür in der Wand finden.

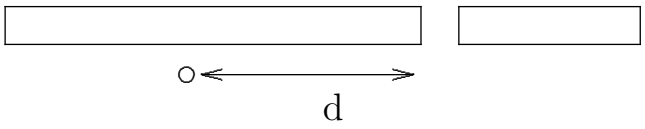


Abbildung 1: Tür in der Wand finden

1. gelaufener Weg $= 2 \cdot \underbrace{(1+2+3+\ldots+d-1)}_{\frac{(d-1)d}{2}} + d = d^2$

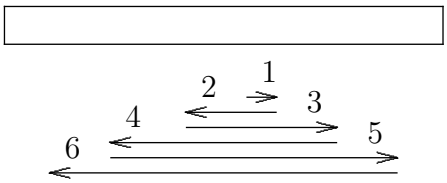


Abbildung 2: Weg um 1 erhöhen

2. Tiefenverdopplung gelaufener Weg $= 2\underbrace{(1+2+2^2+\ldots+2^{n+1})}_{\frac{2^{n+2}-1}{2-1}} + 2^n + \epsilon$

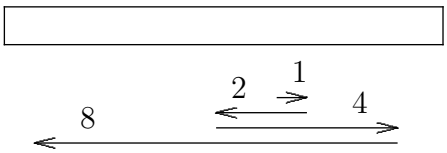


Abbildung 3: Tiefenverdopplung

$= 8 \cdot 2^n - 2 + \underbrace{2^n + \epsilon}_{d} \leq 9 \cdot d$

Theorem 1.1 *Bei Tiefenverdopplung gilt: gelaufener Weg* $\leq 9 \cdot$ *längster Weg + 2*

Definition:

Π ein Problem (Tür in der Wand finden)
$P \in \Pi$ ein Instanz von Π (Konkrete Wand mit konkreter Tür)

OPT (P) optimale Lösung von P (gehe direkt zur Tür)
Strategie S zur Lösung von Π heißt c-kompetitiv:
$\Leftrightarrow \exists A : \forall P \in \Pi : S(P) \leq c \cdot \text{OPT}(P) + A$
Tiefenverdopplung ist 9-kompetitiv
$A \leq 0$: S stark c-kompetitiv

Fragen: gegeben Π, Finde S mit möglichst kleinem c.
Welches ist das kleinste mögliche c?

Theorem 1.2 *(Gal, Baeza, Yater et al.,...) "Tür" in der Wand hat die kompetitive Komplexität 9*

Beweis: 1.2 Angenommen, S ist eine Strategie mit Faktor $c < 9, S = (f_1, f_2, f_3, \ldots)$ Folge von Erkundungstiefen, es muß gelten: $\forall n, \forall \epsilon > 0$:
$2 \cdot (f_1 + \ldots f_{n+1}) + f_n + \epsilon \leq c \cdot (f_n + \epsilon) + \underbrace{A}_{=0}$
$\Rightarrow 2 \cdot (f_1 + \ldots f_{n+1}) + f_n \leq c \cdot f_n$
$\Rightarrow f_1 + \ldots + f_{n-1} + f_{n+1} \leq \underbrace{\frac{c-3}{2}}_{=H<3} \cdot f_n$
$\Rightarrow f_{n+1} \leq H \cdot f_n - \sum_{i=1}^{n-1} f_i$, ebenso $f_n \leq H \cdot f_{n-1} - \sum_{i=1}^{n-2} f_i$
$\Rightarrow f_{n+1} \leq H^2 \cdot f_{n-1} - H \cdot \sum_{i=1}^{n-2} f_i - \sum_{i=1}^{n-1} f_i$
$\leq (H^2 - 1) f_{n-1} - (H+1) \sum_{i=1}^{n-2} f_i$
$\vdots$
$\leq a_m \cdot f_{n-m} - b_m \cdot \sum_{i=1}^{n-1-m} f_i$ mit:
$a_0 = H, b_0 = 1$
$a_{i+1} = a_i H - b_i$
$b_{i+1} = a_i + b_i$

Trick:

Interpretation Matrixmultiplikation
$\begin{pmatrix} a_{i+1} \\ b_{i+1} \end{pmatrix} = \underbrace{\begin{pmatrix} H & -1 \\ 1 & 1 \end{pmatrix}}_{M} \begin{pmatrix} a_i \\ b_i \end{pmatrix} = M^{i+1} \begin{pmatrix} a_o \\ b_0 \end{pmatrix}$
Schreibe $\begin{pmatrix} a_o \\ b_0 \end{pmatrix} = c \cdot w_1 + d \cdot w_2$ wobei w_1, w_2 die Eigenvektoren von M sind.
$\Rightarrow \begin{pmatrix} a_1 \\ b_1 \end{pmatrix} = c \cdot M \cdot w_1 + d \cdot M \cdot w_2 = c \cdot \lambda_1 \cdot w_1 + d \cdot \lambda_2 \cdot w_2$
λ_1, λ_2 Eigenwerte zu w_1, w_2
$\Rightarrow \begin{pmatrix} a_{i+1} \\ b_{i+1} \end{pmatrix} = c \cdot \lambda_1^{i+1} \cdot w_1 + d \cdot \lambda_2^{i+1} \cdot w_2$
Bestimmung der Eigenwerte: Nullstellen vom charakteristischen Polynom $=$
$\det(\lambda \cdot E^1 - M) = \lambda^2 - (H+1)\lambda + H + 1$
$\lambda_{1,2} = \frac{1}{2} \cdot (H+1) \pm \sqrt{\underbrace{(H-3)}_{<0}(H+1) \cdot \frac{1}{4}} \in \mathbb{C}$
$\lambda_2 = \bar{\lambda_1}$
Eigenvektoren w_1, w_2: Lösungen des linearen Gleichungssystem
$M \begin{pmatrix} x \\ y \end{pmatrix} = \lambda_1 \begin{pmatrix} x \\ y \end{pmatrix}$
$\Rightarrow w_2 = \bar{w_1}, w_1 = \begin{pmatrix} 1 \\ \lambda_2 - 1 \end{pmatrix}, w_2 = \begin{pmatrix} 1 \\ \lambda_1 - 1 \end{pmatrix}$
$\Rightarrow w_1, w_2$ linear unabhängig
$\begin{pmatrix} a_{i+1} \\ b_{i+1} \end{pmatrix} = c \cdot \lambda_1^{i+1} \begin{pmatrix} 1 \\ \overline{\lambda_1} - 1 \end{pmatrix} + \bar{c} \cdot \bar{\lambda_1}^{i+1} \overline{\begin{pmatrix} 1 \\ \overline{\lambda_1} - 1 \end{pmatrix}}$
$\Rightarrow a_{i+1} = c\lambda_1^{i+1} + c \cdot \bar{\lambda_1^{i+1}} = 2 \cdot Re(c\lambda_1^{i+1})$
Jede Multiplikation mit λ liefert eine Linksdrehung um Winkel $\in \left(0, \frac{\pi}{2}\right]$
$\Rightarrow$ Es existiert ein i, so daß $c \cdot \lambda_1$ in linker Halbebene
$\Rightarrow$ Realteil$(c \cdot \lambda_1^i) < 0$
$\Rightarrow a_i = \mathrm{Re}(c \cdot \lambda_1^i) < 0$ Widerspruch!
$0 < f_{i+1} \leq a_i \cdot f_i - b_i < 0$ □

Frage:

- Hilft würfeln?
- n-Halbgeraden

[1] Einheitsmatrix

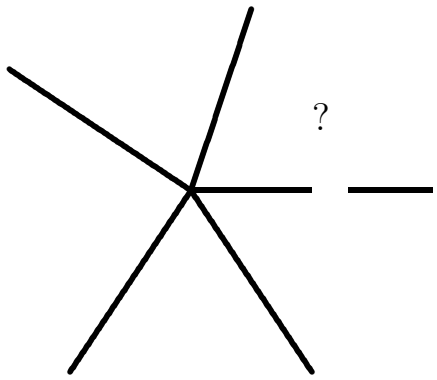


Abbildung 4: n-Halbgeraden

28.04.03 Online-Algorithmen (Strategien): Informationen/Eingabe wird dem Algorithmus erst zur Laufzeit nach und nach mitgeteilt.
Offline-Algorithmen: Kennen alles im Voraus, können optimale Lösung finden.
Online Algorithmus ALG zur Lösung eines Problems Π ist c-kompetitiv, falls gilt: $\exists A$: so dass für alle Instanzen $P \in \Pi$ gilt: $\text{ALG}(P) \leq c \cdot \text{OPT}(P) + A$, wobei ALG (P) die Kosten der Lösung von P mit ALG bezeichnet und OPT (P) die Kosten der optimalen Lösung.
Beispiel: "Tür in der Wand finden":
ALG =Tiefenverdopplung (Doubling) ist 9-kompetitiv, und es gibt keinen besseren (deterministischen) Online Algorithmus für dieses Problem
$\Rightarrow$ 9 ist die Kompetitive Komplexität vom Problem "Tür in der Wand finden"
Weitere Anwendungsbereiche:

- Selbstorganisierende Datenstrukturen
- Paging
- Bezug zur Spieltheorie
- Servertheorie
- Verpacken
- Lastverteilung
- Scheduling
- Routing

- Graphfärbung
- Maschinelles Lernen
- Finanzprobleme
- Navigation für Roboter

Literatur

- Borodin, El-Yaniv - Online Computation and Competitive Analysis, Cambridge, 1998
- Fiat, Woeginger - Online Algorithms: The State of Art, Springer LNCS 1442, 1998

2 Selbstorganisierende Datenstrukturen

2.1 Lineare Listen

(zur Lösung des Wörterbuchproblems)

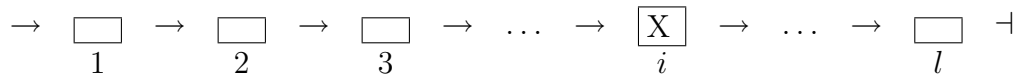


Abbildung 5: Listenmodell

Sequentieller Zugriff, jedes Element kommt höchstens einmal vor.

Statisch:

ACCESS(X) Kosten i, falls X an Position i steht
Kosten $l + 1$, falls X nicht in der Liste vorkommt

Man darf die Liste umorganisieren (um sie für künftige Anfragen effizienter zu machen)

erlaubt:

1. nach erfolgreichem Zugriff auf X darf X um eine beliebige Anzahl von Positionen nach vorne gebracht werden.
   Kosten=0, “Kostenfreie Vertauschung“

2. man darf jederzeit und überall zwei benachbarte Listenelemente vertauschen.
   Kosten=1 “Kostenpflichtige Vertauschung“

Dynamisch:

Zusätzlich INSERT(X): Kosten l+1 (eingefügt wird hinten, wegen Test auf doppeltes Vorkommen)
DELETE(X): Kosten i bzw $l+1$ (wie bei ACCESS)

Situation:

Liste der Länge l gegeben. Es kommt Folge σ von n Zugriffen auf Liste.
OPT kennt σ im Voraus, kann Liste entsprechend organisieren (Es gibt Fälle, in denen OPT kostenpflichtige Vertauschungen benötigt)
Wie? NP-vollständig (Ambühl, 2000)
ALG erhält jeweils nur die nächste Anforderung in der Sequenz $\sigma = \sigma_1\sigma_2 \dots \sigma_i \dots \sigma_n$ und muss sofort reagieren.
Mögliche Kandidaten für ALG :
TRANS (Transponse): Bringe X nach erfolgreichem ACCESS (oder INSERT) um *eine* Position weiter nach vorne (vorsichtig).
MTF (Move to Front): Bringe X nach erfolgreichem ACCESS (oder INSERT) an den Listenanfang (energisch)
FC (Frequency Count): Führt Buch über die erfolgreichen Zugriffe, ordnet Liste entsprechend an. (Gewissenhaft, aufwendig zu implementieren)

Theorem 2.1.1 *(Sleator, Tarjan, 85, Erstes Paper zu Online Analyse)* MTF *ist* $\left(2 - \frac{1}{l+1}\right)$*-kompetitiv bei maximaler Listenlänge* l

Beweis 2.1.1:

Benutzt zwei Techniken:

- Betrachtung der amortisierten Kosten
- Darstellung der amortisierten Kosten durch Potentialfunktionen

Idee:

MTF und OPT starten mit derselben Liste der Länge l, müssen dieselbe Folge $\sigma = \sigma_1\sigma_2 \dots$ bedienen, aber auf unterschiedliche Art.
LOPT_i : Liste von OPT nach Bearbeitung von σ_i
LMTF_i : Liste von MTF nach Bearbeitung von σ_i

Definition:

Amortisierte Kosten von MTF bei Bearbeitung von $\sigma_i : a_i = t_i + \phi_i - \phi_{i-1}$,
wobei t_i = "echte" Kosten (=reine Suchkosten) von MTF bei Bearbeitung von σ_i.
Potentialfunktion ϕ_i: Mißt Ähnlichkeit von LOPT_i und LMTF_i, genauer:
ϕ_i = Anzahl inv(LMTF_i, LOPT_i) der Inversionen der beiden Listen, d.h.
$\text{Inv}(\text{LMTF}_i,\ \text{LOPT}_i) = \{(X, Y), X \text{ steht in } \text{LMTF}_i \text{ vor } Y \text{ und in } \text{LOPT}_i \text{ hinter } Y\}$

Beispiel:

3	15	8	4	7	1	Insgesamt 11 Inversionen (3: 5, 15: 4, 8: 0, 4: 1, 7: 1)
8	1	4	7	15	3	

s_i = reine Suchkosten von OPT bei Bearbeitung von σ_i
P_i = Anzahl kostenpflichtiger Vertauschungen von OPT bei Bearbeitung von σ_i
F_i = Anzahl freier Vertauschungen von OPT bei Bearbeitung von σ_i

Lemma 2.1.2 $a_i \leq (2s_i - 1) + P_i - F_i$

Beweis 2.1.2:

$\sigma_i = \text{ACCESS}(X_j)$, erfolgreich
Situation vor Ausführung von σ_i: Seit ν = Anzahl der Elemente, die in

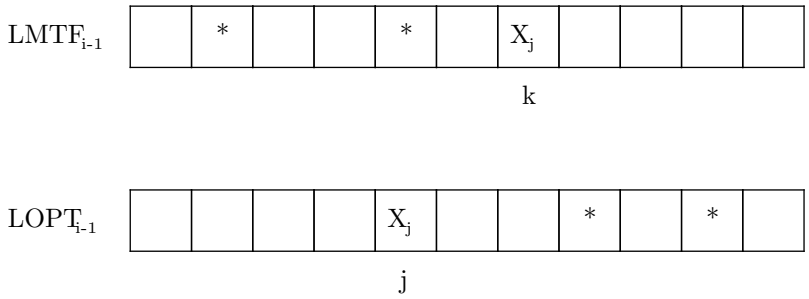


Abbildung 6: Listen von MTF und OPT

LMTF_{i-1} vor X_j stehen und in LOPT_{i-1} dahinter stehen
$\Rightarrow$ von den $k-1$ Elementen, die in LMTF_{i-1} vor X_j stehen, stehen auch in LOPT_{i-1} : $k - 1 - \nu$ vor X_j
$\Rightarrow k - 1 - \nu \leq j - 1$
$\Rightarrow k - \nu \leq j$ (=Suchkosten von OPT)
Vorstellung, erst bearbeitet MTF die Anforderung σ_i, dann OPT
MTF :

- Suchkosten $t_i = k$
- dadurch, daß X_j ganz nach vorne kommt: ν alte Inversionen verschwinden
- $k - 1 - \nu$ neue Inversionen entstehen

$\Rightarrow$ inv(LMTF_i,LOPT_{i-1})-inv(LMTF_{i-1},LOPT_{i-1})$=k - 1 - 2\nu$
$\Rightarrow t_i + k - 1 - 2\nu = 2(k - \nu) - 1 \leq 2j - 1$ Zwischenbilanz.
OPT :

- Suchkosten $s_i = j$
- jede freie Vertauschung von OPT beseitigt eine Inversion (Da X_j in LMTF_i ganz vorn steht.
- jede kostenpflichtige Vertauschung durch OPT schafft ≤ 1 neue Inversionen.

$\Rightarrow$ inv(LMTF_i,LOPT_i) $\leq$ inv(LMTF_i,LOPT_{i-1}) $+P_i - F_i$
$\Longrightarrow a_i = t_i + \phi_i - \phi_{i-1}$
$= t_i$+inv($\text{LMTF}_i, \text{LOPT}_i$)$-$inv($\text{LMTF}_i, \text{LOPT}_{i-1}$)+inv($\text{LMTF}_i, \text{LOPT}_{i-1}$)$-$ inv($\text{LMTF}_{i-1}, \text{LOPT}_{i-1}$)
$\leq 2j - 1 + P_i - F_i = 2s_i - 1 + P_i - F_i$.

30.04.03 MoveTo Front (MTF): Wollen zeigen $\left(2 - \frac{1}{l+1}\right)$-kompetitiv, l=maximale Länge. Hatten definiert: amortisierte Kosten einer MTF -Operation
$a_i = t_i + \phi_u - \phi_{i-1}$, wobei t_i = echte Kosten von MTF (reine Suchkosten)
ϕ_i = Anzahl der Inversionen LMTF_i bzgl. LOPT_i σ_i =ACCESS(X_j) erfolg-

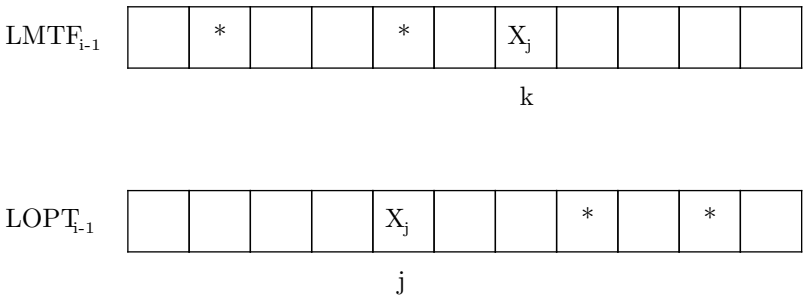


Abbildung 7: Listen von MTF und OPT

reich.

Hatten bewiesen:

Lemma 2.1.2: $a_i \leq 2s_i + P_i - F_i$, wobei:
s_i = Suchkosten von OPT bei der Bearbeitung von σ_i
P_i = Anzahl kostenpflichtiger Vertauschungen von OPT bei der Bearbeitung von σ_i
F_i = Anzahl freier Vertauschungen von OPT bei der Bearbeitung von σ_i

Methode: $a_i = \underbrace{t_i + inv(\text{LMTF}_i, \text{LOPT}_{i-1}) - inv(\text{LMTF}_{i-1}, \text{LOPT}_{i-1})}_{\leq 2j-1 = 2s_i - 1}$
$+ \underbrace{inv(\text{LMTF}_i, \text{LOPT}_i) - inv(\text{LMTF}_i, \text{LOPT}_{i-1})}_{\leq P_i - F_i}$
Falls $\sigma_i = \text{ACCESS}(X_j)$ nicht erfolgreich
$\Rightarrow$ weder MTF noch OPT machen freie Vertauschungen
müssen zeigen: $a_i = \underbrace{t_i}_{=l+1} + \underbrace{\phi_i - \phi_{i-1}}_{\leq P_i} \leq \underbrace{2s_i - 1}_{=2l+1} + P_i - F_i$ $\square$
Falls $\sigma_i = \text{DELTE}(X_j)$ erfolgreich
$\Rightarrow$ keine freien Vertauschungen
$\underbrace{t_i}_{=k} + \underbrace{inv(\text{LMTF}_i, \text{LOPT}_{i-1}) - inv(\text{LMTF}_{i-1}, \text{LOPT}_{i-1})}_{-\nu}$
$\Rightarrow k - \nu = j \leq 2j - 1$ $(j \geq 1)$
$= 2s_i - 1$, Rest analog
Falls $\sigma_i = \text{DELETE}(X_j)$ erfolglos, wie erfolgloses $\text{ACCESS}(X_j)$
Falls $\sigma_i = \text{INSERT}(X_j)$: Wie bei $\text{ACCESS}(X_j)$ mit $k = j = l + 1, \nu = 0$ $\square$

Theorem 2.1.3 *Sei σ eine Folge von n Zugriffen (ACCESS, INSERT oder DELETE). Dann gilt bei gleicher Anfangsliste:*
MTF $(\sigma) \leq 2 \cdot \text{OPT}_S(\sigma) + P - F - n$
OPT $_S$= *Reine Suchkosten von* OPT
P=Kostenpflichtige Vertauschungen von OPT
F=Kostenfreie Vertauschungen von OPT

Beweis 2.1.3:

$$\text{MTF}\ (\sigma) = \sum_{i=1}^{n} t_i \leq \sum_{i=1}^{n} t_i + \underbrace{\phi_n}_{\geq 0} - \underbrace{\phi_0}_{=0} = \sum_{i=1}^{n} (t_i + \phi_i - \phi_{i-1}) = \sum_{i=1}^{n} a_i \leq$$

$$\sum_{i=1}^{n} ((2s_i - 1) + P_i - F_i) = 2 \cdot \text{OPT}_S(\sigma) + P - F - n \ \square$$

Beweis von Theorem 2.1.1:

$\text{MTF}\ (\sigma) \leq \underbrace{2 \cdot \text{OPT}_S(\sigma) + P}_{=2\cdot\text{OPT}(\sigma)} - F - n$

l=maximale Listenlänge
$\Rightarrow \text{OPT}(\sigma) \leq n(l+1)$ (Denn OPT ist sicher besser als der bequeme Algorithmus, der bei jeder Operation bis zum Listenende läuft)
$\Rightarrow n \geq \frac{\text{OPT}(\sigma)}{l+1}$
$\Rightarrow \text{MTF}(\sigma) \leq \left(2 - \frac{1}{l+1}\right) \text{OPT}(\sigma)$ □

Fragen

- Ist das gut?
- Ist MTF übereifrig?
- Ist TRANS besser?

Proposition 2.1.4 TRANS *ist nicht kompetitiv (falls Listenlänge beliebig)*

Beweis 2.1.4: Sei eine Liste mit l Elementen gegeben. Der böse Gegenspieler (adversary) ärgert TRANS und fordert stets ACCESS-Operationen für das letzte Element in LTRANS
$\Rightarrow$ TRANS vertauscht stets die beiden letzten Listenelemente $\Rightarrow \text{TRANS}\ (\sigma) =$

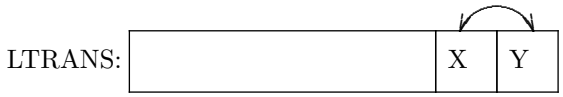


Abbildung 8: Worst Case von TRANS

$2n \cdot l$ bei einer Folge von $2n$ Zugriffen.
Der Gegenspieler OPT bringt zunächst x und y nach vorne,
danach hat OPT Kosten 3 für je 2 Zugriffe auf (x,y)

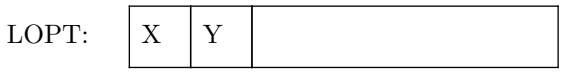


Abbildung 9: Verhalten von OPT

$\Rightarrow \frac{\text{TRANS}(\sigma)}{\text{OPT}(\sigma)} = \frac{2nl}{2(l-2)+3n} \overset{n\to\infty}{\longrightarrow} \frac{2}{3}l$ (d.h. für große l ist TRANS beliebig schlecht)

Theorem 2.1.5 *Jeder deterministische Online-Algorithmus für das statische Listenproblem hat einen kompetitiven Faktor* $\geq 2 - \frac{2}{l+1}$

Beweis 2.1.5 Sei ALG ein solcher Algorithmus. Wenn Gegenspieler n-mal das letzte Element verlangt: $\text{ALG}\ (\sigma) \geq n \cdot l$

Frage: Wie gut kann OPT Folge σ bedienen?

Trick: Definiere Algorithmus A_π wie folgt:

- stellt Permutation π her. Kosten $b \cdot l^2$
- beantworte danach alle ACCESS-Operationen ohne weitere Umstrukturierungen

Betrachte eine einzelne Anforderung ACCESS(X).
Es gibt $(l-1)!$ Permuationen π, bei denen X an Stelle i steht.
$\sum\limits_\pi A_\pi(\text{ACCESS(X)}) = \sum\limits_{i=1}^{l} i(l-1)! = (l-1)!\frac{l(l+1)}{2}$
$\Rightarrow \sum\limits_\pi A_\pi(\sigma) = n \cdot (l-1)!\frac{l(l+1)}{2} + l! \cdot b \cdot l^2$
$\Rightarrow$ Mittelwert: $\frac{1}{l!}\sum\limits_\pi A_\pi(\sigma) = \frac{n(l+1)}{2} + b \cdot l^2$
Trick: Es muss mindestens ein π_0 geben mit $A_{\pi_0}(\sigma) \leq$ Mittelwert
$\Rightarrow \frac{\text{ALG}(\sigma)}{\text{OPT}(\sigma)} \geq \frac{n \cdot l}{\text{OPT}(\sigma)} \geq \frac{n \cdot l}{A_{\pi_0}(\sigma)} \geq \frac{n \cdot l}{n\frac{l+1}{2} + l^2 \cdot b} \overset{n\to\infty}{\longrightarrow} \frac{2l}{l+1} = 2 - \frac{2}{l+1}$ □

Rekapitulation:

- Selbst im statischen Fall ist die Berechnung von OPT NP-vollständig
- MTF ist 2-kompetitiv (auch im dynamischen Fall)
- eine bessere Online-Lösung (deterministisch) gibt es nicht
- Auch in der Offline Version ist keine bessere Approximation von OPT (in polynomieller Laufzeit) bekannt

07.05.03

Frage: Hilft Randomisierung? Folgendes Modell:

Online Algorithmus ("Spieler"): Trifft zur Laufzeit zufällige Entscheidungen nach einer bestimmten Wahrscheinlichkeitsverteilung.

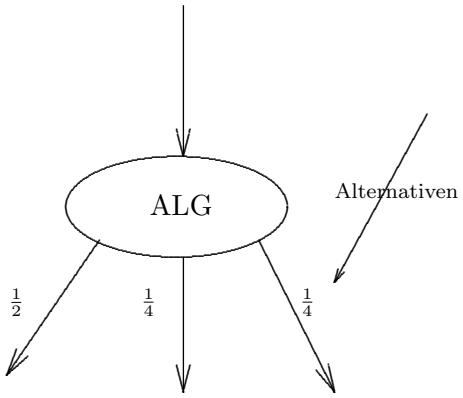


Abbildung 10: Entscheidungsbaum

Vergesslicher Gegenspieler (oblivious adversary): Trifft seine Entscheidungen vor dem Start

- in Kenntnis des Algorithmus und der Wahrscheinlichkeitsverteilung
- ohne Kenntnis der konkreten Entscheidungen des Spielers

Abrechnung: Zu erwartende Kosten des Online Algorithmus gegen Kosten einer optimalen Offline Lösung

Definition: Ein randomisierter Online-Algorithmus ALG heißt c-kompetitiv gegen vergessliche Gegenspieler, falls gilt: $\exists A : \forall P \in \Pi : E^2(\text{ALG}(P)) \leq c \cdot \text{OPT}(P) + A$

Dynamische Listenverwaltung mit randomisiertem Algorithmus bit
statischer Fall: Nur Zugriffe ACCESS(X)

- Rate anfangs für jedes Listenelement X ein Zufallsbit b(X)
  gleichwahrscheinlich 0 oder 1
  unabhängig
  } i.i.d[3]
- Zur Laufzeit (deterministisch) bei ACCESS(X)
  - komplementiere b(X) (d.h. $b(X) = 1 \dot{-} b(X)$)

[2] E = Erwartungswert
[3] independent identically distributed

– falls b(X) jetzt 1, bringe X an den Listenanfang (kostenfreie Vertauschungen, es bleibt b(X)=1)

Theorem 2.1.6 *(Reingold, Westbrock 90'): Für jede Zugriffsfolge σ der Länge n gilt:*
$E(BIT(\sigma)) \leq \underbrace{\frac{7}{4}}_{<2} \text{OPT}(\sigma) - \frac{3}{4}n$

Beweis 2.1.6: Sei σ eine feste Folge von n ACCESS-Operationen. Während BIT die Folge σ bearbeitet, sind aktuellen Werte b(X) i.i.d.

Betrachte zwei Typen von Events:

(i) OPT führt kostenpflichtige Vertauschung aus

(ii) BIT und OPT bearbeiten ACCESS(Y) mit kostenfreien Vertauschungen

Wie vorher: Potentialfunktion ϕ_i, mißt das Verhältnis der Listen LBIT und LOPT nach Bearbeitung des i-ten Events. Intuitiv klar: ϕ muß auch die Bits b(X) berücksichtigen.

Definition:

1. Sei (x, y) eine Inversion von LBIT bezüglich LOPT (d.h. in LBIT steht x vor y, aber in LOPT steht x hinter y). Dann heißt $w(x, y) = (b(y)+1)$ das Gewicht[4] von (x, y). Das Gewicht hängt nur von y ab.

2. $\phi = \sum_{(x,y) \in inv(LBIT,LOPT)} w(x, y)$ (Summe der Gewichte)

3. Trick: Betrachtung der amortisierten Kosten von BIT bei der i-ten Operation
   $a_i = \text{BIT}_i$[5] $+ \phi_i - \phi_{i-1}$

Damit (wie im deterministischen Fall):
$\text{BIT}(\sigma) = \sum_i \text{BIT}_i = \sum_i a_i + \underbrace{\phi_0}_{=0} - \underbrace{\phi_{\text{last}}}_{\geq 0} \leq \sum_i a_i$ (Am Anfang LBIT=LOPT)

[4] Anzahl ACCESS(y) bevor y nach vorne kommt
[5] reale Kosten von BIT

zu zeigen: bei Event vom Typ
(i): $E(a_i) \leq \frac{7}{4}\text{OPT}_i$
(ii): $E(a_i) \leq \frac{7}{4}\text{OPT}_i - \frac{3}{4}$(n-mal)
$\Rightarrow$ Theorem 2.1.6
Ein Event vom Typ (i) (kostenpflichtige Vertauschung durch OPT) schafft maximal 1 neue Inversion, deren Gewicht 1 oder 2 ist, mit derselben Wahrscheinlichkeit.
$\Rightarrow E(a_i) = E(\underbrace{\text{BIT}_i}_{=0}) = E(\underbrace{\phi_i - \phi_{i-1}}_{\leq \frac{1}{2}(1)+\frac{1}{2}(2)}) \leq \frac{3}{2} < \frac{7}{4} = \frac{7}{4} \cdot 1 = \frac{7}{4}OPT_i$[6]
Event vom Typ (ii): (Suchzugriff durch BIT und OPT mit kostenfreien Vertauschungen) Sei I = Anzahl $X : X$ steht in LBIT vor Y und in LOPT

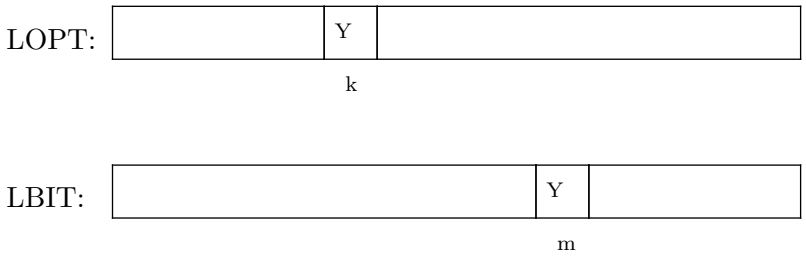


Abbildung 11: Listen von OPT und BIT

hinter Y
= Anzahl $X : (X, Y) \in$ inv(LBIT, LOPT)
wie früher: Von den $m-1$ Vorgängern von Y in LBIT müssen $m-1-I$ auch Vorgänger von Y in LOPT sein
$\Rightarrow m - 1 - I \leq k - 1$
$\Rightarrow \text{BIT}_i = m \leq k + 1 (*)$

Trick: Schreibe $\phi_i - \phi_{i-1} = A + B + C$, wobei
A = Gesamtgewicht aller neuen Inversion.
B = Gesamtgewicht aller entfernten alten Inversionen (≤ 0).
C = Gewichtsänderungen der überlebenden alten Inversionen.
Betrachte zunächst B und C.

falls $b(Y) = 1$**:** BIT bewegt Y nicht, $b(Y) = 0$. OPT kann Y weiter nach vorn bringen und dadurch Inversionen (Y, Z) entfernen. $\Rightarrow B \leq 0$
falls überlebende Inversionen ihr Gewicht verändert haben $\Rightarrow$ sie enthalten

[6]Kosten von OPT bei Bearbeitung dieses Events

y an zweiter Stelle, haben also die Gestalt (X, Y) $\Rightarrow$ Gewicht um 1 kleiner $\Rightarrow C = -I$.

falls $b(Y) = 0$**:** BIT bringt Y an Listenanfang, $b(Y) = 1 \Rightarrow$ Alle alten (X, Y) verschwinden, hatten vorher Gewicht 1. $B = -I, C = 0$, denn keine Inversion, deren Gewicht sich geändert hätte, kann überleben.
$\Rightarrow$ stets $B + C \leq -I(**)$
$\Rightarrow E(a_i) = E(\text{BIT} + A + B + C) \leq E(A) + E(\underbrace{k + I - I}_{**}) = E(A) + k(***)$

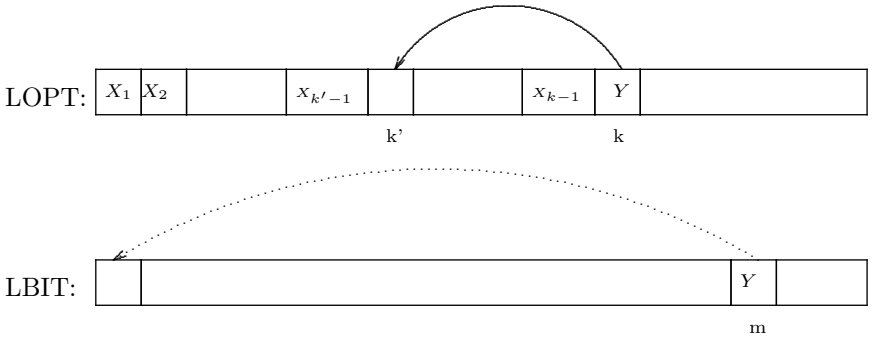


Abbildung 12: Listen LOPT und LBIT

Betrachte jetzt $E(A)$: OPT bringt Y (kostenfrei) an Stelle $k' \leq k$

Zwei Arten neuer Inversionen

- $b(Y) = 0$, Y wird von BIT an Listenanfang gebracht
  $\Rightarrow$ genau die $(Y, X_1), \ldots, (Y, X_{k'-1})$, mit Gewicht $w(Y, X_j) = b(X_j)$[7] $+1$ sind neu.

- $b(Y) = 1$, Y wird von BIT nicht verschoben
  $\Rightarrow$ höchstens $(X'_k + 1, Y), \ldots, (X_{k-1}, Y)$ sind neu und haben Gewicht 1.

$$\Rightarrow E(A) \leq \frac{1}{2}\left(\sum_{j=1}^{k'-1} \frac{1}{2}(1+2)\right) + \frac{1}{2}\left(\sum_{j=k'+1}^{k-1} 1\right) \leq \frac{1}{2}\sum_{j=1}^{k-1}\left(\frac{3}{2}(k-1)\right)$$
$$= \frac{3}{4}(k-1) \Rightarrow E(a_i) \underbrace{\leq}_{***} k + E(A) \leq \frac{7}{4}k - \frac{3}{4} = \frac{7}{4}\text{OPT}_i - \frac{3}{4} \quad \square$$

[7] 0 oder 1, gleich wahrscheinlich

12.05.03

- Man kann auch auf $\frac{8}{5}$ herunterkommen mit: $\frac{4}{5} \cdot \text{BIT} + \frac{1}{5} \cdot \text{TIMESTAMP}$ (S. Alkers '95, determ. 2-kompetitiv)
  - Nach ACCESS(X): bringe X vor das vorderste Y, das nach dem letzten ACCESS(X) erst 1 Mal dran war, falls solch ein Y existiert, sonst lasse X stehen.
- Geht es noch besser?
  Offen.

2.2 Selbstorganisierende Bäume

Idee 1: Statt MoveToFront (nach ACCESS(X)): MoveToRoot!
Müssen Suchbaumstruktur erhalten. Wie?

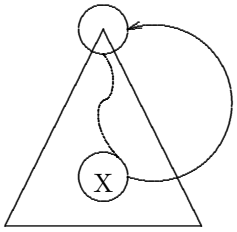


Abbildung 13: MoveToRoot

Idee 2: Durch Rotation.

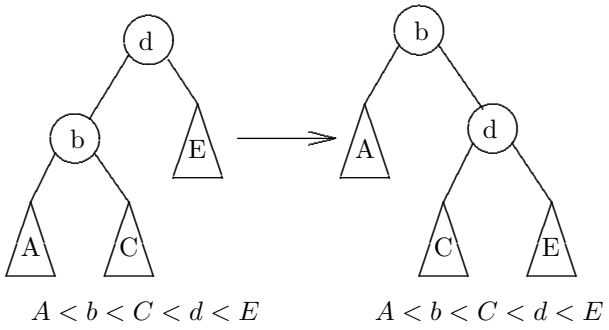


Abbildung 14: Rotation

Idee 3: Wiederhole Rotation um Vater bis X oben steht.

Kostenmodell $\left.\begin{array}{l}\text{Rotation um X kostenfrei}\\ \text{Ansonsten kostet Rotation 1}\end{array}\right\}$ wie bei den Listen.

Theorem 2.2.1 *(Allen, Manuro '78) Sei T Suchbaum mit Schlüsselmenge* $S = \{1, 2, \ldots, 2m\}$ *Dann sind bei der Zugriffsfolge: (ACCESS(1),ACCESS(2), ..., ACCESS(m))*[2] *die mittleren Kosten pro Zugriff in* $\Omega(m)$ *bei Verwendung von* MTR .

Dagegen wäre beim AVL-Baum sogar der Worst case pro Zugriff in $O(\log m)$.

Beweisskizze 2.2.1: Egal mit welchem T man startet, nach $\underbrace{\text{ACCESS}(1), \ldots, \text{ACCESS}(m)}_{\text{"ganzer Durchlauf"}}, \text{ACCESS}(1), \ldots, \text{ACCESS}(i-1)$ sieht der Baum so aus. Wie kommt das? Grund: lange Ketten bleiben unter MTR erhalten.

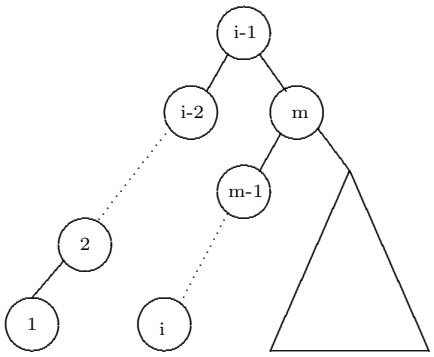


Abbildung 15: MTR nach $m + i - 1$ Schritten

(Siehe Abbildung 15 und 16 auf der nächsten Seite)

Abhilfe (Sleator, Tarjan '85): Rotiere erst am Großvater, dann am Vater. $\longrightarrow$ Splay[8] Trees MTR *. Literatur: Ottman/Widmayer, Algorithmen Datenstrukturen.

- falls Vater(x) existiert
  - falls Großvater(x) existiert
    - Falls Großvater(x), Vater(x), x auf Rechts- oder Linkspfad: Rotiere Großvater(x), Vater(x)

[8] ausgebreitet, gespreizt

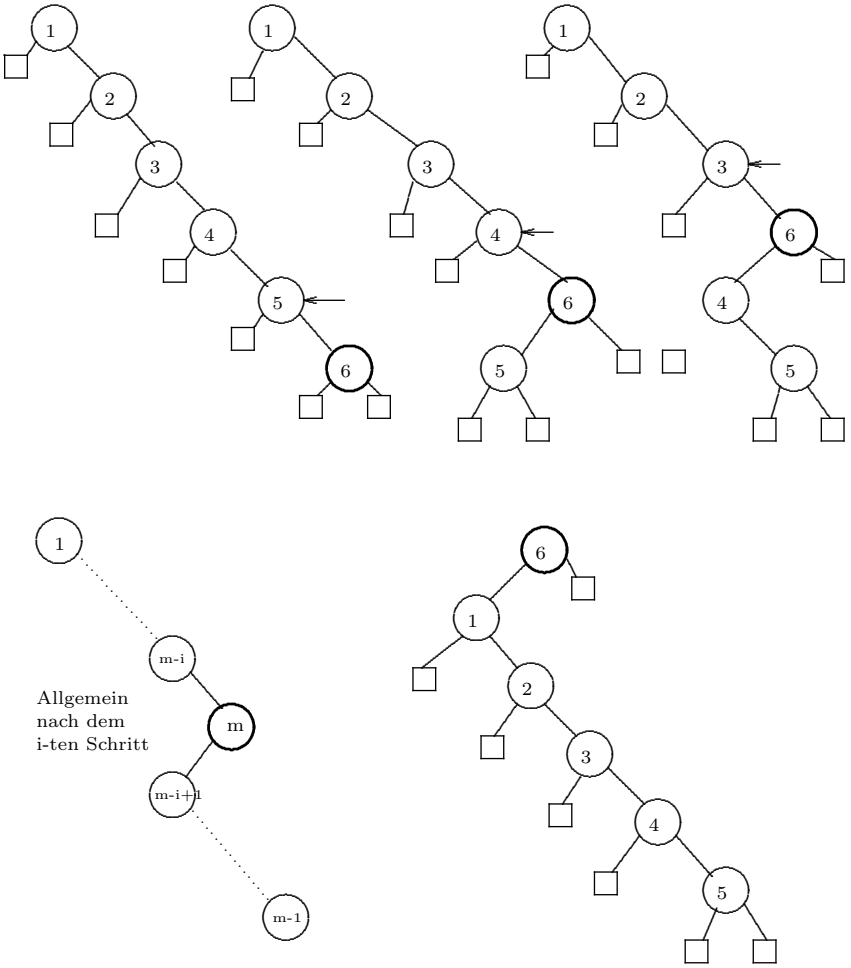


Abbildung 16: MTR(6) bei entartetem Baum

  - sonst rotiere Vater(x), [Groß]vater(x)
- sonst rotiere Vater(x)

Siehe: Abbildung 17 auf der nächsten Seite und 18 auf der nächsten Seite

3 Fälle

- zig-zig[9] Siehe Abbildung 19 auf Seite 22
- zig-zag Siehe Abbildung 20 auf Seite 22
- zig Siehe Abbildung 21 auf Seite 23

[9]links-links oder rechts-rechts

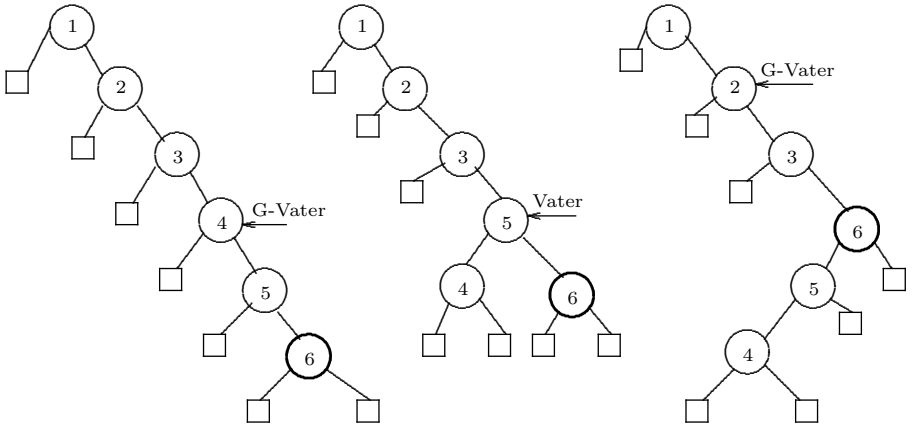


Abbildung 17: MTR*(6) bei entartetem Baum (1)

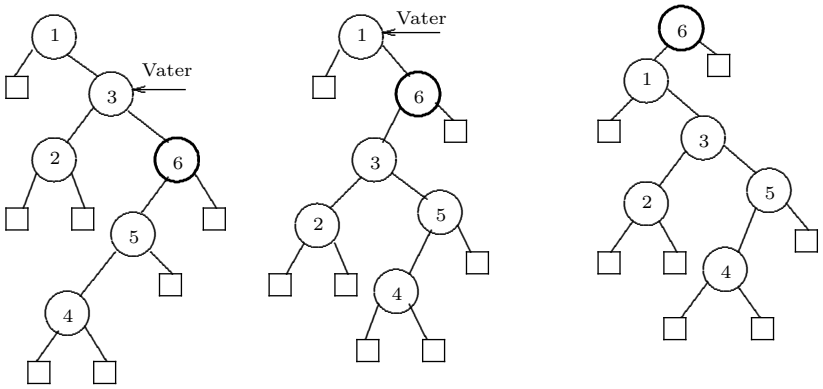


Abbildung 18: MTR*(6) bei entartetem Baum (2)

Achtung! Auch die zig-zig Fälle können Höhe reduzieren

Analyse: Jedes gespeicherte Element i habe das Gewicht $w(i) > 0$ (die $w(i)$ können später noch festgesetzt werden)

Definition x Knoten von T: $S(x) = \sum_{i \in \text{Teilbaum } T_x} w(i)$
$r(x) = \log_2 S(x)$
$\phi(T) = \sum_{x \in T} r(x)$.
Splay(x, T) = echte Kosten des Aufrufs MTR *(x) in T
$A(x, T) = \text{Splay}(x, T) + \phi(\widetilde{T}) - \phi(T)$ amortisierte Kosten von Zugriff auf x.
$\widetilde{T}$ = der aus T durch MTR *(x) entstehende Baum.

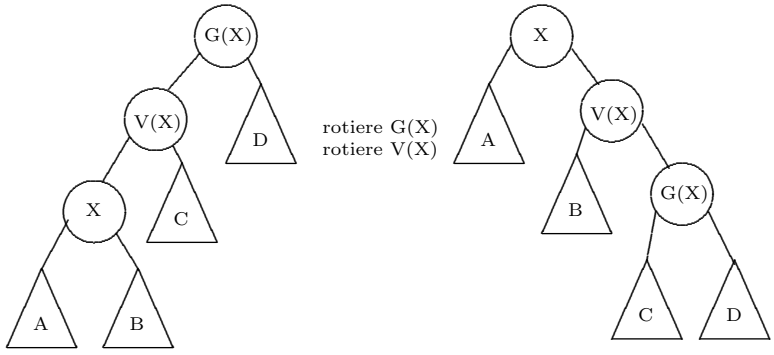


Abbildung 19: zig-zig

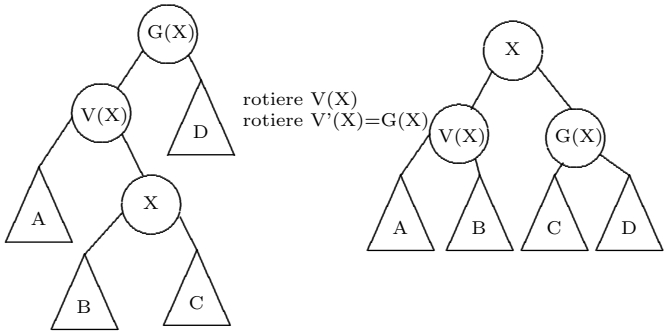


Abbildung 20: zig-zag

Brauchen ein technisches Lemma.

Lemma 2.2.2 *Seien T, T' zwei beliebige (Splay-)Trees über* $\{1, \ldots, n\}$ *und sei* $W = \sum_{i=1}^{n} w(i)$
Dann gilt: $|\phi(T) - \phi(T')| \leq \sum_{i=1}^{n} \log\left(\frac{W}{w(i)}\right)$

Beweis 2.2.2: Seiten x_i und x_i' die Knoten von T und T', die Element i enthalten.

Klar: $w(i) \leq S(x_i), S(x_i') \leq W$
$\Rightarrow \log(w(i)) \leq r(x_i), r(x_i') \leq \log W$
$\Rightarrow \phi(T) - \phi(T') = \sum_{i=1}^{n} (r(x_i) - r(x_i'))$

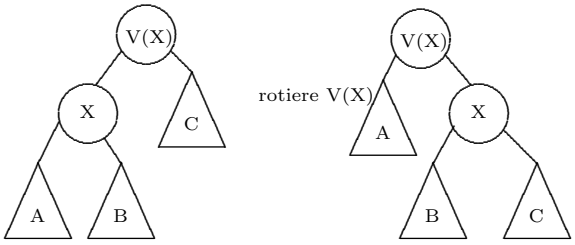


Abbildung 21: zig

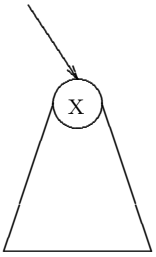


Abbildung 22: Teilbaum T_x

$\leq \sum_{i=1}^{n} (\log(W) - \log(w(i))) = \sum_{i=1}^{n} \log \frac{W}{w(i)}$
und symmetrisch $\square$

Lemma 2.2.3 ("ACCESS-Lemma") *$A(x,T)$ = Kosten einer Elementaroperation[10] von ACCESS(x) $+\phi(T') - \phi(T)$*
$\leq 3\,(r'(x) - r(x))$ im Falle zig-zig oder zig-zag
$\leq 3\,(r'(x) - r(x)) + 1$ im Falle zig
Dabei r' : Rang Funktion des Baum T', der aus T bei dem Zugriff entsteht.

Beweis 2.2.3

zig: z.z.: $1 + \phi(T') - \phi(T) \leq (r'(x) - r(x)) + 1$. Nur x und y haben ihre r-Werte geändert.
$1 + \phi(T') - \phi(T) = 1 + r'(x) + r'(y) - r(x) + r(y)$
$= \underbrace{r'(y) - r(y)}_{\leq 0} + \underbrace{r'(x) - r(x)}_{\geq 0} + 1$
$\leq 3\,(r'(x) - r(x)) + 1$

[10] zig-zig, zig-zag, zig

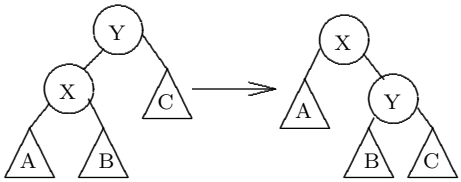


Abbildung 23: Verhalten von zig

14.05.03
$S(x) = \sum_{i \in T_x} w(i)$ "Größe von x"
$r(x) = \log_2 S(x)$ "Rang von x"
$\phi(T) = \sum_{x \in T} r(x)$
Wie üblich: Splay(x, T)=echte Kosten von ACCESS(X) in T (einschl. kostenloser Restrukturierung mit MTR *(x).
$A(x,T) = \text{Splay}(x,T) + \phi(\tilde{T}) - \phi(T)$, wobei $\tilde{T}$ der entstehende Splay Tree ist.

Lemma 2.2.3 ("ACCESS Lemma") *$A(x,T) \leq 3\,(r(t) - r(x)) + 1$, wobei t = Wurzel von T.*

Beweis: 2.2.3 Fallunterscheidung:

(i) es finden (bei ACCESS(x)) keine Umbauten statt $\Rightarrow x = t =$ Wurzel von T
$\Rightarrow A(x,T) \leq 3\underbrace{(r(t) - r(x))}_{=0} + 1$
$A(x,T) = \underbrace{\text{Splay}(x,T)}_{=1,\text{da Wurzel}} + \underbrace{\phi(\tilde{T}) - \phi(T)}_{=0,\text{da } \tilde{T}=T}$

(ii) Es finden Umbauten statt.

zu zeigen: $\sigma \in \{\text{zig, zig-zig, zig-zag}\}$
Sei T der Baum vor der Ausführung von σ.
Sei T' der Baum nach der Ausführung von σ.
Seien r, r' die Rangfunktionen von T und T'.
Dann gilt:
$\sigma = \text{zig} : 1 + \phi(T') - \phi(T) \leq 3(r'(x) - r(x)) + 1$
$\sigma \in \{\text{zig-zig, zig-zag}\} : 2 + \phi(T') - \phi(T) \leq 3(r'(x) - r(x)) + 0$
Daraus folgt Lemma 2.2.3, denn:

- zig kommt höchstens 1x vor (bei ACCESS(X), wenn der Suchpfad ungerade Länge hat)
- Im letzten Baum $\tilde{T}$ gilt: $\underbrace{r'(x)}_{11} = \underbrace{r(t)}_{12}$, denn beide Elemente x,t sind Wurzeln von Bäumen mit identischen Einträgen.

Fall 1 $\sigma =$ zig. (Siehe letzte Vorlesung)

Fall 2 $\sigma =$ zig-zig. Klar: Nur x,y,z können ihre Ränge ändern.

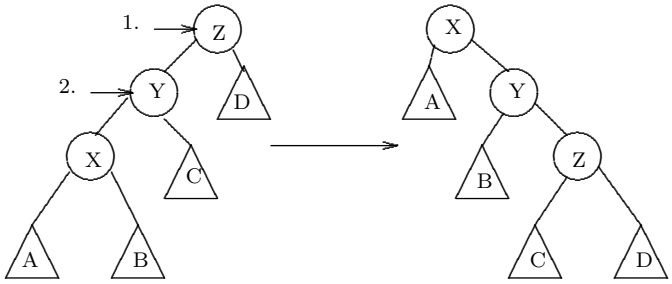


Abbildung 24: Fall zig-zig

Es gilt: $2 + \phi(T') - \phi(T) = 2 + \underbrace{r'(x) - r(z)}_{=0^{13}} + \underbrace{r'(y)}_{\leq r'(x)} + r'(z) - r(x) \underbrace{-r(y)}_{\leq -r(x)}$
$\leq 2 + r'(x) + r'(z) - 2r(x) \leq 3(r'(x) - r(x))$ d.h.:
z.z.: $r(x) + r'(z) - 2r'(x) \overset{!}{\leq} -2 \Rightarrow \log S(x) + \log S(z) - \log S'(x)$
$= \log \underbrace{\frac{S(x)}{S'(x)}}_{\in(0,1)} + \log \underbrace{\frac{S'(z)}{S'(x)}}_{\in(0,1)}$
Es gilt: $S(x) < S'(x), S'(z) < S'(x)$
$S(x) + S'(z) < S'(x)$
$\frac{S(X)}{S'(x)} + \frac{S'(z)}{S'(x)} < 1$

Zum Glück gilt: Die Funktion $f(v, w) = \log v + \log w$ hat für $0 < v, w < 1, v + w < 1$ ihr Maximum -2 [an der Stelle $(v, w) = \left(\frac{1}{2}, \frac{1}{2}\right)$]

[11] Rangfunktion von $\tilde{T}$
[12] Rangfunktion im Ausgangsbaum T
[13] Da beide den kompletten Teilbaum enthalten

Denn: $\log(v) + \log(w) \leq -2 \Rightarrow \log(vw) \leq -2 \Leftrightarrow \underbrace{vw}_{14} \leq \frac{1}{4}$ □

Fall 3 σ = zig-zag. Klar: Nur x, y, z können Ränge ändern.

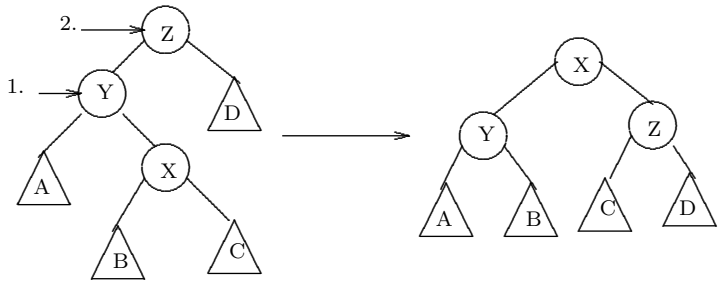


Abbildung 25: Fall zig-zag

$$
\begin{aligned}
&2 + \phi(T') - \phi(T) \\
&= 2 + \underbrace{r'(x) - r(z)}_{=0} + r'(y) + r'(z) - r(x) \underbrace{- r(y)}_{\leq -r(x)} \\
&\leq 2 + r'(y) + r(z) - 2r(x) \overset{!}{\leq} \underbrace{2}_{15} \underbrace{(r'(x) - r(x))}_{\geq 0}
\end{aligned}
$$

z.z.: $2 \overset{!}{\leq} 2 \cdot r'(x) - r'(y) - r'(z)$

Nun gilt: $S'(x) \geq S'(y) + S'(z)$, also $\frac{S'(x)}{2} \geq \frac{S'(y)+S'(z)}{2}$

[16] $\geq \sqrt{S'(y) \cdot S'(z)}$

$\Rightarrow \frac{S'(x)^2}{4} \geq S'(y) \cdot S'(z)$

$\Rightarrow \frac{S'(x)^2}{S'(y) \cdot S'(z)} \geq 4$

$$\Rightarrow \underbrace{\log \frac{S'(x)}{S'(y)} + \log \frac{S'(x)}{S'(z)}}_{2r'(x) - r'(y) - r'(z)} = \log \frac{S'(x)^2}{S'(y) \cdot S'(z)} \geq \log 4 = 2 \quad \square$$

Theorem 2.2.4 ("Balance Theorem") *Sei T ein Splay-Tree mit n Elementen. Dann verursacht eine Folge von m ACCESS-Operationen Kosten in $O((m+n) \log n + m)$*

[14] wird maximal für $v + w = 1$

[15] Mehr als nötig

[16] arithmetisches/geometrisches Mittel: $\sqrt{ab} \geq \frac{a+b}{2}$

Beweis: 2.2.4 Wähle alle Gewichte $w(i) = \frac{1}{n} \Rightarrow W = \sum_{i=1}^{n} w(i) = 1$

amortisierte Kosten pro ACCESS Operation im aktuellen Baum T:

$$A(x,T) \leq 3(\underbrace{r(t)}_{\log(W)=0} - \underbrace{r(x)}_{\geq \log(\underbrace{w(x)}_{=\frac{1}{n}})}$$

$$\leq 3 \cdot \log n + 1$$

$$\text{Echte Gesamtkosten} = \underbrace{\sum_{x} A(x,T)}_{\leq 3m \log n + m} + \underbrace{\phi(T) - \phi(T')}_{\leq \sum_{i=1}^{n} \log \underbrace{\frac{W}{w(i)}}_{=n} = n \log n} \quad \square$$

Was besagt Theorem 2.2.4? Für lange Zugriffsfolgen, d.h. große m sind Splay-Trees so gut wie balancierte Bäume ($O(m \log n)$). Was soll's?

Theorem 2.2.5 (Statisches Optimalitätsproblem) *In einer Folge von m Zugriffen auf Elemente der Menge $\{1, 2, \ldots, n\}$ sei $q(i) \overset{?}{\geq} 1$ die Anzahl der Zugriffe auf das Element i. Dann sind die Gesamtkosten aller Zugriffe in* $O\left(m + \sum_{i=1}^{n} \left(q(i) \cdot \log\left(\frac{m}{q(i)}\right)\right)\right)$

Beweis 2.2.5 Diesmal setzen wir $w(i) = \frac{q(i)}{m} \Rightarrow W = \sum_{i=1}^{n} w(i) = 1.$

Amortisierte Kosten pro Zugriff auf Element i:

$$\leq 3(\underbrace{r(t)}_{=0} - \underbrace{r(x_i)}_{\geq \log\left(\frac{q(i)}{m}\right)}) + 1 \leq 3 \log\left(\frac{m}{q(i)}\right) + 1$$

$\Rightarrow$ echte Gesamtkosten

$$\leq \sum_{i=1}^{n} \left[q(i)\left(3 \log\left(\frac{m}{q(i)}\right) + 1\right)\right] + \underbrace{\phi(T) - \phi(T')}_{\leq \sum_{i=1}^{n} \log \underbrace{\frac{W}{w(i)}}_{=\frac{m}{q(i)}} = n \cdot \log \frac{m}{q(i)}}$$

$$\leq c \cdot \sum_{i=1}^{n} q(i) \log \tfrac{m}{q(i)} + \underbrace{\sum_{i=1}^{n} q(i)}_{=m} \quad \square$$

19.05.03

Rekapitulation: *Statisches* Optimalitätstheorem:

In welcher Hinsicht ist das optimal? Wären die $q(i)$ im voraus bekannt, so könnte man einen optimalen *statischen* Suchbaum T bauen, bei dem die Elemente mit großen Werten $q(i)$ möglichst weit oben sitzen (in Zeit $O(n^2)$ mit dynamischer Programmierung), der zur Laufzeit nicht mehr verändert wird.
Man kann zeigen: In T hat i eine Tiefe in $\Omega\left(\log \frac{m}{q(i)}\right)$. ($\rightarrow$ K. Mehlhorn Datenstrukturen und Algorithmen Band 1)

Frage: Sind Splay-Trees auch dynamisch optimal, d. h. im Vergleich mit optimalen Bäumen, die zur Laufzeit umstrukturiert werden dürfen? Offen!

3 Paging (Caching)

3.1 Modell und Algorithmen

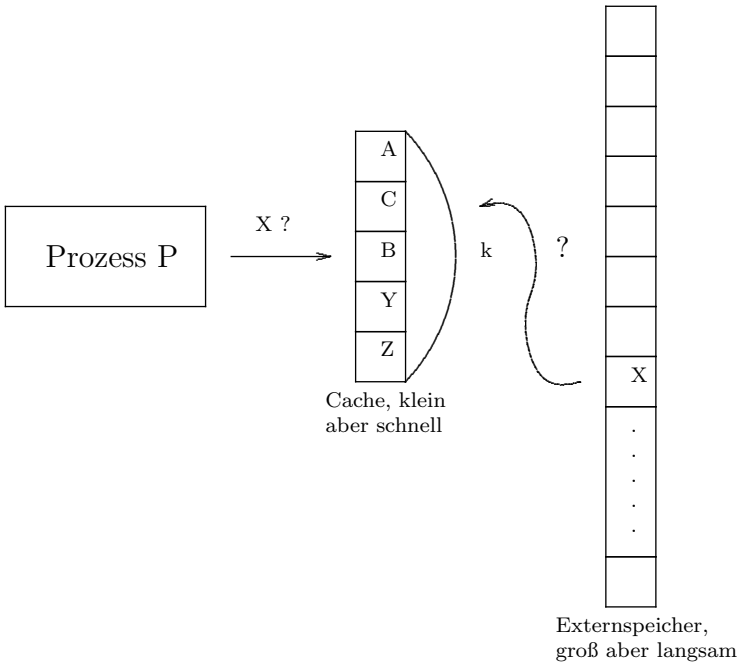


Abbildung 26: Modell des Caching

- Prozess P benötigt Seite x;
- schaut im Cache nach;

- falls x vorhanden: OK, Kosten = 0
- falls x nicht vorhanden (Seitenfehler)
  - hole x aus Externspeicher; Kosten = 1
  - bringe x in Cache;
  - falls Cache vorher voll: Verdränge eine $\underbrace{\text{der vorhandenen}}_{\text{welche?}}$ Seiten aus dem Cache um für x Platz zu schaffen.

Verschiedene Strategien möglich: Wenn nötig, verdränge aus dem Cache diejenige Seite:

- Deren letzte Benutzung am längsten zurückliegt. (Least Recently Used, LRU)
- Die sich schon am längsten im Cache befindet. (First In First Out, FIFO)
- Die als letzte in den Cache gebracht wurde. (Last In First Out, LIFO)
- Die am seltensten benutzt wurde.
- Clock[17] $\left.\begin{array}{l}\text{beim Einladen}\\ \text{nach Zugriff}\end{array}\right\}$ Zugriffsbit=1

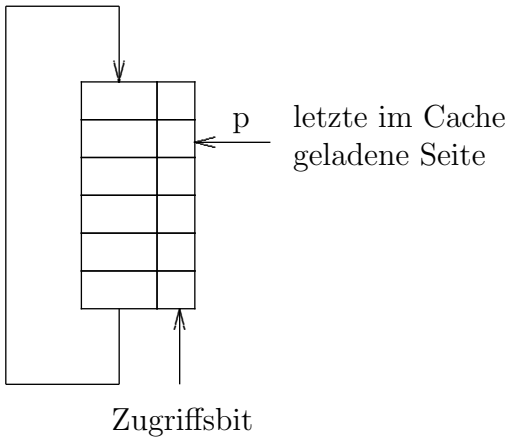


Abbildung 27: Algorithmus Clock

Bei Seitenfehler: p rückt vor bis zur ersten Seite deren Bit 0 ist und

[17] real, gute Approximation an LRU, weniger Overhead als LRU

entfernt sie. Dabei wird für jede angetroffene Seite mit Bit=1 deren Bit auf 0 gesetzt.

- Flush when full: Bei Seitenfehler: Entleere den gesamten Cache

Frage: Was wäre optimale Auslagerungsstrategie, wenn man alle zukünftigen Seitenanfragen kennen würde?

Ideal: Verdränge eine Seite, die nie wieder benötigt wird. Existenz?

realistisch: Verdränge die Seite, die erst am spätesten in der Zukunft benutzt wird. Longest Forward Distance (LFD).

Folklore Resultat: LFD ist optimal.

Theorem 3.1.1 *LFD ist optimaler Offline-Algorithmus.*

Beweis 3.1.1: Lebt vom Lemma 3.1.2

Lemma 3.1.2 *Sei* ALG *irgendein Offline-Paging Algorithmus. Sei* $\sigma = \sigma_1\sigma_2\ldots\sigma_n$ *eine Anforderungsfolge. Dann gibt es für jedes* $i \leq n$ *einen Algorithmus* ALG $_i$ *mit folgenden Eigenschaften:*

(1) ALG $_i\,[\sigma_1\ldots\sigma_{i-1}]$ [18] $=$ ALG $[\sigma_1\ldots\sigma_{i-1}]$

(2) Falls σ_i *bei* ALG *und* ALG $_i$ *(gleicher Cache) Seitenfehler verursacht:* ALG $_i$ *verdrängt die Seite, die in* $\sigma_{i+1}\ldots\sigma_m$ *am spätesten wieder an die Reihe kommt. (wie LFD)*

(3) ALG $_i(\sigma)$[19] $\leq$ ALG (σ)

Beweis 3.1.2: Müssen ALG $_i$ definieren. Die ersten $i-1$ bearbeitet ALG $_i$ wie ALG $\Leftrightarrow$ (1)
Falls bei σ_i kein Seitenfehler auftritt: ALG $_i$=ALG $\surd$
Falls bei σ_i ein Seitenfehler auftritt: ALG $_i[\sigma_i]$ = die Seite v, die am spätesten in $\sigma_{i+1},\ldots,\sigma_n$ vorkommt.
ALG $[\sigma_i]$ = eine Seite u.
Falls $u = v$ ALG $_i$=ALG, fertig.
Falls $u \neq v$: Jetzt folgende Cacheinhalte:

[18] Notation: ALG $[\sigma]$: Handlungsweise von ALG
[19] Notation: ALG (σ): Kosten von ALG

$\begin{matrix}\text{ALG}: & X+v \\ \text{ALG}_i & X+u\end{matrix}$ (denn vor σ_i waren beide Caches identisch. ALG hat noch v, ALG_i hat noch u)
definiere $\text{ALG}_i[\sigma_{i+1}], \ldots, \text{ALG}_i[\sigma_n]$ wie folgt: Sei $i+1 \leq j \leq n$

- falls $\sigma_j \notin \{u, v\}$ und $\text{ALG}[\sigma_j] \neq v$ (* Entweder machen beide oder keiner einen Seitenfehler *)
  $\text{ALG}_i[\sigma_j]$ =ALG$[\sigma_j]$ (* hinterher Caches: $\begin{matrix}\text{ALG}: & X'+v \\ \text{ALG}_i & X'+u\end{matrix}$ *)

- falls $\sigma_j \notin \{u, v\}$ und $\text{ALG}[\sigma_j]$=v (* Beide machen Seitenfehler *)
  $\text{ALG}_i[\sigma_j] = u$ (* Danach beide Caches gleich *)
  $\forall k > j : \text{ALG}_i[\sigma_k] = \text{ALG}[\sigma_k]$

- falls $\sigma_j = u$ und $\text{ALG}[\sigma_j] = v$ (* ALG hat Seitenfehler, ALG_i nicht *)
  $\text{ALG}_i[\sigma_j] = \text{NOP}$[20] (* Caches jetzt identisch *)
  ab jetzt ALG_i =ALG

- falls $\sigma_j = u$ und $\text{ALG}[\sigma_j] = u' \neq u, v$ (* ALG hat Seitenfehler, ALG_i nicht *)
  $\text{ALG}_i[\sigma_j]$ =NOP. (* Caches jetzt: $\begin{matrix}\text{ALG}: & X'-u'+u+v = & X''+v \\ \text{ALG}_i: & & \underbrace{X''}_{21}+u'\end{matrix}$ *)

- falls $\sigma_j = v$ (* Dann muss vorher u bzw u' verlangt worden sein! *)
  (* ALG_i macht Seitenfehler, ALG nicht *)
  $\text{ALG}_i[\sigma_j] = u'$
  ab jetzt: $\text{ALG}_i = \text{ALG}$.
  einziger Fall, wo ALG_i Seitenfehler macht, ALG aber nicht.
  Aber: Def von $v \Rightarrow$ vorher war anderer Fall ($\sigma_j = u$) wo ALG Fehler macht, ALG_i nicht.

□

Damit Beweis von Theorem 3.1.1: Sei OPT ein optimaler Offline-Algorithmus für σ.
Wende $n-$mal das Lemma 3.1.2 an, $i = 1, \ldots, n$.
OPT_1 verhält sich bei σ_1 wie LFD, $\text{OPT}_1(\sigma) \leq \text{OPT}(\sigma)$.
Wende Lemma 3.1.2 auf OPT_1 an:

[20] Notation: Kein Seitenfehler $\Rightarrow$ ALG $[\sigma_j]$=NOP
[21] $X'' = X' - u' + u$

OPT_2 verhält sich bei σ_1 und σ_2 wie LFD, $\text{OPT}_2(\sigma) \leq \text{OPT}_1(\sigma)$.
$\vdots$
OPT_n verhält sich $\left.\begin{array}{l}\text{bei } \sigma_1 \dots \sigma_{n-1} \text{ wie OPT}_{n-1} \\ \text{bei } \sigma_n \text{wie LFD}\end{array}\right\}$
$\overset{\text{i.V.}}{\Rightarrow}$[22] verhält sich bei $\sigma_1 \dots \sigma_n$ wie LFD mit $\text{OPT}_n(\sigma) \leq \text{OPT}_{n-1}(\sigma) \leq \dots \leq \text{OPT}(\sigma)$. □

Bemerkung: Man kann Modell verfeinern: (h, k)-Paging
Online-Algorithmus hat Cache der Größe k
Offline-Algorithmus hat Cache der Größe $h \leq k$

3.2 Markierungsalgorithmen

Sei $\sigma = \sigma_1\sigma_2 \dots \sigma_n$ eine Anforderungsfolge.
Zerlege σ in Phasen bezüglich k:

Phase: Maximaler Abschnitt von σ, in dem k verschiedene Seiten angefordert werden[23].

Beispiel: $k = 4$
a b b c a b c d a b|e a b c|d e a c a d e|b c a d c b|f e g h

Definition: Ein Paging-Algorithmus heißt Markierungsalgorithmus, falls gilt:

(i) Zu Beginn einer Phase sind alle Seiten unmarkiert.

(ii) Während einer Phase wird jede angeforderte (auch eingeladene) Seite markiert.

(iii) Bei Seitenfehler wird eine unmarkierte Seite verdrängt.

Klar: Flush When Full ist ein Markierungsalgorithmus.

21.05.03

Theorem 3.2.1 *Sei* ALG *ein Markierungsalgorithmus, dann ist* ALG $\frac{k}{k-h+1}-$*kompetitiv. (dabei* $k =$ *Cache-Größe von* ALG *,* $k \leq h$ *Cache-Größe von* OPT *. Spezialfall* $h = k$*:* ALG *ist* k*-kompetitiv.)*

[22] Induktionsvorraussetzung
[23] eventuell mehrmals

Beweis 3.2.1: Sei ALG ein Markierungsalgorithmus. Pro Phase werden genau k verschiedene Seiten angefordert, angeforderte Seiten bleiben im Cache von ALG für den Rest der Phase.
⇒ pro Phase macht ALG höchstens k Fehler.
Jetzt untere Schranke für die Fehler von OPT : Am Anfang der verschobenen

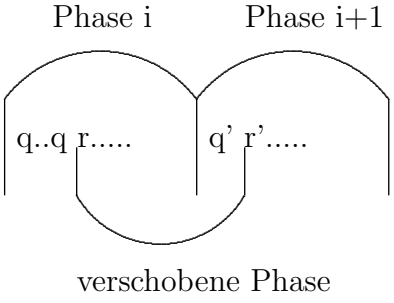


Abbildung 28: Phasenverschiebung bei OPT

Phase hat OPT die Seite q im Cache. Im laufe der verschobenen Phase werden mindestens k-verschiedene Seiten angefordert.
⇒ OPT macht in dem Intervall $k-(h-1) = k-h+1$ Fehler
Aufaddiert über die Phasen ⇒ $\text{ALG}(\sigma) \leq \frac{k}{k-h+1} \cdot \text{OPT}(\sigma) + \underbrace{k}_{24}$ □

Schon gesehen: Flush when Full ist Markierungsalgorithmus.
Clock ist *kein* Markierungsalgorithmus. (Übungsaufgabe)

Lemma 3.2.2 *Last recently used (LRU) ist Markierungsalgorithmus.*

Beweis 3.2.2: Sei σ eine Anforderungsfolge. Zerlege σ in Phasen. Wäre LRU kein Markierungsalgorithmus, so würde in einer Phase p eine markierte Seite x verdrängt. Sei x die erste Seite mit diesem Schicksal. Betrachte 1. Anforderung von x in Phase p. (Muss es geben, sonst wäre x nicht markiert)
Nach dieser Anforderung ist x die am spätestens gebrauchte Seite im Cache. Bevor x von LRU verdrängt wird, müssen alle $k-1$ anderen Seiten im Cache mindestens einmal angefordert worden sein.

$k-1$	verschiedene Seiten
1	Anforderung von x (die erste)
1	Anforderung die x verdrängt
$k+1$	Zugriffe auf verschiedene Seiten in Phase p *Widerspruch*

[24] Kosten von ALG in letzter Phase

Frage: Gibt es auch Algorithmen, die keine Markierungsalgorithmen sind? Ja, z.b. FIFO.

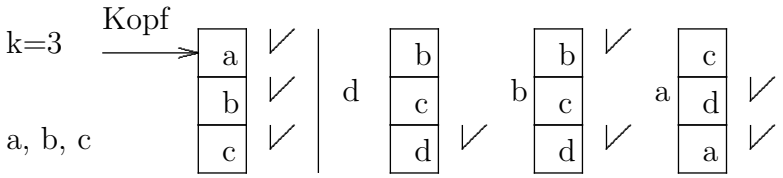


Abbildung 29: FIFO ist kein Markierungsalgorithmus

Beispiel: Die markierte Seite b wird verdrängt.
Aber: FIFO ist auch $\frac{k}{k-h+1}-$kompetitiv wegen folgender Variante der Markierungsalgorithmen:

Definition: Ein Paging-Algorithmus heißt konservativ, falls gilt: In jeder Folge, in der höchstens k verschiedene Seiten angefordert werden, entstehen höchstens k Fehler. Damit:

Theorem 3.2.2 *Jeder konservative Paging-Algorithmus ist* $\frac{k}{k-h+1}-$*kompetitiv.*

Beweis wie bei Theorem 3.2.1

Lemma 3.2.3 *Konservative und Markierungsalgorithmen verhalten sich wie in Abbildung 30.*

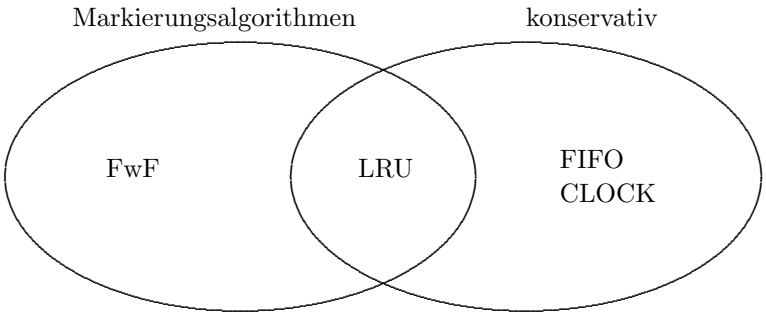


Abbildung 30: Mengen Diagramm zu Paging Algorithmen

Beweis 23:

(i) FIFO ist konservativ. Sei σ Anforderungsfolge mit $\leq k$ verschiedenen Seiten.
zu zeigen: $\text{FIFO}(\sigma) \leq k$
Folge τ entsteht durch Streichung aller Anforderungen, die keinen Seitenfehler verursachen. (Hat keinen Einfluss auf den FIFO-Cache)

$$\begin{array}{lccccccc} \sigma = \sigma_1 \dots \sigma_{i-1} & \sigma_i & \dots & \sigma_j & \dots & \sigma_k & \dots & \sigma_n \\ & \tau_1 & & \tau_2 & & \tau_r & & \end{array}$$

- jedes τ_i macht einen Seitenfehler.
- $\text{FIFO}(\sigma) = \text{FIFO}(\tau) = r$.

Aus Abbildung 31 $\Rightarrow$ alle τ_i sind voneinander verschieden $\Rightarrow r \overset{25}{\leq} k$

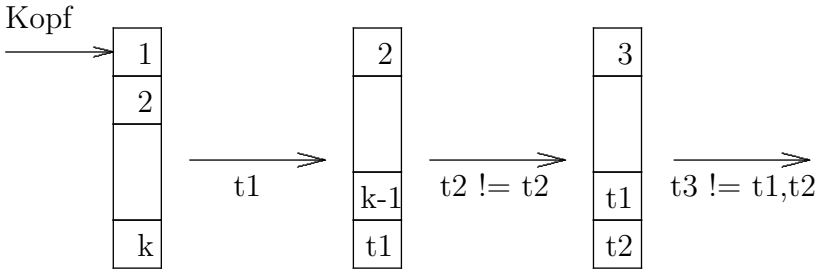


Abbildung 31: Verlauf von $\text{FIFO}(\tau)$

(ii) Flush when Full (FwF) ist nicht konservativ.
Beispiel: $k = 4$ (Siehe Abbildung 32) 3 verschiedene Seiten, 5 Fehler.

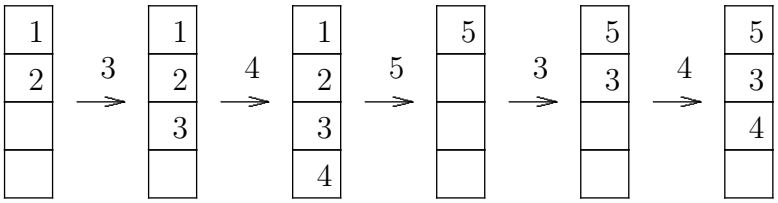


Abbildung 32: Flush when Full ist nicht konservativ

[25] Voraussetzung über σ

(iii) LRU ist konservativ.
Angenommen, die Folge σ fordert $\leq k$ verschiedene Seiten an und LRU macht dabei $\geq k+1$ Fehler.
$\Rightarrow$ Es gibt ein x, das in σ zweimal vorkommt und beide Male Seitenfehler ergibt. Nach dem ersten Zugriff auf x ist x die "frischeste" Seite im Cache. Vor dem zweiten (Seitenfehler-) Zugriff ist x die älteste Seite.
$\Rightarrow$ es hat zwischendurch Zugriffe auf $k-1$ andere Seiten gegeben

	$k-1$	verschiedene Zugriffe
	1	erster Zugriff auf x
$\Rightarrow$	1	Zugriff, der x verdrängt
	$k+1$	*Widerspruch!*

Theorem 3.2.4 *Sei* ALG *ein* (k,k)*-Paging-Algorithmus mit kompetitiven Faktor c. Dann ist* $c \geq k$

Beweis 3.2.4: Betrachte eine Situation, in der es genau $k+1$ Datenseiten gibt. (d.h. alle bis auf eine Seite sind im Cache) Sei ALG ein Paging-Algorithmus. Konstruiere eine böse Folge σ für ALG , die immer diejenige Seite anfordert, die gerade nicht im Cache ist $\Rightarrow \sigma$ Länge $n : \text{ALG}(\sigma) = n$.
Wieviele Fehler macht LFD, der optimale Offline-Algorithmus?
Angenommen, LFD[i] gibt Fehler (Siehe Abbildung 33): j wird von LFD so

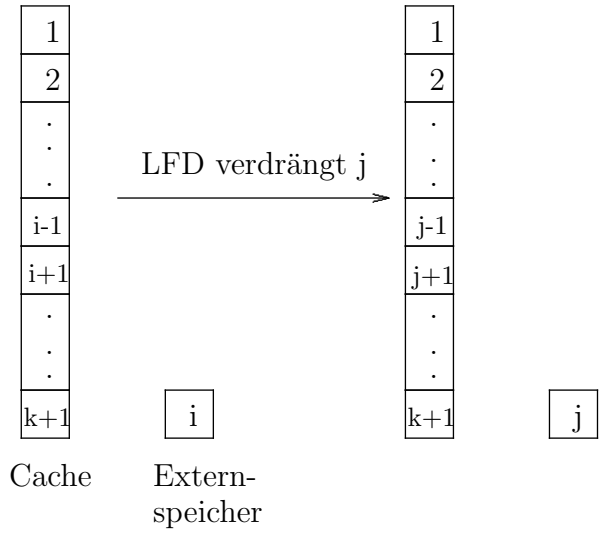


Abbildung 33: Cache Situation von LFD

gewählt, das alle übrigen $k-1$ Seiten im Cache vorher angefordert werden.

(jede mindestens 1x) $\Rightarrow$ LFD macht bei den nächsten $k-1$ Anforderungen in σ keinen Fehler. $\square$

2 Fragen:

1. Werden die Algorithmen mit steigender Cache-Größe schlechter?
2. Sind alle Algorithmen gleich gut (bzw schlecht)?

Zu 1:

Lemma 3.2.6 (Belady's Anomalie) *Es gibt Anforderungen, für die FIFO mit kleinerem Cache weniger Fehler macht.*

Beweis 3.2.6: $\sigma = 0, 1, 2, 3, 0, 1, 4, 0, 1, 2, 3, 4$ auf leerem Cache. Siehe Abbildung 34 und 35 auf der nächsten Seite.

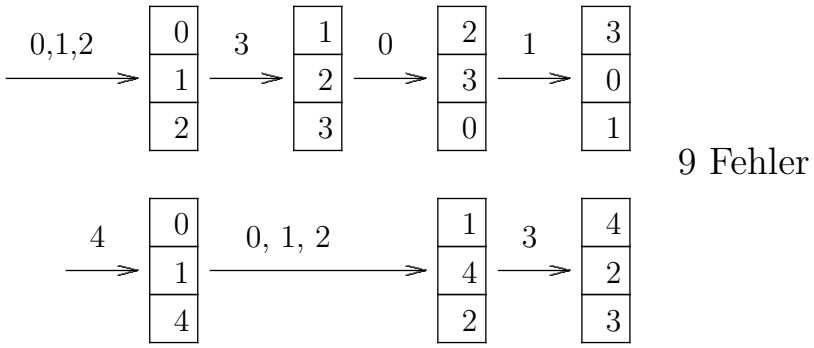


Abbildung 34: FIFO mit $k = 3$

- Dies ist ein Einzelfall.
- Kann z.b. bei LRU nicht vorkommen!
- zu 1: Theorem 3.2.4 und Theorem 3.2.1 implizieren nur, dass es für wachsende Cache-Größe k für den Online Algorithmus schwieriger wird, mit OPT mitzuhalten.

Least Frequently used (LFU) ist nicht kompetitiv. (verdrängt stets die Seite aus dem Cache, auf die seit Systemstart insgesamt am seltesten zugegriffen wurde)
Es gebe $k+1$ Datenseiten. Betrachte für eine Variable l die Folge: $\sigma = 1^l 2^l \dots (k-1)^l (k\ k+1)^{l-1}$ (Siehe Abbildung 36 auf Seite 39)

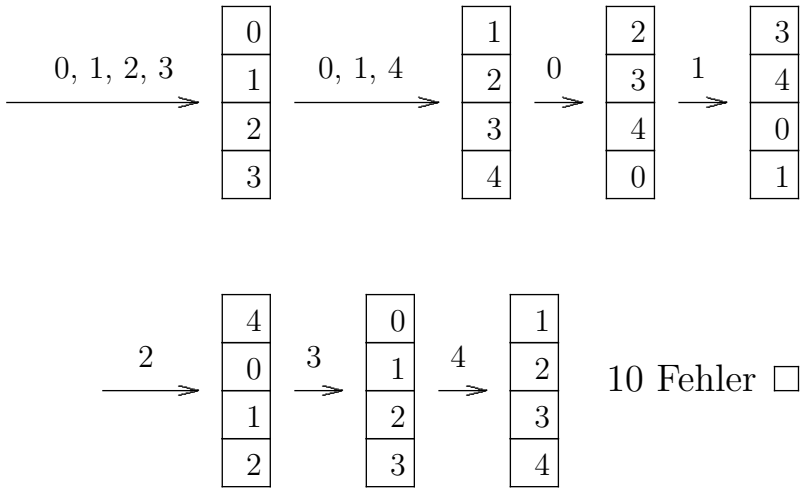


Abbildung 35: FIFO mit $k = 4$

LFU macht $k - 1 + 2(l - 1) = k + 2l - 3$ Fehler
OPT macht $k - 1 + 2 = k + 1$ Fehler
$\Big\}\Rightarrow$
$\Rightarrow$ Behauptung, denn l ist beliebig groß.

26.05.03

Frage: Sind z.b. LRU und FIFO gleich gut? In der Praxis: LRU besser als FIFO. Problem der Online Analyse: Beweise, dass LRU besser ist als FIFO $\rightarrow$ feineres Modell?

Ansatz: (ACCESS Graphs) Modelliere Lokalität der Seitenanforderungen: Definiere Zugriffsgraph $G = (V, E)$ mit $V = \{$ Alle Seiten im Externspeicher$\}$. Erlaubte Zugriffsfolgen = Pfade in G. (falls G = vollständiger Graph: Altes Modell) Beispiel siehe Abbildung 37 auf Seite 40. Graph G $\rightarrow$ kompetitiver Faktor C_g für Algorithmus ALG .

Man kann zeigen:

1. Auf jedem Graphen ist FIFO höchstens $\frac{k+1}{2}$ kompetitiv.

2. LRU ist auf keinem Graphen schlechter als FIFO.

3. Für manche Graphklassen (z.b. Bäume) ist LRU echt besser als FIFO.

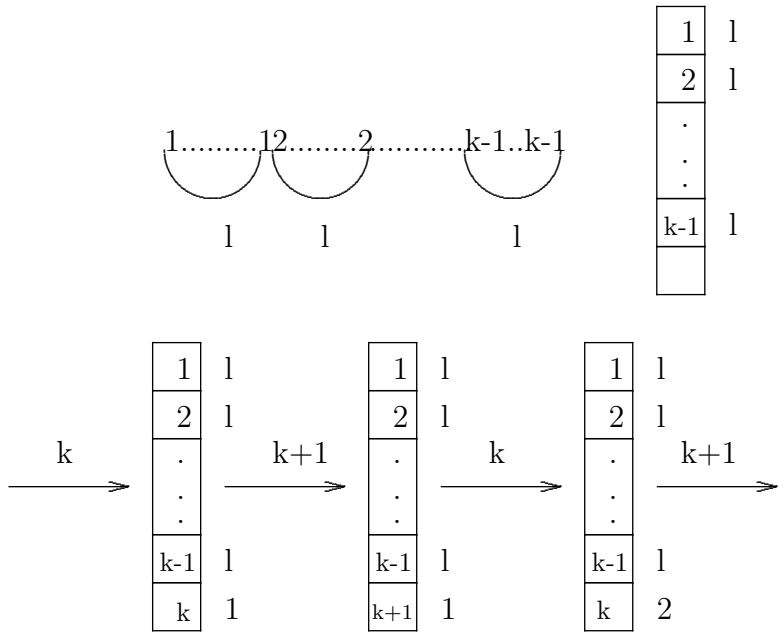


Abbildung 36: Verhalten von LFU

G Baum $\Rightarrow C_G^{\text{LRU}} = \max_{T(k+1)}{}^{26}(\#\text{Blätter von } T-1) < \frac{k+1}{2}$ für Bäume mit hohem Verzweigungsgrad.

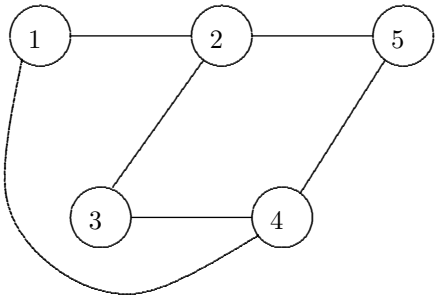
3.3 Randomisierte Paging Algorithmen

Frage: Hilft Randomisierung? (Erinnerung: Listen $2 \searrow \frac{7}{4}$) Sinnvoller randomisierter Paging-Algorithmus:

Definition: MARK ist randomisierter Markierungsalgorithmus: Teilt Anforderung in Phasen auf = Anforderung von k verschiedenen Seiten.

- Zu Beginn einer neuer Phase: alle Seiten im Cache unmarkiert.
- Während der Phase: Jede angeforderte (auch frisch eingeladene) Seite wird markiert.
- Bei einem Seitenfehler wird eine der nicht markierten Seiten zufällig ausgewählt und verdrängt.

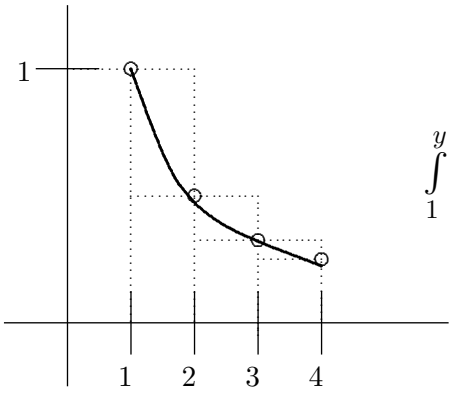
[26] Teilbaum T mit $k+1$ Knoten



Nach Zugriff auf Seite 5 dürfen nur Seite 2 oder 4 (oder 5), nicht aber 1 oder 3 verlangt werden.

Abbildung 37: ACCESS Graph Beispiel

Theorem 3.3.1 *(Fiet et al. '88) MARK ist $2H_k$-kompetitiv. (gegen vergeßliche Gegenspieler) Dabei: $H_k = \sum_{i=1}^{k} \frac{1}{i}$ = k-te harmonische Zahl. Es gilt $\ln k \leq H_k \leq 1 + \ln k$. (Siehe Abbildung 38)*



$$\int_1^y \frac{1}{x} \mathrm{dx} = \ln y$$

Abbildung 38: Graph von $\frac{1}{x}$

Beweis 3.3.1: Sei σ eine Anforderungsfolge. Betrachte die i-te Phase.

Redeweise: Zu Beginn von Phase i: k alte Seiten im Cache. Davon verschieden, während Phase i angeforderte *neue* Seiten: m_i Stück.
in Phase $i-1$ und i werden $k+m_i$ Seiten angefordert $\Rightarrow$ OPT macht in Phase i und $i-1$ insgesamt m_i Seitenfehler. (OPT hat Cache Größe k) Klar: OPT macht in Phase 1 mindestens m_1 Fehler.
$\Rightarrow \text{OPT}(\sigma) \geq \frac{1}{2} \sum_{i=1}^{t} m_i$ wobei t = Anzahl der Phasen von σ. (*).
Betrachte jetzt MARK. *Beachte:* Auch Anforderung einer alten Seite in Phase i kann Seitenfehler verursachen. (Wenn nämlich diese Seite in Phase i bereits

verdrängt wurde)⇒ Seitenfehler von MARK. In Phase i m_i neue Seiten. Annahme: in Phase i erst alle m_i Neuanforderungen, dann $k - m_i$ Zugriffe auf die alten Seiten. (Worst Case) Betrachte jetzt den ersten Zugriff[27] auf die j-te Seite p_j in Phase i : $1 \leq j \leq k - m_i$
W'keit(p_j beim ersten Zugriff noch im Cache) $= \frac{\text{Anzahl der günstigen Fälle}}{\text{Anzahl der möglichen}}$
$= \frac{\text{Anzahl der unmarkierten alten Seiten im Cache}}{\text{Anzahl aller (auch der verdrängten) unmarkierten alten Seiten}}$
$= \frac{k-(j-1)-m_i}{k-(j-1)}$
W'keit(Seitenfehler beim ersten Zugriff auf j-te alte Seite)
$= 1 - \frac{k-(j-1)-m_i}{k-(j-1)} = \frac{m_i}{k-(j-1)}$
⇒ zu erwartende Anzahl von Seitenfehler bei Zugriff auf die 1 bis $k - m_i$-te alte Seite:
$\leq \sum_{j=1}^{k-m_i} \frac{m_i}{k-(j-1)} = m_i \left(H_k - H_{m_i}\right)$
⇒ zu erwartende Kosten von MARK in Phase $i \leq \underbrace{m_i}_{28} + m_i \left(H_k - H_{m_i}\right) =$

$m_i(H_k + 1 - H_{m_i}) \leq \underbrace{m_i}_{29} H_k$ □

Frage: Geht es noch besser? Man kann statt $2H_k$ sogar H_k erreichen (Algorithmus PARTITION). Aber nicht mehr.

Theorem 3.3.2 *(Fiat et al) Kein randomisierter Algorithmus kann im (k, k) Modell einen kompetitiven Faktor $< H_k$ haben.*

Beweis 3.3.2: Sei ALG irgendein randomisierter Paging-Algorithmus. Der vergeßliche Gegener:

- Benutzt $k + 1$ Seiten.
- Kennt ALG und die Wahrscheinlichkeitsverteilung, die ALG für seine Entscheidungen benutzt, aber nicht die zur Laufzeit getroffenen konkreten Entscheidungen von ALG .

⇒ Gegner kann nicht wissen, welche Seite gerade nicht im Cache von ALG ist. (Und ihn damit ärgern, dass er diese anfordert) Aber er kann im voraus für jede Seite i und jeden Zeitpunkt in der Anforderungsfolge ausrechnen:
$p(i)$ =W'keit(Seite i *nicht* im Cache von ALG)
Gegner erzeugt beliebig lange Folgen von Phasen, kann selbst jede Phase mit

[27] Nur dies kann Seitenfehler verursachen, da MARK Markierungsalgorithmus
[28] m_i neue Seiten, machen auf jeden Fall Seitenfehler
[29] $2\cdot$ Mindestkosten von OPT (*)

≤ 1 Fehler bedienen. Möchte ALG zu erwartenden Kosten H_k zwingen. Dazu teilt Gegner jede Phase in k Teilphasen auf. Pro Teilphase j:

- Anforderung einer Folge von bereits markierten Seiten.
- Anforderung und Markierung *einer* unmarkierten Seite.

$\Rightarrow$ zu erwartende Kosten für ALG $\overset{!}{\geq} \frac{1}{k+1-j}$ (Wenn das geht. folgt Behauptung wegen: $\sum_{j=1}^{k} \frac{1}{k+1-j} = H_k$)

Teilphase j: $k+1-j$ Seiten unmarkiert, $j-1$ Seiten markiert (Zu Beginn der j-ten Teilphase).
Sei $\gamma = \sum_{\text{a markiert}} p(a)$

1. Fall: $\gamma = 0 \Rightarrow 1 = \sum_{\text{a unmarkiert}} p(a)$
$(k+1-j$ unmarkiert$) \Rightarrow$ es existiert u unmarkiert in Teilphase j mit $p(u) \geq \frac{1}{k+1-j}$. Fordere u an.

2. Fall: $\gamma > 0$ Sei m eine markierte Seite mit $p(m) > 0$ fest gewählt für Teilphase j. Fordere m an.
Solange erwartete Kosten von ALG in dieser Teilphase $< \frac{1}{k+1-j}$ und $\gamma > p(m)$ fordere nochmals Seite m' an mit maximalen p(m').
Bricht ab, weil pro Durchlauf Kosten von:

$$\frac{\overbrace{\gamma}^{\text{variabel}}}{\underbrace{\#\text{ markierte Seiten}}_{\text{fest}}} \geq \frac{\overbrace{p(m)}^{\text{fest}}}{\#\text{ markierter Seiten}}$$

Falls jetzt erwartete Kosten $\geq \frac{1}{k+1-j}$: fordere irgendeine unmarkierte Seite an, fertig!
Sonst ($\gamma \leq p(m)$) : Fordere unmarkierte Seite u mit maximalen $p(u)$ an, markiere sie.
Im letzten Fall: erwartete Kosten von Teilphase j :

$$\geq p(m) + p(u) \geq p(m) + \frac{\overbrace{1-\gamma}^{30}}{\underbrace{k+1-j}_{31}} \geq p(m) + \tfrac{1-p(m)}{k+1-j} \geq \tfrac{1}{k+1-j} \quad \square$$

[30] Gesamtgewicht der unmarkierten Seiten
[31] Anzahl der unmarkierten Seiten

28.05.03

4 Online-Algorithmen und Spieltheorie

Warum? Bei der Analyse von Online Algorithmen:

ALG	$\Leftrightarrow$	OPT
Spieler		Gegenspieler

4.0 Elementares zu endlichen Zwei Personen Nullsummenspielen

Historisch: v. Neumann, O. Morgenstern, 1943: Game Theory and Economic Behavior.

Anschaulich:

Spiel: Menge von Regeln, die eindeutig festlegen, welche Züge jeder Spieler in jeder Situation machen darf.

Partie: Instanz eines Spiels $\Leftrightarrow$ zulässige Folge von Zügen, von Anfang bis Ende.

endliches Spiel:
- In jeder Situation nur endlich viele Züge möglich.
- Es gibt eine Schranke für die Maximallänge einer Partie.

Zwei Personen-Spiel: Es nehmen nur 2 Spieler teil: Weiß (1.Zug) und Schwarz.

Nullsummenspiel: Für jeden Endzustand ist festgelegt, welche Auszahlung Weiß erhält (kann auch negativ sein), diese ist von Schwarz zu leisten.

Beispiele: Nimm, Schach, Poker, Mühle[32], Monopoly[32], Mensch-Ärger-Dich-Nicht[32], Mau-Mau[32].
Zwei Unterschiede zwischen diesen Spielen:

1. Bei Monopoly, Mensch-Ärger-Dich-Nicht, Poker, Mau-Mau "wird gewürfelt". (Es finden Zufallszüge statt)

[32]Mit entsprechenden Abbruchbedingungen endlich

2. Bei Nimm, Mühle, Mensch-Ärger-Dich-Nicht, Schach: Spieler wissen über vorausgegangene Züge Bescheid und kennen die Situation. Aber bei Poker, Mau-Mau kennt man die Situation/Züge des Gegners nicht vollständig.

Beispiel 4.0.1 (Hölzchen (Variante von Nimm)) *Es gibt 4 Hölzchen, die Spieler nehmen abwechselnd 1 oder 2 Hölzchen fort. Wer das letzte Hölzchen nimmt, hat verloren.*

Darstellung eines Spiels: (1) Extensive Form mittels Spielbaum. (Siehe Abbildung 39) Beschrifte die Knoten von:

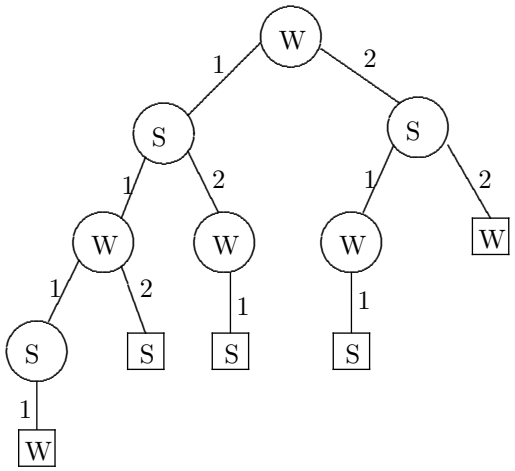


Abbildung 39: Spielbaum von Hölzchen

- Weiß mit der Maximalauszahlung, die Weiß in dieser Situation für sich erzwingen kann.
- Schwarz mit der Minimalauszahlung an Weiß, die Schwarz in dieser Situation erzwingen kann.

Beschriftung bottom-up: Beschriftung der Blätter durch Spielregeln festgelegt. Beschriftung der inneren Knoten:

2 Fälle: (Siehe Abbildung 40 auf der nächsten Seite)

Im Beispiel: Weiß gewinnt = +1, Weiß verliert = -1. Damit ergibt sich der Baum aus Abbildung 41 auf der nächsten Seite.

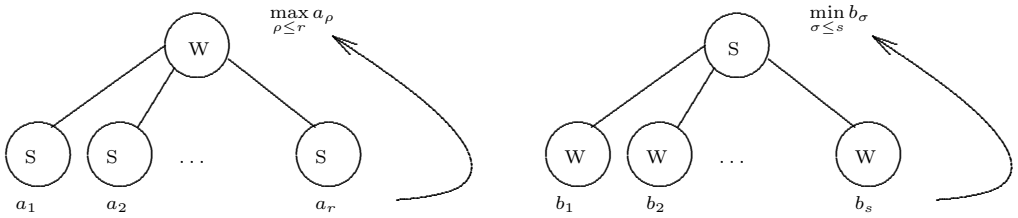


Abbildung 40: Beschriftung der Knoten des Spielbaums

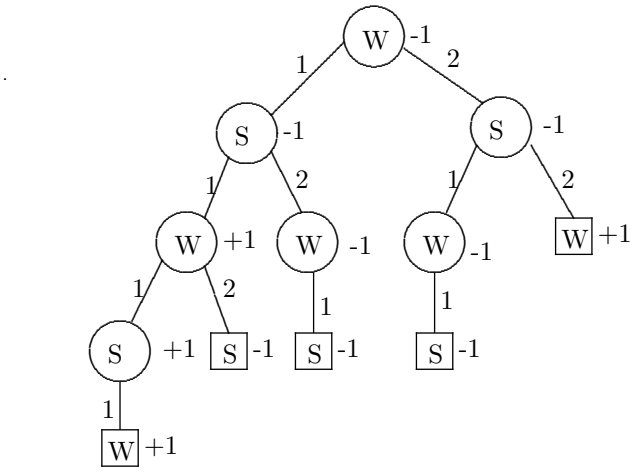


Abbildung 41: Spielbaum mit Bewertung

Hier: Weiß kann keine größere Auszahlung als -1 erreichen. (wenn Schwarz gut spielt) Schwarz kann erzwingen, dass er mindestens 1 von Weiß erhält.
$\rightsquigarrow$ "Schwarz hat Gewinnstrategie"

Strategie von Schwarz: Eine Folge von Funktionen

$$f_i : (w_1, s_1, w_2, s_2, \ldots, w_{i-1}, s_{i-1}, w_i) \to s_i$$

Salopp: Strategie von Schwarz: definiert seinen nächsten Zug in Abhängigkeit von dem [ihm bekannten] Stand der Partie.
Strategien von Weiß im Spiel "Hölzchen":

$$\underbrace{\begin{array}{ll} g_1 & \{1 \\ g_2 & \begin{cases} (1,1) \to 1 \\ (1,2) \to 1 \end{cases} \end{array}}_{\text{weiß}_1} \quad \underbrace{\begin{array}{ll} g_1 & \{1 \\ g_2 & \begin{cases} (1,1) \to 2 \\ (1,2) \to 1 \end{cases} \end{array}}_{\text{weiß}_2} \quad \underbrace{\begin{array}{ll} g_1 & \{2 \\ g_2 & \{(2,1) \to 1 \end{array}}_{\text{weiß}_3}$$

Strategien von Schwarz:

$$\underbrace{\begin{array}{ll} f_1 & \left\{\begin{array}{lll}(1) & \to & 1\\ (2) & \to & 1\end{array}\right. \\ f_2 & \{(1,1,1)\to 1\end{array}}_{\text{schwarz}_1} \quad \underbrace{\begin{array}{ll} f_1 & \left\{\begin{array}{lll}(1) & \to & 1\\ (2) & \to & 2\end{array}\right. \\ f_2 & \{(1,1,1)\to 1\end{array}}_{\text{schwarz}_2} \quad \underbrace{\begin{array}{lll} f_1 & \left\{\begin{array}{lll}(1) & \to & 2\\ (2) & \to & 1\end{array}\right. \\ - & - & \end{array}}_{\text{schwarz}_3} \quad \underbrace{\begin{array}{lll} f_1 & \left\{\begin{array}{lll}(1) & \to & 2\\ (2) & \to & 2\end{array}\right. \\ - & - & \end{array}}_{\text{schwarz}_4}$$

Nach Auswahl der Strategien stehen Partie und Auszahlung fest.
$\rightsquigarrow$ 2. Darstellung für Spiele: Auszahlungsmatrix.
$A = (a_{ij})$ mit a_{ij} = Auszahlung an Weiß, die von Schwarz zu leisten ist, falls Weiß Strategie i und Schwarz j spielt.
Im Beispiel Hölzchen: $A = \begin{pmatrix} 1 & 1 & -1 & -1 \\ -1 & -1 & -1 & -1 \\ -1 & 1 & -1 & 1 \end{pmatrix}$

Allgemein: Gegegeben Auszahlungsmatrix A des Spiels: Wieviel wird Weiß bei Wahl von Strategie i mindestens gewinnen: $\min_j a_{ij}$. (Das Minimum der i-ten Zeile, da der Gegner die Spalte j frei wählen kann)
Welche Strategie maximiert diesen Wert?
Jedes i_0 mit $\min_j a_{i_0 j} = \underbrace{\max_i \min_j a_{ij}}_{=-1\text{(bei Hölzchen)}}$ (gute Strategie)
Wieviel kann Schwarz bei der Wahl von Strategie j höchstens verlieren? $\max_i a_{ij}$
Welche Strategie von Schwarz minimiert diesen Wert?
Jedes j_0 mit $\max_i a_{ij_0} = \underbrace{\min_j \max_i a_{ij}}_{=-1}$

Lemma 4.0.2 *Sei A eine Auszahlungsmatrix. Dann gilt:*

$$\underbrace{\max_i \min_j a_{ij}}_{\textit{Mindestgewinn W.}} \leq \underbrace{\min_j \max_i a_i j}_{\textit{Maximalverlust von S.}}$$

Beweis 4.0.2: $\forall i,j : \underbrace{\min_j a_{ij} \leq a_{ij} \leq \max_i a_{ij}}_{\forall i}$
$\Rightarrow \forall j : \max_i \min_j a_{ij} \leq \max_i a_{ij}$
$\Rightarrow \max_i \min_j a_{ij} \leq \min_j \max_i a_{ij}$
Man kann nicht auch die andere Richtung beweisen.

Beispiel 4.0.3 (Stein-Schere-Papier) *Auszahlungsmatrix:*

$$\begin{array}{c} \\ Weiß \left\{ \begin{array}{r} St \\ Sch \\ Pa \end{array} \right. \end{array} \overbrace{\begin{pmatrix} 0 & 1 & -1 \\ -1 & 0 & 1 \\ 1 & -1 & 0 \end{pmatrix}}^{Schwarz}$$

$\max_i \min_j a_{ij} = -1$

$\min_j \max_i a_{ij} = 1$

Spielbaum von Stein-Schere-Papier: (Siehe Abbildung 42)

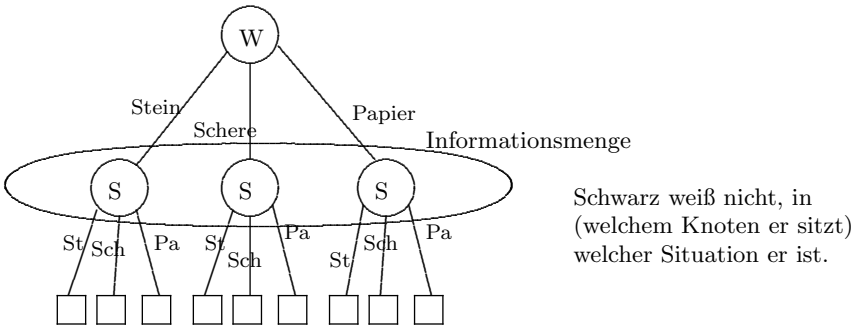


Abbildung 42: Spielbaum von Stein-Schere-Papier

Definition: Ein Spiel heißt eindeutig, falls gilt:
$\max_i \min_j a_{ij} = \min_j \max_i a_{ij}$ (i.A. nur "$\leq$")

Theorem 4.0.4 *Jedes endliche 2-Personen-Nullsummenspiel mit vollständiger Information ist eindeutig.*

Beweis 4.0.4: Annotiere Spielbaum, wie gehabt.

Korollar (Zemento): Entweder hat Weiß beim Schach eine Gewinnstrategie oder Schwarz hat eine oder beide können Remis erzwingen.

2.06.03

Definition: (i_0, j_0) heißt Sattelpunkt von der Matrix A, falls gilt: $a_{i_0 j_0}$ ist Minimum der i_0-ten Zeile und Maximum der j_0-ten Spalte.

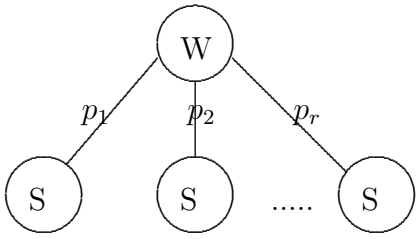
Lemma 4.0.5 *Spiel γ ist eindeutig $\Leftrightarrow$ Auszahlungsmatrix hat Sattelpunkt. Dann gilt* $\max_i \min_j a_{ij} = a_{i_0 j_0} = \min_j \max_i a_{ij} = \nu$ *(Wert des Spiels)*

Beweis 4.0.5 "⇒" Die Funktion $i \to \max_i a_{ij}$ habe ihr Maximum bei i_0.
Die Funktion $j \to \min_j a_{ij}$ habe ihr Minimum bei j_0.
Behauptung: (i_0, j_0) ist Sattelpunkt.
$\forall i : a_{ij_0} \leq \max_i a_{ij_0} \overset{\text{Def.}j_0}{=} \min_j \max_i a_{ij}$
$\overset{\gamma \text{ eindeutig}}{\Rightarrow} \max_i \min_j a_{ij} \overset{\text{Def.}i_0}{=} \min_j a_{i_0 j} \leq a_{i_0 j_0}$
$\Rightarrow a_{i_0 j_0}$ ist Maximum von Spalte j_0.
$\forall j : a_{i_0 j_0} \leq \max_i a_{ij_0} \overset{\text{wie oben}}{=} \min_j a_{i_0 j} \leq a_{i_0 j}$
$\Rightarrow a_{i_0 j_0}$ ist Minimum der Zeile i_0.
"⇐" Gelte: $\max_i a_{ij_0} = a_{i_0 j_0} \min_j a_{i_0 j}$
$\max_i a_{ij0} \geq \min_j \max_i a_{ij} \leq \max_i \min_j a_{ij} \geq \min_j a_{i_0 j}$
$\overset{\text{L. 4.02}}{\Rightarrow} \gamma$ eindeutig.

Achtung: Auch wenn γ eindeutig ist und $a_{ij} = \nu$ gilt braucht (i, j) kein Sattelpunkt zu sein (Ü).

Frage: Was ist mit Spielen *ohne* vollständige Information? (z.B. Poker, Stein-Schere-Papier)

Idee: (v. Neumann/Morgenstern) Randomisieren. Bisher betrachtet: reine Strategien. Erlaube jetzt jedem Spieler Zufallsentscheidungen zu treffen. (Siehe Abbildung 43) Behavionistische Strategie.



wobei: $p_i \geq 0$ und $\sum_{i=1}^{r} p_i = 1$

Abbildung 43: Zufallsentscheidungen

Annahme:

- Spiel ist endlich.
- Speicherplatz ist unbeschränkt.

$\Rightarrow$ Man kann alle Zufallsentscheidungen im Voraus treffen.
$\Rightarrow$ So entsteht für jeden Spieler eine Wahrscheinlichkeitsverteilung auf den reinen Strategien.

Damit: Spieler Weiß entscheidet sich vor Spielbeginn für eine Wahrscheinlichkeitsverteilung $p_1, \ldots, p_n$ auf seinen reinen Strategien $s_1, \ldots, s_n$, dass heißt für einen Vektor $\vec{p} = \begin{pmatrix} p_1 \\ \vdots \\ p_n \end{pmatrix}$ mit $\forall i : p_i \geq 0, \sum_{i=1}^{n} p_i = 1$ (Analog für Vektor $\vec{q}$ bei Schwarz)
Auszahlung: Statt einer Zahl a_{ij} aus A: Erwartungswert $\sum_{i,j} p_i a_{ij} q_j = \tilde{A}(\vec{p}, \vec{q}) = \vec{p}^t A \vec{q}$. Bilinearform.

Theorem 4.0.6 (v. Neumann, Morgensterns Minimax-Theorem) *Jedes endliche 2-Personen-Nullsummenspiel hat einen Wert bei Verwendung gemischter Strategien*
d.h. $\max_{\vec{p}} \min_{\vec{q}} \tilde{A}(\vec{p}, \vec{q}) = \min_{\vec{q}} \max_{\vec{p}} \tilde{A}(\vec{p}, \vec{q}) = \nu =$ *Wert des Spiels*

Beweis 4.0.6: Sei $A \in M_{n \times m}(\mathbb{R})$ gegeben.

Behauptung 1: Verwendung von max, min (statt inf, sup) ist korrekt!
z.b. die Vektoren $\vec{p}$ entstammen der Menge
$N^m = \left\{ \begin{pmatrix} x_1 \\ \vdots \\ x_n \end{pmatrix}, x_i \geq 0 \text{ und } \sum_{i=1}^{n} x_i = 1 \right\}$ konvex, kompakt.
Entsprechend $\vec{q} \in N^n$.
$\tilde{A} : N^n \times N^m \to \mathbb{R}$ stetig, bilinear.

Behauptung 2: (Kern des Beweises) Es gibt: $\vec{q} \in N^m$ mit $A\vec{q} \leq 0$ für alle Komponenten oder es gibt $\vec{p} \in N^n$ mit $A^t \vec{p} \geq 0$ für alle Komponenten.

Beweis 2: Sei $\mathcal{a}_j = \begin{pmatrix} a_{1j} \\ a_{2j} \\ \vdots \\ a_{nj} \end{pmatrix}$ die j-te Spalte der Matrix A.

Sei $\mathcal{v}_i = \begin{pmatrix} 0 \\ \vdots \\ 0 \\ 1 \\ 0 \\ \vdots \\ 0 \end{pmatrix}$ der i-te Einheitsvektor aus $\mathbb{R}^n$. Sei C die konvexe Hülle der Vektoren $\mathcal{a}_1, \ldots, \mathcal{a}_m$ und $\mathcal{v}_1, \ldots, \mathcal{v}_n$ im $\mathbb{R}^n$

$$\Rightarrow C = \left\{ \sum_{\mu=1}^{m} t_\mu \mathcal{a}_\mu + \sum_{\nu=1}^{n} s_\nu \mathcal{v}_\nu \ ; \ t_\mu, s_\nu \geq 0 \text{ und } \sum_{\mu=1}^{m} t_\mu + \sum_{\nu=1}^{n} s_\nu = 1 \right\}$$

Fall 1 C enthält $\vec{0}$

$$\Rightarrow 0 = \sum_{\mu=1}^{m} t_\mu \mathcal{a}_\mu + \sum_{\nu=1}^{n} s_\nu \mathcal{v}_\nu$$

$\Rightarrow \sum_{\mu=1}^{m} t_\mu > 0$, denn die $\mathcal{v}_\nu$ sind linear unabhängig und $\sum t_\mu + \sum s_\nu = 1$

Schaue Zeile ν an: $\sum_{\mu=1}^{m} t_\mu a_{\nu\mu} = -s_\nu \leq 0$

Setze $q_j = \frac{t_j}{\sum_{i=1}^{m} t_u} \Rightarrow \mathcal{y} = \begin{pmatrix} q_1 \\ \vdots \\ q_m \end{pmatrix} \in N^m$

und $A\mathcal{y} = \frac{1}{\sum_{j=1}^{m} t_i} \begin{pmatrix} -s_1 \\ \vdots \\ -s_n \end{pmatrix} \leq 0 \surd$

Fall 2 C enthält nicht $\vec{0}$

C konvex $\Rightarrow$ Es existiert Hyperebene H mit:

- $\vec{0} \in H$
- C liegt ganz auf der einen Seite von H

Siehe Abbildung 44 auf der nächsten Seite.

$$H = \left\{ \begin{pmatrix} x_1 \\ \vdots \\ x_n \end{pmatrix} \in \mathbb{R}^n; \sum_{i=1}^{n} h_i x_i = 0 \right\} \text{ und } C \subseteq H^+$$

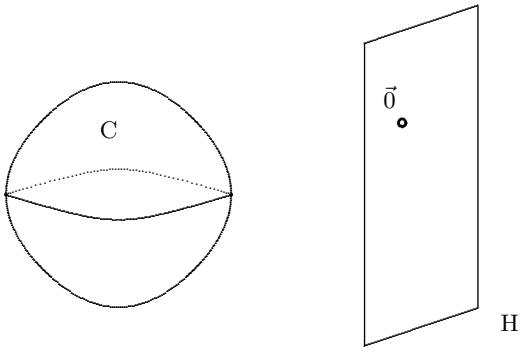


Abbildung 44: Hyperebene H

$$H^+ = \left\{ \begin{pmatrix} x_1 \\ \vdots \\ x_n \end{pmatrix} \in \mathbb{R}^n; \sum_{i=1}^{n} h_i x_i > 0 \right\}$$

Trick $\mathscr{w}_\nu \in H^+ \Rightarrow h_\nu > 0 \forall \nu$

Setze $p_i = \frac{h_i}{\sum_{i=1}^{n} h_i} \Rightarrow p = \begin{pmatrix} p_1 \\ \vdots \\ p_n \end{pmatrix} \in N^n$ und wegen $a_\mu \in C \subseteq H^+$

$$\Rightarrow \sum_{i=1}^{n} h_i a_{\mu i} > 0$$

$$\Rightarrow 0 < \begin{pmatrix} \sum_i a_{i1} p_i \\ \vdots \\ \sum_i a_{n1} p_i \end{pmatrix} = A^t p = (p^t A)^t$$

$\Rightarrow$ Behauptung 2.

Lemma 4.0.7 (Loomis Lemma) *(i)* $\forall \mathscr{y} \in N^m : \max_{\mathscr{p} \in N^n} \tilde{A}(\mathscr{p}, \mathscr{y}) = \max_{1 \le i \le n} \tilde{A}(\mathscr{w}_i, \mathscr{y})$

(ii) $\forall p \in N^n : \min_{\mathscr{y} \in N^m} \tilde{A}(\mathscr{p}, \mathscr{y}) = \min_{1 \le j \le m} \tilde{A}(\mathscr{p}, \mathscr{w}_j)$

"Gegen eine bekannte gemischte Strategie braucht man nicht zu randomisieren"

Beweis 4.0.7: (i): Sei $\mathscr{y}$ beliebig und $\mathscr{p} = \begin{pmatrix} p_1 \\ \vdots \\ p_n \end{pmatrix}$

$= \sum_{i=1}^{n} p_i \mathscr{w}_i$. Dann gilt:

$\max_{1\le i\le n} \tilde{A}(\mathscr{w}_i, \mathscr{y}) = \sum_{\nu=1}^{n} p_\nu \max_{1\le i\le n} \tilde{A}(\mathscr{w}_i, \mathscr{y})$
$\geq \sum_{\nu=1}^{n} \tilde{A}(\mathscr{w}_\nu, \mathscr{y}) \overset{33}{=} \tilde{A}(\mathscr{p}, \mathscr{y}) \forall \mathscr{p}$
$\Rightarrow \max_{1\le i\le n} \tilde{A}(\mathscr{w}_i, \mathscr{y}) \geq \max_{\mathscr{p}} \tilde{A}(\mathscr{p}, \mathscr{y})$ Richtung $\leq$ trival
$\Rightarrow$ "=" ((ii) analog wie (i))

Damit Beweis des Theorems 4.0.6 auf Seite 49: Zu zeigen: $\nu_1 = \max_{\mathscr{p}} \min_{\mathscr{y}} \tilde{A}(\mathscr{p}, \mathscr{y}) = \min_{\mathscr{y}} \max_{\mathscr{p}} \tilde{A}(\mathscr{p}, \mathscr{y}) = \nu_2$
($\leq$ trival, siehe Lemma 4.0.2)
zeigen: Zwischen ν_1, ν_2 ist kein Platz für eine reelle Zahl.
Angenommen in Behauptung 2 gilt: $\exists \mathscr{y} \in N^m$ mit: $A\mathscr{y} \leq 0$
$\Rightarrow \forall i, 1 \leq i \leq n : \underbrace{\sum_{j=1}^{m} a_{ij} q_i}_{=\tilde{A}(\mathscr{w}_i, \mathscr{y})} \leq 0$
$\Rightarrow \max_{\mathscr{p}} \tilde{A}(\mathscr{p}, \mathscr{y}) \overset{4.0.7}{=} \max_i \tilde{A}(\mathscr{w}_i, \mathscr{y}) \leq 0$
$\Rightarrow \nu_1 \leq \nu_2 = \min_{\mathscr{y}} \max_{\mathscr{p}} \tilde{A}(\mathscr{p}, \mathscr{y}) \leq 0$
Angenommen in Behauptung 2 gilt: $\exists \mathscr{p} \in N^n$ mit: $A^t p \geq 0$
$\Rightarrow \forall j, 1 \leq j \leq m : \underbrace{\sum_{i=1}^{n} p_i a_{ij}}_{\tilde{A}(\mathscr{p}, \mathscr{w}_j)} \geq 0$
$\overset{\text{wie oben}}{\Longrightarrow} 0 \leq \max_{\mathscr{p}} \min_{\mathscr{y}} \tilde{A}(\mathscr{p}, \mathscr{y}) = \nu_1 \leq \nu_2$
$\Rightarrow$ Nie gilt $\nu_1 < 0 < \nu_2$
Sei $w \in \mathbb{R}$ beliebig. Betrachte die Matrix $B = (b_{ij})$ mit $b_{ij} = a_{ij} + w$
$\tilde{B}(\mathscr{p}, \mathscr{y}) = \mathscr{p}^t B \mathscr{y} = \sum_{i,j} p_i b_{ij} q_j$
$= \sum_{i,j} p_i a_{ij} q_i + w \underbrace{\sum_{i,j} p_i, q_i}_{=1}$
Gerade gezeigt (für B statt A) gilt nie: $\nu_1(B) < 0 < \nu_2(B)$
$\Leftrightarrow \nu_1(A) + w < 0 < \nu_2(A) + w$
$\Rightarrow$ Es gilt nicht: $\nu_1(A) < -w < \nu_2(A)$, wobei w beliebig ist. $\square$

[33] Da $\tilde{A}$ bilinear

16.06.03 Konsequenz: Jedes endliche Zwei-Personen-Nullsummenspiel hat einen Wert bei Verwendung gemischter Strategien[34].

Theorem 4.0.8 (Volles Loomis Lemma) *Sei Γ ein EZPN Spiel mit Auszahlungsmatrix A. Die Abbildung*

$$f: \begin{array}{ccc} N^n & \to & \mathbb{R} \\ \mathscr{p} & \to \min\limits_{\mathscr{q} \in N^m} \tilde{A}(\mathscr{p}, \mathscr{q}) & \text{habe Maximum in } \mathscr{p}^* \end{array}$$

$$g: \begin{array}{ccc} N^m & \to & \mathbb{R} \\ \mathscr{q} & \to \max\limits_{\mathscr{p} \in N^m} \tilde{A}(\mathscr{p}, \mathscr{q}) & \text{habe Minimum in } \mathscr{q}^* \end{array}$$

Dann ist $(\mathscr{p}^*, \mathscr{q}^*)$ "Sattelpunkt" des Spiels und es gilt:
$\nu = \tilde{A}(\mathscr{p}, \mathscr{q}) = \max\limits_i \tilde{A}(\mathscr{v}_i, \mathscr{q}^*) = \min\limits_j \tilde{A}(\mathscr{p}^*, \mathscr{v}_j) = \min\limits_{\mathscr{q}} \max\limits_i \tilde{A}(\mathscr{v}_i, \mathscr{q}) = \max\limits_{\mathscr{p}} \min\limits_j \tilde{A}(\mathscr{p}, \mathscr{v}_j)$

Beweis 4.0.8: Siehe Lemmata 4.0.7 auf Seite 51 und 4.0.5 auf Seite 47.

Beispiel 4.0.9 $A = \begin{pmatrix} 3 & 1 \\ 2 & 4 \end{pmatrix}$
$\max\limits_i \min\limits_j a_{ij} = 2 < 3 = \min\limits_j \max\limits_i a_{ij}$ *Spiel hat keinen Wert bei Verwendung reiner Strategien. Betrache gemischte Strategie für Weiß: Wähle Zeile 1 mit Wahrscheinlichkeit p und Zeile 2 mit Wahrscheinlichkeit* $1-p$
$\Rightarrow \nu = \max\limits_{\mathscr{p}} \min\limits_j \underbrace{\tilde{A}(\mathscr{p}, \mathscr{v}_j)}_{\sum\limits_{i=1}^{2} p_i a_{ij}} = \max\limits_{\mathscr{p}} \min\left(p \cdot 3 + (1-p) \cdot 2, p \cdot 1 + (1-p) \cdot 4\right) =$
$\max\limits_{\mathscr{p}} \min(p+2, -3p+4)$ *wird maximal, wenn* $p + 2 = -3p + 4 \Leftrightarrow p = \frac{1}{2}$.

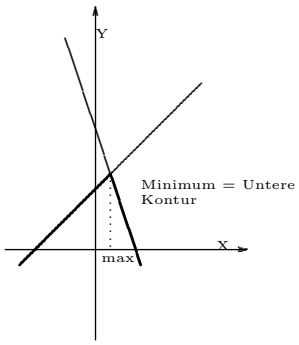


Abbildung 45: Funktionsgraphen bei Weiß

[34]Wahrscheinlichkeitsverteilung auf den reinen Strategien

(Siehe Abbildung 45 auf der vorherigen Seite.)
$\Rightarrow$ *Optimale Strategie für Weiß ist:* $(\frac{1}{2}, \frac{1}{2})$.
$\Rightarrow \nu = \frac{1}{2} + 2 = \frac{5}{2}$
Wenn Schwarz wüsste, dass Weiß $\mathscr{p}^* = (\frac{1}{2}, \frac{1}{2})$ *spielt, könnte er mit jeder seiner reinen Strategien kontern. Aber Schwarz weiß das nicht. Kann es sich nicht leisten z.B. Spalte 1 zu wählen, denn er weiß nicht, ob Weiß nicht Zeile 1 spielt. Gute gemischte Strategie* $\mathscr{y}^*$ *für Schwarz wie vorhin:*
$\nu = \min_{\mathscr{y}} \max(3q + 1(1-q), 2q + 4(1-q))$
$= \min_{\mathscr{y}} \max(2q + 1, -2q + 4) \Rightarrow$ *Minimum wird angenommen für* $2q + 1 =$

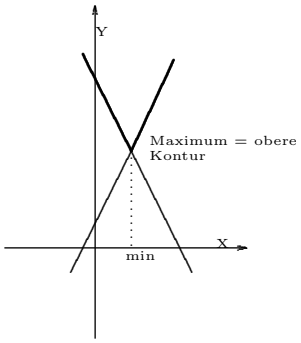


Abbildung 46: Funktionsgraphen bei Schwarz

$-2q + 4 \Rightarrow q = \frac{3}{4}, 1 - q = \frac{1}{4}$. *(Siehe Abbildung 46.) Optimale Strategie für Schwarz:* $(\frac{3}{4}, \frac{1}{4})$

Achtung: Minmax-Theorem gilt leider nicht für alle unendlichen Spiele!

Beispiel 4.0.10 *Spieler Weiß wählt* $i \in \mathbb{N}$
Spieler Schwarz wählt $j \in \mathbb{N}$
$a_{ij} = \begin{cases} 1, & i > j \\ 0, & i = j \\ -1, & i < j \end{cases}$ *Strategie für Weiß:* $\mathscr{p} = (p_1, p_2, \ldots)$ *mit* $p_i \geq 0$ *und* $\sum_{i=1}^{\infty} p_i = 1$ *(Diskrete Wahrscheinlichkeitsverteilung)*
$\Rightarrow \forall \epsilon > 0\ \exists n_\epsilon : \sum_{i=n_\epsilon}^{\infty} p_i < \epsilon \Rightarrow \sup_{\mathscr{p}} \inf_j \tilde{A}(\mathscr{p}, \mathscr{v}_j) = -1$, *denn: für jedes feste* $\mathscr{p}$ *gilt:* $\tilde{A}(\mathscr{p}, \mathscr{v}_j) = -1 \sum_{i=1}^{j-1} p_i + 0 + \underbrace{1 \sum_{i=j+1}^{\infty} p_i}_{\searrow 0 \text{ für } j \to \infty}$ *Für jedes* $\mathscr{p} : \inf_j \tilde{A}(\mathscr{p}, \mathscr{v}_j) = -1 \Rightarrow$

$\sup_{\mathscr{P}} \inf_{j} \tilde{A}(\mathscr{P}, \mathscr{W}_j) = -1$
Entsprechend: $\inf_{\mathscr{Y}} \sup_{i} \tilde{A}(\mathscr{W}_i, \mathscr{Y}) = +1$ *analog.*

Hauptproblem der Spieltheorie: Mehr als 2 Personen.
Keine Nullsummenspiele: Leicht heilbar durch Einführung eines weiteren Spielers.

Zurück zur Online Analyse Allgemeines Modell für Online-Probleme:

1. Anforderungs-Antwort-Systeme (ähnlich zu 2 Personen Spielen).
2. Metrische Task Systeme.
3. k-Server Modell.

4.1 Anforderungs-Antwort-Systeme

Definition Anforderungs-Antwort-System: Gegeben sei eine Anforderungsmenge $\mathcal{R}$, Folge von endlichen Antwortmengen $A_i, i = 1, 2, \ldots$ und eine Folge von Funktionen $\text{cost}_n : \mathcal{R}^n \times A_1 \times \ldots \times A_n \to (\mathbb{R}_{\geq 0} \cup \{\infty\})$
Idee: Abwechselnd stellt Gegner ADV eine Anforderung $r_i \in \mathcal{R}$,
beantwortet Algorithmus ALG diese Anforderung mit Antwort $a_i \in A_i$
Wird Anforderungsfolge $r_1, \ldots, r_n$ mit Antwortfolge $a_1, \ldots, a_n$ beantwortet, so entstehen Kosten $\text{cost}_n(r_1, \ldots, r_n, a_1, \ldots, a_n)$
(Zeitliche Abfolge: $r_1, a_1, r_2, a_2, \ldots, r_n, a_n$)
Dann heißt $(\mathcal{R}, (A_i)_i, (\text{cost}_i)_i)$ ein Anforderungs-Antwort-System. Damit lassen sich verschiedene Online Probleme beschreiben. z.B.

Paging: $\mathcal{R} = \{1, \ldots, N\}$($N$ = Anzahl sämtlicher Seiten)
$A_i = A = \{0, 1, \ldots, k\}$ wobei k = Cache Größe und 0 = NOP, $1 \leq i \leq k$: Verdränge Seite i.
cost_n entsprechend:
$\text{cost}_n(r_1, \ldots, r_n, a_1, \ldots, a_n) = \#\{m, 1 \leq m \leq n \text{ und } a_m \neq 0\}$

Selbstorganisierende Listen (Ü)

Definition Online Algorithmus: Ein Online Algorithmus ALG für ein Anforderungs-Antwort-System $\mathcal{T}$ ist:
Folge von Funktionen $g_i : \begin{array}{ccc} \mathcal{R}^i & \to & A_i \\ (r_1, \ldots, r_i) & \to & a_i \end{array}$ Dass heißt ALG kennt alle

bisherigen Anforderungen. (Man könnte auch schreiben

$$gi: \begin{array}{rcl} (\mathcal{R} \times A)^{i-1} \times R & \to & A_i \\ (r_1, a_1, \ldots, r_{i-1}, a_{i-1}, r_i) & \to & a_i \end{array})$$

Wir schreiben ALG $= \{g_i\}_i$
Mit einer Anforderungsfolge $\sigma = (r_1, r_2, \ldots, r_n)$ sei
ALG $[\sigma] = (a_1, a_2, \ldots, a_n) \in A_1 \times A_2 \times \ldots \times A_n$
mit $a_j = g_j(r_1, \ldots, r_j)$ für $j = 1, \ldots, n$
$(a_1, \ldots, a_n) =$ ALG$[\sigma]$ heißt die von ALG zu σ produzierte Antwortfolge.
Kosten: ALG$(\sigma) = \text{cost}_n(\sigma,$ ALG$[\sigma])$

4.2 Randomisierte Algorithmen

Hatten Online Probleme gesehen, bei denen Randomisierung Vorteile bringt, in dem ALG seine jeweils nächste Antwort zufällig bestimmt. (behavionistischer Ansatz)

Beispiele: BIT bei Listen, MARK bei Paging.

Annahmen:

1. Ausreichend Speicherplatz.
2. Kein Spieler vergißt seine Züge.

Randomisierter Algorithmus ALG = Wahrscheinlichkeitsverteilung auf einer Familie von deterministischen Algorithmen $(\text{ALG}_\omega)_\omega$ mit $\omega \in \Omega, (\Omega, \mu)$ Wahrscheinlichkeitsraum. (Meistens μ diskret, d.h. $\sum_{\omega \in \Omega} \mu(\omega) = 1$)
Kosten von ALG = ALG $(\sigma) = E_\omega\,(\text{ALG}_\omega(\sigma))$

18.06.03

4.3 Deterministische Gegenspieler

Allgemein: Gegenspieler ADV $= (Q, S)$ mit $Q =$ Anforderungskomponente. $S =$ Bedienkompente (d.h. ADV muß die von ihm mit Q erzeugten Anforderungen evtl. auch selbst beantworten)

Wir betrachten 3 Typen von Gegenspielern

1. Vergeßlicher Gegner (oblivious adversary)
   - Kennt ALG und μ, aber nicht das ausgewählte ALG $_\omega$

- darf selbst am Ende die optimale Offline Lösung liefern.

Wie bisher betrachtet, d.h. Q entscheidet vor Beginn für endliche Anforderungsfolge $\sigma \in \mathcal{R}^n$
S erzeugt optimale Offline Lösung. OPT $[\sigma]$ = diejenige Antwortfolge $a_1, \ldots, a_n$ mit $\text{cost}_n(\sigma, a_1, \ldots, a_n)$ minimal = OPT (σ)
Sei $\gamma(t) = c \cdot t + A$ (Kompetitiver Faktor c und additive Konstante A in einem)
ALG ist $\gamma-$ kompetitiv gegen vergeßliche Gegner
$:\Leftrightarrow \forall \sigma : \text{ALG}(\sigma) \leq \gamma(\text{OPT}(\sigma))$.

2. Adaptiver Offline Gegner (adaptive offline adversary)

   - Kennt ALG und sieht die zur Laufzeit von ALG $_\omega$ erzeugten Antworten.
   - Darf selbst am Ende die optimale Lösung liefern.

   Q = Folge von Funktionen q_i : $\bigotimes_{j=1}^{i-1} \mathcal{R} \times A_j \rightarrow \mathcal{R} \cup \{\text{stop}\}$
   $(r_1, a_1, \ldots, r_{i-1}, a_{i-1}) \rightarrow r_i$
   Q muss in Abhängigkeit von ALG eine Konstante d_Q benennen mit $q_{d_Q}\left(r_1, a_1, \ldots, r_{d_Q-1}, a_{d_Q-1}\right) = \text{stop}$
   Sei $\sigma(\text{ALG}_\omega, Q)$ die auf diese Weise erzeugte Anforderungsfolge $= (r_1, r_2, \ldots, r_{d_Q})$
   S wie bei OBL, d.h. S erzeugt optimale Offline Lösung OPT $[\sigma(\text{ALG}_\omega, Q)]$
   ALG heißt $\gamma-$ kompetitiv gegen adaptive Offline Gegner
   $:\Leftrightarrow \forall Q : E_\omega\left(\text{ALG}_\omega\left(\sigma(\text{ALG}_w, Q)\right)\right) \leq \gamma\left(E_\omega\left(\text{OPT}(\sigma(\text{ALG}_w, Q))\right)\right)$

3. Adaptiver Online Gegner

   - Kennt ALG und sieht die zur Laufzeit von ALG $_\omega$ erzeugten Antworten.
   - Muss selbst noch zur Laufzeit antworten, d.h.
     Q: Wie beim Adaptiven Offline-Gegner
     S: Folge von Funktionen p_i : $\left(\bigotimes_{j=1}^{i-1} \mathcal{R} \times A_j\right) \times \mathcal{R} \rightarrow A_i$
     $((r_1, a_1, \ldots, r_{i-1}, a_{i-1}), r_i) \rightarrow b_i$
     d.h. S kennt alle früheren Anfordungen und alle Antworten von ALG (← Könnte man den Gegner verbieten. Dann blinder adaptiver Online-Gegner)

Sei $b(\text{ALG}_\omega, Q, S)$ die von S erzeugte Antwortfolge $b_1, b_2, \ldots$ (ADV blind: $S(\sigma(\text{ALG}_\omega, Q))$) selbst ein gewöhnlicher Online Algorithmus.

ALG heißt $\gamma-$ kompetitiv gegen adaptive Online Gegner
$\Leftrightarrow \forall ADV(Q,S) : E(\text{ALG}_\omega(\sigma(\text{ALG}_w, Q)))$
$\leq \gamma \left(E_\omega(\text{cost}_{d_q}(\sigma(\text{ALG}_\omega, Q), b(\text{ALG}_w, Q, S)))\right)$

Fragen

1. Wie stark sind diese Gegnertypen?
   Klar: vergeßlicher Gegner < adaptiver Online Gegner < adaptiver Offline Gegner.

2. Hilft Gegnern Randomisierung?

4.4 Randomisierte Gegenspieler

Hatten gesehen ALG können von Randomisierung sehr profitieren, z.B. Paging: log k-kompetitiv statt k-kompetitiv. Beim Gegner auch?

Theorem 4.4.1 *Bei Anforderungs-Antwort-System bringt Randomisierung keinen der 3 Gegnertypen einen Vorteil.*

Beweis 4.4.1: Sei ALG $=(\text{ALG}_i)_i$ mit diskreter Wahrscheinlichkeitsverteilung $p_i, p_i \geq 0, \sum p_i = 1$ Es genügt die Behauptung für den adaptiven Online Gegner zu zeigen. (Denn da ist der Beweis am schwierigsten) Sei ADV ein randomisierter Online-Gegner.
ADV$= \left((Q_k, a_j)_j, (S_k, w_k)_k\right)$ mit diskreten Wahrscheinlichkeitsverteilungen $(q_j)_j$ und $(w_k)_k$.
Für jede feste Wahl von Q_j und S_k gilt:
Erwartete Kosten von ALG $= \sum_i p_i \cdot \text{ALG}_i\left(\sigma(\text{ALG}_i, Q_j)\right) = a(j)$
Erwartete Kosten von ADV$(Q_j, S_k) = \sum_i p_i \cdot \text{cost}_{d_Q}\left(\sigma(\text{ALG}_i, Q_j), b(\text{ALG}_i, Q_j, S_k)\right) =$
$b(j,k)$

Klar: Deterministischer Gegenspieler kann für jedes $\epsilon > 0$ j^* und k^* so wählen, dass $\frac{a(j^*)}{b(j^*,k^*)} \geq \sup_{j,k} \frac{a(j)}{b(j,k)} - \epsilon$
Wir zeigen: Auch der randomisierte Gegenspieler kann nicht mehr erreichen!
Denn: $\frac{\text{Erwartete Kosten von ALG gegen ADV}}{\text{Erwartete Kosten von ADV}}$
$= \frac{\sum_j q_j a(j)}{\sum_{j,k} q_j w_k b(j,k)}$

$$= \frac{\left(\sum_j q_j a(j)\right) \cdot \overbrace{\sum_k w_k}^{=1}}{\sum_{j,k} q_j w_k b(j,k)} = \frac{\sum_{j,k} q_i w_k a(j)}{\sum_{j,k} q_j w_k b(j,k)}$$

Trick: $\frac{\sum_i A_i}{\sum_i B_i} \overset{B_i>0}{\leq} \sup_i \frac{A_i}{B_i}$ Denn: $\forall i : A_i \leq l \cdot B_i \Rightarrow \sum_i A_i \leq \sum_i l B_i = l \sum_i B_i$

$\overset{\text{Trick}}{=} \sup_{j,k} \frac{a(j)}{b(j,k)}$ □

Korollar 4.4.2 *Sei* ALG *ein randomisierter Online Algorithmus,* ALG γ*-kompetitiv gegen jeden deterministischen Online Gegner* ⇒ ALG γ*-kompetitiv gegen jeden randomisierten adaptiven Online Gegner.*

4.5 Stärke der deterministischen Gegner

Theorem 4.5.1 *Für ein Anforderungs-Antwort-System* $\mathcal{T}$ *sei* ALG γ*-kompetitiv gegen adaptiven Online Gegner* ALG *',* ALG δ*-kompetitiv gegen vergeßliche Gegner. Dann ist* ALG $\gamma \cdot \delta$*-kompetitiv gegen Offline Gegner*

Beweis 4.5.1: Sei Q das Anforderungssystem eines beliebigen adaptiven Offline Gegners. Seien $\text{ALG} = (\text{ALG}_\omega)_\omega$, ALG '$= \left(\text{ALG}'_\psi\right)_\psi$.

Müssen zeigen: $E\left(\text{ALG}_\omega(\sigma(\text{ALG}_\omega, Q))\right) \leq \gamma\left(\delta\left(E(\text{OPT}(\sigma(\text{ALG}_w, Q)))\right)\right)$
Betrachten jetzt den folgenden blinden adaptiven Online Gegner:
$\overline{\text{ADV}} = (Q, \text{ALG}')$, randomisiert
4.4.2⇒ $E_\omega(\text{ALG}_\omega(\sigma(\text{ALG}_\omega, Q))) \leq \gamma(E_\omega E_\psi(\text{ALG}'_\psi(\sigma(\text{ALG}_w, Q))))$
⇒ ALG ist γ-kompetitiv gegen ADV.

Andererseits: ALG ' ist δ-kompetitiv gegen vergeßliche Gegner.
$E_\psi(\text{ALG}'_\psi(\sigma(\text{ALG}_\omega, Q))) \leq \delta(\text{OPT}(\sigma(\text{ALG}_w, Q)))$
Jetzt auf beiden Seiten E_ω und γ anwenden ⇒ Behauptung □

23.06.03

Korollar 4.5.2 *Gibt es einen* γ*-kompetitiven Algorithmus* ALG *gegen adaptive Online Gegner, so ist* ALG γ^2*-kompetitiv gegen adaptive Offline Gegner.*

Beweis 4.5.2: Theorem 4.5.1.

Wir zeigen jetzt: Adaptive Offline Gegner sind so stark, dass Randomisierung gegen sie nicht hilft.

Theorem 4.5.3 *Sei $\mathcal{T}$ ein Anforderungs-Antwort-System. Angenommen, es gibt einen randomisierten, γ-kompetitiven Algorithmus* ALG *gegen adaptive Offline Gegner. Dann gibt es auch einen γ-kompetitiven deterministischen Algorithmus*[35]

Beweis 4.5.3: Zu $\mathcal{T}$ gehört ein Baum T: (Siehe Abbildung 47) T kann un-

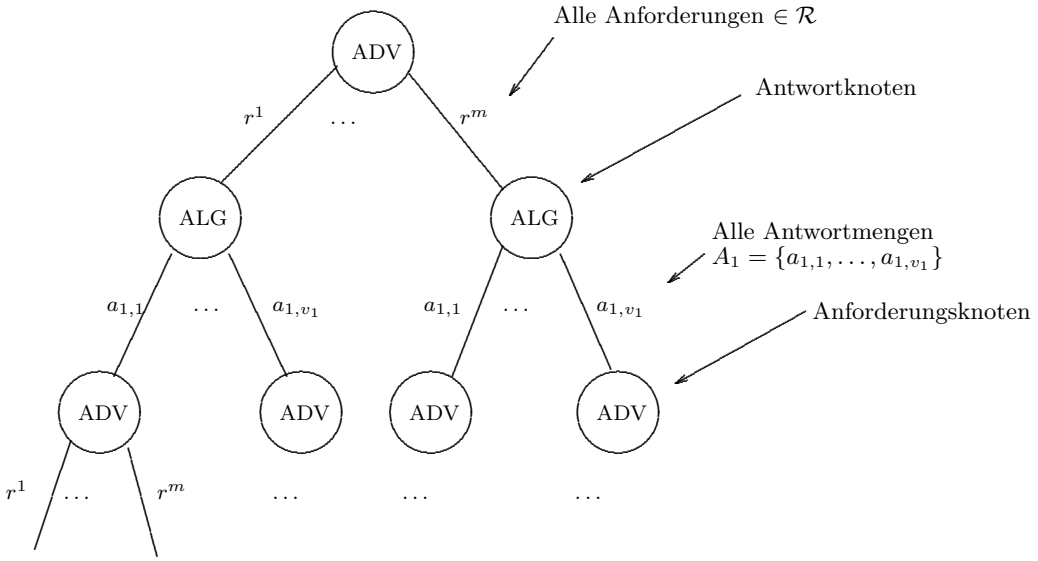


Abbildung 47: Baum zum Anforderungs-Antwort-System

endlich sein!
Zu jedem Anforderungsknoten v auf Tiefe $2n + 1$ gehört eine eindeutige Folge $r_1, a_1, \ldots, r_n, a_n$ durch Beschriftungen auf dem Pfad von der Wurzel von v Sei $\sigma = r1, \ldots, r_n$ und $a = a_1, \ldots, a_n$. Setze $\text{cost}(v) = \text{cost}_n(\sigma, a)$. Jetzt partielle Färbung von T, abhängig vom Kompetitivitätsfaktor γ. Falls $\text{cost}(v) > \gamma \cdot \text{OPT}(\sigma)$: Dann färbe v schwarz[36]. Schneide den an v hängenden Teilbaum von T ab, definiere Restdauer ρ bis zur Niederlage mit $\rho(v) = 0$.
Propagiere Schwarzfärbung und ρ-Werte der schwarzen Knoten nach oben durch folgende Regeln:

[35]Ein deterministischer Algorithmus ist gegen jeden Gegner gleich gut
[36]schwarz = “Sieg“ vom Gegner

- v Antwortknoten, alle Söhne schwarz
  $\Rightarrow v$ wird schwarz gefärbt, $\rho(v) = \max_{w \text{ Sohn von } v} \rho(w) + 1$
- v Anforderungsknoten, mindestens ein Sohn ist schwarz
  $\Rightarrow v$ wird schwarz gefärbt, $\rho(v) = \min_{w \text{ schwarzer Sohn von } v} \rho(w) + 1$

Klar: $v \in T$ schwarz $\Rightarrow \rho(v)$ endlich.

Beachte: Bisher hängt alles nur von $\mathcal{T}$ und γ ab.

1. Fall: Wurzel von T ist schwarz.
Definiere Anforderungskomponente Q durch: Wähle stets den den schwarzen Sohn mit kleinsten ρ-Wert. Für dieses Q gilt: $\forall$ det. Algorithmen ALG_ω: $\text{ALG}_\omega(\sigma(\text{ALG}_\omega, Q)) > \gamma(\text{OPT}(\sigma(\text{ALG}_\omega, Q)))$ nach spätestens $d_Q = \rho(v)$ vielen Schritten.

2. Fall: Wurzel von T ist nicht schwarz.
Definiere deterministischen Algorithmus ALG_ω durch: Wähle immer einen nicht schwarz gefärbten Sohn. Für dieses ALG_ω gilt:
$\forall Q : \text{ALG}_\omega(\sigma(\text{ALG}_\omega, Q)) \leq \gamma(\text{OPT}(\sigma(\text{ALG}_\omega, Q)))$, denn man kommt nie in ein schwarzes Blatt.

Nach Voraussetzung: Es gibt $\text{ALG} = (\text{ALG}_\omega)_\omega$ randomisiert mit: $\forall Q$: $\text{ALG}_\omega(\sigma(\text{ALG}_\omega, Q)) \leq \gamma\left(E_\omega(\text{OPT}(\sigma(\text{ALG}_\omega, Q)))\right)$
$\Rightarrow$ für jedes Q gibt es ein ALG_ω mit $\text{ALG}_\omega(\sigma(\text{ALG}_\omega, Q)) \leq \gamma(\text{OPT}(\sigma(\text{ALG}_\omega, Q)))$
$\Rightarrow$ Fall 1 tritt nicht ein.
$\Rightarrow$ Fall 2 tritt ein. $\square$

Bemerkung:

- Beweis ist nicht konstruktiv.
- Verallgemeinert Theorem 4.0.4 auf Seite 47 über 2-Personen-Nullsummenspiele mit vollständiger Information.

4.6 Zusammenhang Spiele und Anforderungs-Antwort-Systeme

Sei $\mathcal{T}$ ein endliches Anforderungs-Antwort-System
$\Rightarrow$ Baum T von $\mathcal{T}$ ist endlich. Fasse T als Spielbaum auf.

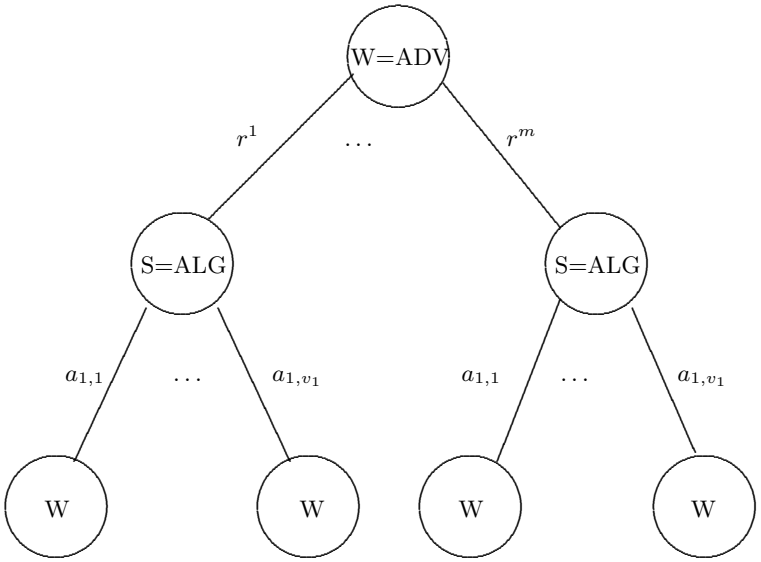


Abbildung 48: Baum T des Anforderungs-Antwort-Systems

Reine Strategie von Schwarz: deterministischer Algorithmus.
Reine Strategie von Weiß: deterministische Anforderungskomponente.
Alle Blätter sind Anforderungsknoten. (Knoten von Weiß)
Bei Beschriftung $r_1, a_1, \ldots, r_n, a_n$ zu Blatt b: Auszahlung in b: $\frac{\text{cost}_n(r_1,\ldots,r_n,a_1,\ldots,a_n)}{\text{OPT}(r_1,\ldots,r_n)}$
$\Rightarrow \mathcal{T}$ist endliches 2-Personen-Nullsummenspiel mit vollständiger Information.
$\Rightarrow$ Spiel ist eindeutig.
$\Rightarrow$ Wert ν des Spiels ist mit reinen Strategien erreichbar.
$\Rightarrow$ Randomisierung hilft dem Gegner ADV nicht. (Theorem 4.4.1 auf Seite 58)
$\Rightarrow$ Hilft auch dem Algorithmus nicht. (Theorem 4.5.3 auf Seite 60)

Fazit: Endliches Anforderungs-Antwort-System gegen adaptive Offline Gegner ist 2-Personen-Nullsummenspiel mit vollständiger Information.

Frage: Geht das auch für adaptive Online Gegner? Ein Problem: Definition der Kostenfunktion: Hier gehen (im Nenner) die Kosten ein, die der ADV online verursacht. Mögliche Abhilfe: Erweiterung von T: Zug von ADV: (r_{i+1}, b_{i+1})
Neues Problem: Für ALG sind die b_i nicht bekannt
$\Rightarrow$ keine vollständige Information.

Vergeßlicher Gegenspieler: Baum beschreibt Situation mit unvollständiger Information für Weiß = ADV.

Aber: Spiel in strategischer Form $\{\text{ALG}_1, \ldots, \text{ALG}_n\}$ mögliche deterministische Algorithmen = reine Strategien von Schwarz.
$\{\sigma_1, \ldots, \sigma_k\}$ möglche Anforderungsfolgen = reine Strategien von Weiß.
Auszahlungsmatrix: $a_{ij} = \frac{\text{OPT}(\sigma_j)}{\text{ALG}_i(\sigma_j)}$ Auszahlung an Schwarz = ALG .

Problem: Randomisierung von ALG : $\sum_i p_i \frac{\text{OPT}(\sigma_j)}{\text{ALG}_i(\sigma_j)} \neq \frac{\text{OPT}(\sigma_j)}{\sum_i p_i \text{ALG}_i(\sigma_j)}$

Abhilfe: Auszahlungsmatrix: $b_{ij} = \frac{\text{ALG}_i(\sigma_j)}{\text{OPT}(\sigma_j)}$ Auszahlung an Weiß = ADV.
Jetzt geht es, solange ADV nicht randomisiert. Braucht er aber auch nicht, da er die Verteilung des ALG kennt.

25.06.03

4.7 Yao's Prinzip

(Konstruktion von unteren Schranken)
Es sei $\mathcal{T}$ ein unbeschränktes Anforderungs-Antwort-System. $\{\sigma_j\}_j$ eine Wahrscheinlichkeitsverteilung über der Menge aller Anforderungsfolgen. Betrachte alle Folgen σ^n = Menge Σ^n, der Länge n. Ist p_j die Wahrscheinlichkeit von σ_j in der Menge aller Folgen dann ist $p_j^n = \frac{p_j}{\sum\limits_{b_k^n \in \Sigma^n} p_k}$ die Wahrscheinlichkeit von σ_j in Σ^n.
$(\text{ALG}_\omega)_\omega$ ein randomisierter Algorithmus für $\mathcal{T}$.

Theorem 4.7.1 (Yao) *Es gelte für eine Zahl c :*

- $\liminf\limits_{n\to\infty} \frac{\inf\limits_\omega \left(E_{j^n}\left(\text{ALG}_\omega(\sigma_j^n)\right)\right)}{E_{j^n}(\text{OPT}(\sigma_j^n))} \geq c$
- $\limsup\limits_{n\to\infty} E_{j^n}(\text{OPT}(\sigma_j^n)) = \infty$

Dann kann kein Algorithmus zur Lösung von $\mathcal{T}$ einen Faktor $< c$ haben, gegen beliebige Gegenspieler.
(In Worten: Die Performance des besten deterministischen Algorithmus ALG $_\omega$ *bei zufälliger Eingabe ist untere Schranke für den kompetitiven Faktor von* ALG *)*

Beweis 4.7.1: Es genügt Behauptung für den schwächsten Gegenspieler - oblivious ADV - zu beweisen. Angenommen Behauptung ist falsch. Dann hat ALG einen kompetitiven Faktor $c' < c$, d.h. es gibt $A > 0$ mit:
$\forall \sigma : E_\omega(\text{ALG}(\sigma)) \leq c' \cdot \text{OPT}(\sigma) + A$
$\Rightarrow \forall n : E_\omega(\text{ALG}(\sigma^n)) \leq c' \cdot \text{OPT}(\sigma^n) + A$
$\Rightarrow \forall n : E_{j^n}\left(E_\omega(\text{ALG}_\omega(\sigma_j^n))\right) \leq c' \cdot \underbrace{E_{j^n}(\text{OPT}(\sigma_j^n))}_{\nearrow\infty \text{ für Folgen } n^t}$
$\Rightarrow E_{j^n}\left(E_\omega(\text{ALG}_\omega(\sigma_j^n))\right) = E_\omega\left(E_{j^n}(\text{ALG}_\omega(\sigma_j^n))\right)$
$\geq \inf_\omega E_{j^n}(\text{ALG}(\sigma_j^n))$

$\Rightarrow \forall t \geq t_0 : \frac{\inf_\omega E_{j^{n_t}}(\text{ALG}(\sigma_j^{n_t}))}{E_{j^{n_t}}(\text{OPT}(\sigma_j^{n_t}))} \leq c' + \underbrace{\overbrace{\frac{A}{E_{j^{n_t}}(\text{OPT}(\sigma_j^{n_t}))}}^{\to 0}}_{\to\infty} \leq c' + \frac{c-c'}{2} < c$ Widerspruch! □

5 Metrische Task-Systeme

(Borodin, Winial, Sahs '87)

5.1 Einführung

Beispiel: Eiscreme-Herstellung

per Maschine:

Sorten	Kosten
Vanille	1
Schokolade	2

per Hand:

Sorten	Kosten
Vanille	2
Schokolade	4

Sortenwechsel in Maschine: 1
Umstellung Maschine-Hand: 2
Darstellung der Situation durch Zustände: (Siehe Abbildung 49 auf der nächsten Seite)

- Man muss Anforderungen (Tasks) erledigen.
- Wie teuer die Erledigung einer Aufgabe r ist, hängt vom aktuellen Zustand ab.
- Zustandswechsel verursacht Kosten.

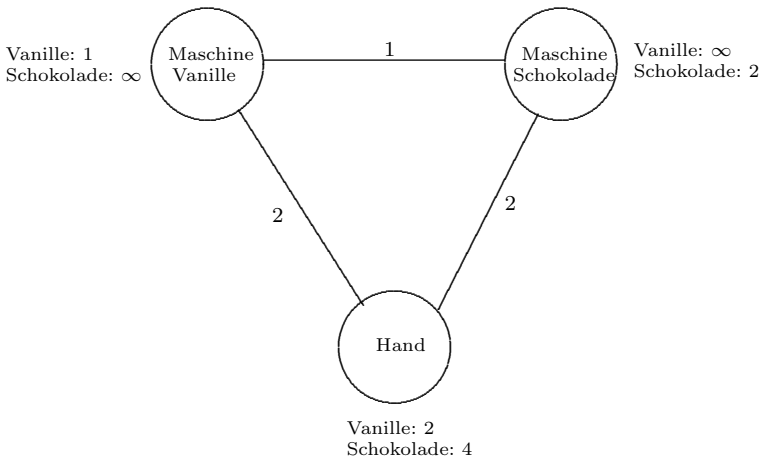


Abbildung 49: Eiscreme Herstellung

Definition 5.1.1 *Metrisches Task-System: Paar* (M, R) *mit* $M = (M, d)$ *Metrischer Raum.*
Metrischer Raum: Grundmenge M; $d : M \times M \to \mathbb{R}_{\geq 0}$ *mit:*

- $d(s,t) = 0 \Leftrightarrow s = t$
- $d(s,t) = d(t,s)$ *Symmetrie.*
- $d(s,t) \leq d(s,p) + d(p,t)$ *Dreiecksungleichung*

$R =$ *Menge von Aufgaben* r *mit*
$r : M \to \mathbb{R}_{\geq 0} \cup \{\infty\}$
für $s \in M : r(s) =$ *Kosten der Erledigung von Aufgabe* r *im Zustand* s

Gegeben eine Aufgabenfolge $\sigma = r_1, r_2, \ldots, r_n$ und Anfangszustand $s_0 \in M$
Müssen Aufgaben r_i in der Reihenfolge von σ bearbeiten, können aber die Zusätnde $s_i \in M$ wählen, in denen wir das tun.
Sei ALG ein Algorithmus für metrische Task-Systeme.
ALG$[i]$ = Zustand $s_i \in M$, in dem ALG Aufgabe r_i bearbeitet.
$s_0 :=$ Anfangszustand

$$\text{ALG}\ (\sigma) = \underbrace{\sum_{i=1}^{n} d(s_{i-1}, s_i)}_{\text{Übergangskosten}} + \underbrace{\sum_{i=1}^{n} r_i(s_i)}_{\text{Bearbeitungskosten}}$$

Weiteres Beispiel: Paging. Cache Größe k, Anzahl aller Datenseiten: $N = \{P_1, \ldots, P_n\}$

$M = \{\text{Mögliche Cache Inhalte}\} = \binom{N}{k}$
Metrik $d : d(s_1, s_2) =$ Anzahl der anzufordernden Seiten, um von s_1 nach s_2 zu gelangen $= k - |s_1 \cap s_2|$
Aufgaben: Seitenanforderungen.
Kosten der Anforderung von Seite $r = r(s) = \begin{cases} 0, r \in s \\ \infty, \text{sonst} \end{cases}$
Hatten früher "Demand Paging" betrachtet, also Algorithmen wie FIFO,LRU,... bei denen eine neue Seite nur dann in den Cache geladen wird, wenn sie gebraucht wird. Bei diesem Modell ist es auch erlaubt Seiten "auf Vorrat" in den Cache zu laden durch entsprechende Zustandsübergänge. Das ändert nichts, wegen:

Lemma 5.1.3 *Sei* ALG *irgendein Paging Algorithmus. Dann es einen Demand Paging-Algorithmus* ALG *', so dass gilt:* $\forall \sigma : \text{ALG}'(\sigma) \leq \text{ALG}(\sigma)$ *(Ü)*

Noch ein Beispiel: Selbstanordnende Listen
Sei ALG ein Listenalgorithmus, der

- Erst kostenpflichtige Vertauschungen ausführt.
- Dann erst auf das verlangte Listenelement zugreift.

Modellierung von ALG als metrisches Task-System $M = \{\text{Permutation } \pi \text{ der l-Listenelemente}\}$
Metrik: $d(\pi_1, \pi_2) =$ kleinste Anzahl Vertauschungen benachbarter Elemente, um von π_1 nach π_2 zu kommen.
Kosten bei Zugriff auf Element $r : r(\pi) =$ Position von r in Permutation π.

5.2 Stetige Task-Systeme

Bisher: Diskrete Sicht: Jede Aufgabe r_i wird in einem einzigen Zustand erledigt.
Jetzt: "Stetige" Sicht: Man darf während der Erledigung einer Aufgabe r den Zustand wechseln.

Modell:

- Zur Erledigung von Aufgabe r_i steht das Zeitintervall $[i, i+1)$ zur Verfügung.
- ALG kann in diesem Zeitintervall beliebig viele (endlich viele) Zustände annehmen.

$$\text{ALG}: \begin{array}{rcl} [i, i+1) & \to & M \\ t & \to & \text{ALG}[t] = \text{der Zustand von ALG im Zeitpunkt } t \end{array}$$

Bearbeitungskosten $r_i : \int_i^{i+1} r_i(\text{ALG}[t])\text{dt} = \sum_s \gamma_s r_i(s)$

Wobei s diejenigen Zustände durchläuft, die im Zeitintervall $[i, i+1)$ angenommen werden und γ_s den Zeitanteil vom Zeitintervall $[i, i+1)$ bezeichnet, für den ALG im Zustand s ist.

Klar: Stetige Algorithmen können nicht kleinere Kosten erzielen als diskrete Algorithmen. Sei ALG stetig und $\sigma = r_1, r_2, \ldots, r_n$ Für jedes r_i sei s_i derjenige im Intervall $[i, i+1)$ angenommene Zustand s, in dem $r_i(s)$ am kleinsten ist.
ALG ' = der diskrete Algorithmus der r_i komplett im Zustand s_i erledigt.
Klar: Bewegungskosten von ALG ' sind $\leq$ denen von ALG .
Bearbeitungskosten ebenso, denn $r_i(s_i)$ = Kosten von ALG ' $= \sum_s \gamma_s \underbrace{r_i(s_i)}_{\text{minimal}} \leq$
$\sum_s \gamma_s \cdot r_i(s)$ = Kosten von ALG .

5.3 Traversierungsalgorithmen

Beispiel: Ski Rental Problem. Anfänger fährt in Winterurlaub. Will an guten Tagen Ski laufen.

Problem: Tag gut: $\begin{array}{ll} \text{SKI mieten} & m \text{ pro Tag} \\ \text{SKI kaufen} & K \text{ einmalig} \end{array}$
Modellierung als Metrisches Task-System (Siehe Abbildung 50)

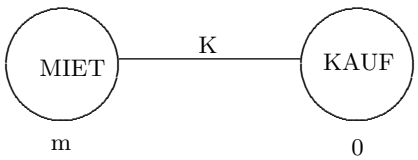


Abbildung 50: Skilaufen

30.06.03 Ohne Einschränkung: $\frac{K}{m} \in \mathbb{N}$

Problem: Gegeben durch Anzahl der Schneetage.

Algorithmen: Kauf am t-ten Schneetag.
Algorithmus RENT: Miete bis akkumulierte Mietkosten = Kaufpreis, dann wird gekauft.
$\Rightarrow$ d.h. $t = \frac{K}{m} + 1$ 1) RENT ist 2-kompetitiv, 2) Besser gehts nicht } Ü

Frage: Lässt Algorithmus RENT sich verallgemeinern? Betrachte zunächst ein Metrisches Task-System mit nur 2 Zuständen, s_1 und s_2. (Siehe Abbildung 51) Ansatz für Algorithmus ALG : starte in s_1 Schleife:

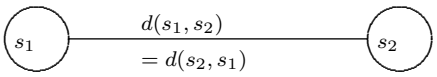


Abbildung 51: Metrisches Task-System mit 2 Zuständen

- Bleibe solange s_1, bis die dort entstandenen Bearbeitungskosten = $d(s_1, s_2)$ sind, dann wechsele nach s_2.
- Bleibe solange s_2, bis die dort entstandenen Bearbeitungskosten = $d(s_2, s_1)$ sind, dann wechsele nach s_1.

Dies ist ein Traversierungsalgorithmus: im Zeichen $s_1 \to s_2 \to s_1$ (setzt Stetigkeit voraus)

Behauptung: Der 2-Traversierungsalgorithmus ist 4-kompetitiv.

Beweis: Pro Zyklus $s_1 \to s_2 \to s_1$ entstehen dem Algorithmus Kosten $4d(s_1, s_2)$ Kosten von OPT ?

1. Fall OPT macht mindestens einen Zustandswechsel $\Rightarrow$ hat mindestens Kosten $d(s_1, s_2)$.
2. Fall OPT ständig in $s_1 \Rightarrow$ OPT erledigt insbesondere diejenigen Aufgaben in s_1, die ALG auch in s_1 erledigt $\Rightarrow$ Kosten $\geq d(s_1, s_2)$ $\square$

Sei G der vollständige Graph über den metrischen Raum. Sei T minimaler Spannbaum von G. Benutze T zur Definition eines Traversierungsalgorithmus ALG .

1. Fall T hat nur Knoten v, bleib in v, löse alle Aufgaben (so gut wie OPT)
2. Fall T hat mindestens eine Kante.

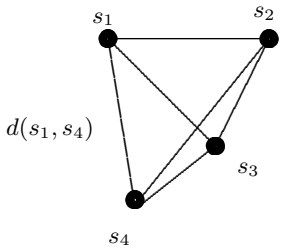


Abbildung 52: Vollständiger Graph

2.1 T hat nur eine Kante (u, v) ALG : $u \to v \to u$.

2.2 Mehr als eine Kante. Sei (u, v) Kante von T mit maximalen Gewicht w, wobei $2^{M-1} < w \leq 2^M$. Entferne (u, v) aus T
$\Rightarrow$ T zerfällt in Teilbäume T_1, T_2 (Siehe Abbildung 53) Seien w_1, w_2

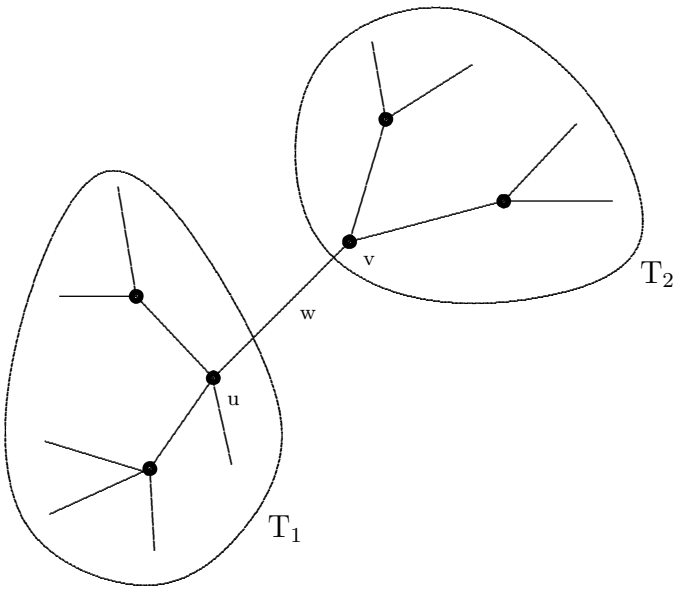


Abbildung 53: T zerfällt in 2 Teilbäume

die Kanten mit Maximalgewicht in T_1 und T_1 und gelte $2^{M_i-1} < w_i \leq 2^{M_i}$ für $i = 1, 2$. Seien ALG $_i$ die rekursiv definierten Algorithmen für T_i.
Definition ALG : Starte in u, führe 2^{M-M_1} Durchläufe von ALG $_1$ in T_1 aus.
Gehe nach v, führe 2^{M-M_2} Durchläufe von ALG $_2$ in T_2 aus.
Gehe wieder nach u.

Theorem 5.3.2 *Traversierungsalgorithmus* ALG *ist* $8(N-1)$*-kompetitiv, wobei* $N = |M|$.

Beweis 5.3.2:

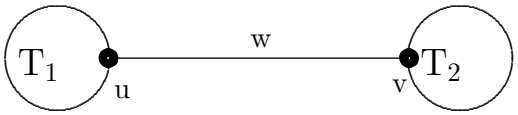
Behauptung 1: ALG hat pro Zyklus Kosten $\leq 4(N-1)2^M$

Behauptung 2: OPT hat pro Zyklus Kosten $\geq 2^{M-1}$
$\Rightarrow$ Theorem

Beweis 1:

Behauptung 3: Jede Kante mit Gewicht v von T, wobei $2^{m-1} < v \leq 2^m$ wird von ALG genau 2^{M-m} mal in jede Richtung durchlaufen.

Beweis 3: Induktion (über Anzahl der Kanten): Aus Abbildung 54 $\Rightarrow$ Be-



mit $2^{M-1} < w \leq 2^M$

Abbildung 54: Baum T

hauptung 3 für Kante (u, v) nach Definition von ALG .
Sei e eine Kante von - z.B. - T_1 mit Gewicht v, wobei $2^{m-1} < v \leq 2^m$ gilt.
$\text{IndVor}(T_1) \overset{\text{Beh. } 3}{\Rightarrow}$ Kante e wird von ALG $_1$ 2^{M_1-m} mal in jede Richtung durchlaufen.
ALG $_1$ wird in ALG 2^{M-M_1} mal ausgeführt. (Definition ALG)
$\Rightarrow$ Kante e wird von ALG $2^{M-M_1} \cdot 2^{M_1-m} = 2^{M-m}$ mal durchlaufen. $\boxed{3}$

Beweis 1: Sei e irgendeine Kante von T mit Gewicht v, $2^{m-1} < v \leq 2^m$
$\overset{\text{Beh. } 3}{\Rightarrow}$ e wird in jede Richtung 2^{M-m} mal von ALG durchlaufen, Kosten pro Durchlauf $= v \leq 2^m$
$\Rightarrow$ Gesamtkosten aller e-Durchläufe: $2 \cdot 2^{M-m} \cdot 2^m = 2 \cdot 2^M$ (Hängt nicht von e ab.)
$\Rightarrow$ Gesamtübergangskosten von ALG $= 2 \sum_{e \in T} 2^M = 2(N-1) \cdot 2^M$
$\Rightarrow$ Gesamtkosten $\leq 2 \cdot$ Gesamtübergangskosten $\leq 4 \cdot (N-1)2^M$ $\boxed{1}$

Beweis 2: Induktion über Kantenanzahl.

1. Fall Auch OPT besucht Knoten von T_1 und T_2, T-Minimum Spanning Tree
$\Rightarrow$ Kosten des Übergangs von T_1 nach T_2 sind mindestens so groß wie Benutzung von (u, v), also $\geq 2^{M-1}$.

2. Fall OPT bleibt in T_1

   2.1 $T_1 = \{u\} \Rightarrow$ Auch ALG besucht u hat Bearbeitungskosten $\geq d(u, v) > 2^{M-1}$
   $\Rightarrow$ diese Kosten hat OPT ebenfalls.

   2.2 T_1 hat Kanten $\overset{\text{i.V.}}{\Rightarrow}$ OPT hat in T_1 Kosten $\geq 2^{M_1-1}$ pro Zyklus von ALG $_1$.
   ALG enthält 2^{M-M_1} Durchläufe von ALG $_1$
   $\Rightarrow$ OPT hat in T_1 insgesamt Kosten $> 2^{M-M_1} \cdot 2^{M_1-1} = 2^{M-1}$ $\boxed{2}$
   $\boxed{\text{Th}}$

Frage: Wie gut ist $8(N-1)$-kompetitiv?

Theorem 5.3.3 *Kein deterministischer Algorithmus für Metrische Task-Systeme kann einen kleineren Faktor als* $2N-1$ *haben,* $N =$ *Anzahl Zustände*

02.07.03

Beweis 5.3.3: Sei ALG ein stetiger Algorithmus für MTS mit N Zuständen. Sei $\epsilon > 0$ beliebig. Gegenspieler "ärgert" ALG mit einer Folge von Aufgaben r_i, $1 \leq i \leq n$, mit
$r_i(s) = \begin{cases} \epsilon & \text{falls } s = \text{ALG}[i-1] \\ 0 & \text{sonst} \end{cases}$ Angenommen ALG wechselt bei Erledigung von $\sigma = r_1, r_2, \ldots, r_n$ k-mal den Zustand, nimmt die Zustände $s_0, s_1, \ldots, s_k$ an.
$\Rightarrow$ ALG $(\sigma_n) = \sum_{i=1}^{k} d(s_{i-1}, s_i) + (n-k)\epsilon$
obere Schranke für OPT (σ_n) ? Technik vom Beweis der unteren Schranke für selbstorganisierende Listen (2.1.5 auf Seite 12)

- Konstruiere eine Famile $\mathfrak{J}$ von $2N-1$ verschiedenen Algorithmen.
- Konstruiere obere Schranke für die Gesamtkosten $\sum_{B \in \mathfrak{J}} B(\sigma_n)$.
- Dann OPT $(\sigma_n) \leq \frac{1}{2N-1} \sum_{B \in \mathfrak{J}} B(\sigma_n) \leq \frac{1}{2N-1} S$

Definition der Familie $\mathfrak{J}$: Soll folgende Invariante erfüllen:
Zu jedem Zeitpunkt

- Ist genau ein $B \in \mathfrak{J}$ in demselben Zustand s wie ALG .
- Sind für jeden anderen $s' \neq s$ genau zwei Algorithmen B'_1, B'_2 im Zustand s'.

(insgesamt $2N-1$ viele Algorihmen)
Gesamtkosten von $\mathfrak{J}$, angesetzt auf σ_n

1. Fall: (tritt n-k mal auf) ALG behält seinen alten Zustand $s = \text{ALG}[i-1]$. Definiere alle $\tilde{B} \in \mathfrak{J}$ bleiben in ihren alten Zuständen. Nur ein B bezahlt (weil er im alten Zustand s von ALG ist) und zwar ϵ

2. Fall: (tritt k mal auf) ALG wechselt den Zustand von s nach s'. Definiere: B'_1 wechselt Zustand von s' nach s, alle andern $\tilde{B} \in \mathfrak{J}$ behalten ihren Zustand
   $\Rightarrow$ Invariante gilt wieder.
   Bewegungskosten: $d(s', s) = d(s, s')$
   Bearbeitungskosten: B und B' zahlen jeder ϵ, weil sie im Zustand s von ALG sind.

Gesamtkosten $\mathfrak{J}(\sigma_n) = \sum_{B \in \mathfrak{J}} B(\sigma_n) \leq (n-k)\epsilon + \sum_{i=1}^{k} d(s_{i-1}, s_i) + 2k\epsilon = \text{ALG}(\sigma_n) + 2k\epsilon$

Hilfsüberlegung: $\text{ALG}\ (\sigma_n) \geq \sum_{i=1}^{k} d(s_{i-1}, s_i) \geq k \cdot \underbrace{\min_{v,w \in M} d(v,w)}_{=1}$[37]

Gesamtkosten: $\mathfrak{J}(\sigma_n) = \sum_{B \in \mathfrak{J}} B(\sigma_n) \leq (n-k)\epsilon + \sum_{i=1}^{k} d(s_{i-1}, s_i) + 2k\epsilon$
$\Rightarrow \text{OPT}(\sigma_n) \leq \frac{(1+2\epsilon)\text{ALG}(\sigma_n)}{2N-1} (= \text{ALG}(\sigma_n) + 2k\epsilon \leq \text{ALG}(\sigma_n)(1+2\epsilon))$
$\Rightarrow \frac{\text{ALG}(\sigma_n)}{\text{OPT}(\sigma_n)} \geq \frac{2N-1}{1+2\epsilon}, \epsilon > 0$ beliebig $\Rightarrow$ Beh.

Situation: $2N-1$ untere Schranke.
Kennen $8(N-1)$-kompetitiven Algorithmus
$\Rightarrow$ lineare Abhängigkeit von N ist korrekt.

[37] Metrischer Raum auf 1 normiert

Frage: Lässt sich Faktor 8 verbessern?
Ja!

Theorem 5.3.4 *Es gibt einen $2N-1$ kompetitiven deterministischen Algorithmus für MTS mit N Zuständen.*

Beweis 5.3.4: Idee: Behalte Verhalten und die Kosten von OPT im Blick!

Realisierung: Arbeitsfunktionen (work functions).

Definition: Sei $\sigma = r_1, \ldots, r_n$ Aufgabenfolge. Für $i \leq n, \sigma_i = r_1, \ldots, r_i; \sigma_0 = \emptyset$
Sei $s \in M$ Dann heißt $w_i(s)$ = minimale Kosten aller möglichen Bearbeitungsfolgen von σ_i die im Anfangszustand s_0 starten und in s enden, die Arbeitsfunktion der Aufgaben σ_i.

Elementare Eigenschaften:

Lemma 5.3.5 *(i)* $\text{OPT}\ (\sigma_i) = \min_{s \in M} w_i(s)$

(ii) $\forall s \in M : w_{i+1}(s) = \min_{x \in M} (w_i(x) + r_{i+1}(x) + d(x,s))$ *(Optimalitätsprinzip: Kann zur Berechnung von* $\text{OPT}\ (\sigma)$ *verwendet werden - Laufzeit? Dynamische Programmierung Ü)*

(iii) $w_i(s) \leq w_{i+1}(s)$ *("weniger Aufgaben verursachen geringere Kosten")*

(iv) $w_{i+1}(s) \leq w_i(s) + r_{i+1}(s)$

Beweis 5.3.5: Trivial.
Definiere jetzt ALG durch Angabe der Zustände $s_0, s_1, s_2, \ldots, s_n$ mit:
ALG löst Aufgabe r_i im Zustand s_i, s_{i+1} = Ein Zustand x^* gemäß:

Lemma 5.3.6 *Für alle i gilt: Es gibt Zustände $x^* \in M$ mit:*

(i) $w_{i+1}(x) + d(s_i, x)$ *nimmt Minimum in x^* an, und*

(ii) $w_{i+1}(x^*) = w_i(x^*) + r_{i+1}(x^*)$ *($\leq$ gilt stets: 5.3.5 (iv))*

Interpretationsversuch: Gehe von s_i in Zustand $s_{i+1} = x^*$, für den gilt:

(i) Die Bewegungskosten $d(s_i, s_{i+1})$ plus Kosten einer optimalen, in s_{i+1} endenden Lösung sind möglichst klein,

(ii) Er eignet sich gut zur Lösung von r_{i+1}.

Beweis 5.3.6: Problem: Müssen zeigen, dass solche Zustände x^* wirklich existieren.
Sei $z \in M$ ein Minimum der Funktion:
$x \to w_{i+1}(x) + d(s_i, x)$
Sei x^* (in Lemma 5.3.5 (ii)) so gewählt, dass
$w_{i+1}(z) = w_i(x^*) + r_{i+1}(x^*) + d(x^*, z)$ (d.h. Minimum für $s = z$ wird in $x = x^*$ angenommen)
$\Rightarrow w_i(x^*) + r_{i+1}(x^*) + d(s_i, x^*) = w_{i+1}(z) + \underbrace{d(s_i, x^*) - d(x^*, z)}_{\leq d(s_i,z) \triangle \neq}$

$\Rightarrow w_{i+1}(x^*) + d(s_i, x^*) \overset{5.3.5 \text{ (iv)}}{\leq} w_i(x^*) + r_{i+1}(x^*) + d(s_i, x^*)$
$\leq w_{i+1}(z) + d(s_i, z) (*)$
$\Rightarrow$ Funktion $x \to w_{i+1}(x) + d(s_i, x)$ nimmt auch in x^* ihr Minimum an und in (*) gilt Gleichheit.
$\Rightarrow w_{i+1}(x^*) = w_i(x^*) + r_{i+1}(x^*)$
Damit gezeigt: x^* hat Eigenschaften (i) und (ii) $\boxed{5.3.6}$

07.07.03

z.z.: Mit dieser Wahl der Zustände s_i ist ALG $2N-1$ kompetitiv. Brauchen dazu obere Schranke für $d(s_i, s_{i+1}) + r_{i+1}(s_{i+1}) =$
$ALG(r_{i+1})$.
$w_{i+1}(s_{i+1}) + d(s_i, s_{i+1}) \overset{5.3.6(i)}{\leq} w_{i+1}(s_i)\,(I)$
$\text{ALG}(r_{i+1}) = d(s_i, s_{i+1}) + \underbrace{r_{i+1}(s_{i+1})}_{5.3.6\,(ii)}$
$= \underbrace{d(s_i, s_{i+1}) + w_{i+1}(s_{i+1})}_{(I)} - w_i(s_{i+1})$
$\overset{(I)}{\leq} \underbrace{w_{i+1}(s_i) - w_i(s_{i+1})}_{(II)}$

Problem: Summe von II teleskopiert nicht!

Trick: Abschätzung so vergröbern, das obere Schranke teleskopiert.

Vergröberung: $w_{i+1}(s_i) - w_i(s_{i+1})$
$\leq w_{i+1}(s_i) - w_i(s_{i+1}) + \underbrace{w_{i+1}(s_i) - w_i(s_i)}_{>0\,5.3.5\,,(iii)}$

$$+\underbrace{w_{i+1}(s_{i+1}) - w_i(s_{i+1})}_{>0\,5.3.5\,,(iii)} + \underbrace{2\sum_{s\neq s_i,s_{i+1}} (w_{i+1}(s) - w_i(s))}_{>0\,5.3.5\,,(iii)}$$

$$= \underbrace{\left(2\sum_{s\neq s_{i+1}} (w_{i+1}(s)) + w_{i+1}(s_{i+1})\right)}_{B_{i+1}} - \underbrace{\left(2\sum_{s\neq s_i} (w_i(s)) + w_i(s_i)\right)}_{B_i}$$

Jetzt: $\text{ALG}(\sigma_i) = \sum_{j=1}^{i} \text{ALG}(r_j)$

$$\leq \sum_{j=1}^{i} (B_j - B_{j-1}) = B_i - \underbrace{B_0}_{\geq 0}$$

$$B_i = 2\underbrace{\sum_{s\neq s_i} (w_i(s))}_{N-1\,\text{Summanden}} + w_i(s_i)$$

$$= (2(N-1)+1)\cdot \underbrace{(\overbrace{\min_{x\in M} w_i(x)}^{\text{OPT}(\sigma_i)} + \overbrace{\max_{(s,s')} d(s,s')}^{\text{Durchmesser}\,\triangle\,\text{von}\,(M,d)})}_{\geq w_i(s')}$$

Damit: $\forall i : \text{ALG}(\sigma_i) \leq (2N-1)\text{OPT}(\sigma_i) + \underbrace{(2N-1)\triangle}_{\text{abhängig von Zustandszahl und Durchmesser von } (M,d)}$

5.3.4

5.4 Randomisierte Algorithmen

Naheliegende Frage: Beim Paging hat uns Randomisierung von k zu $H_k \sim \log_k$ verholfen. Geht das auch hier?

Antworten:

1. Ja, falls (M,d) uniform, d.h. $\forall s \neq s' \in M\, d(s,s') = 1$
2. Ob das für beliebige Metrische Räume mit N Zuständen geht, ist offen.
3. Man kann aber stets $O\left(\frac{(\log N)^6}{\log\log N}\right)$ schaffen.

Theorem 5.4.1 *Es gibt einen randomisierten Algorithmus für uniforme metrische Task-Systeme, der gegen vergeßliche Gegner $2H_n$-kompetitiv ist.*
$\left(H_n = 1 + \frac{1}{2} + \frac{1}{3} + \ldots + \frac{1}{n}\right)$

Beweis 5.4.1: Betrachte stetige Algorithmen, bei denen auch vor Beendigung einer Aufgabe ein Zustandswechsel möglich ist. Definiere Zeitintervalle

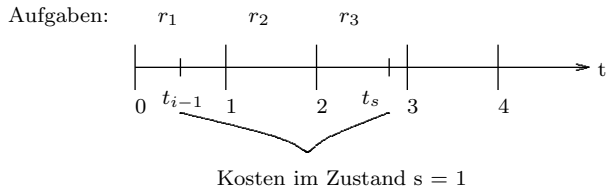


Abbildung 55: Aufteilung in Phasen

$[t_{i-1}, t_i)$ wie folgt: $t_0 = 0$. Wenn t_{i-1} schon definiert ist: Für jeden Zustand $s \in M$ sei t_s derjenige Zeitpunkt mit:
Hätte man alle Aufgaben ab t_{i-1} im Zustand s erledigt, wären Bearbeitungskosten jetzt $= 1$. Setze $t_i = \max_{s \in M} t_s$.
("Grosses t_s = billiger Zustand s")

Beachte:

- t_s ist von t_{i-1} abhängig, man sollte $t_s = t_s^{i-1}$ schreiben.
- Bis jetzt hängen Definitionen von $[t_{i-1}, t_i)$ nur von MTS und der Aufgabenfolge $r_1, r_2, \ldots$ ab.

Definiere jetzt randomisierten Algorithmus ALG durch sein Verhalten in Phase $[t_{i-1}, t_i)$ wie folgt:
$t = t_{i-1}$
Solange es noch $s \in M$ gilt mit $t < t_s$.
Wähle zufällig ein solches s aus und gehe nach s;
$t_i = t$
Beweis der Kompetivität von ALG :

Bemerkung: Nur im ersten Zustand einer Phase entstehen Bearbeitungskosten $= 1$, sonst weniger, da Bearbeitung ja erst nach t_{i-1} beginnt.
$\Rightarrow$ hat in jedem angenommen Zustand Bearbeitungskosten ≤ 1, Übergangskosten pro Zustandswechsel $= 1$
$\Rightarrow$ Gesamtkosten von ALG pro Phase $\leq 2 \cdot$ Anzahl der Zustandswechsel. Erwartungswert?
Sei X_n = Anzahl der Zustandswechsel bei N Zuständen. Betrachte die sortierten Werte $t_s, s \in M$ Angenommen, der erste in Phase i gewählte Zustand ist derjenige mit dem k-kleinsten t_s Wert.

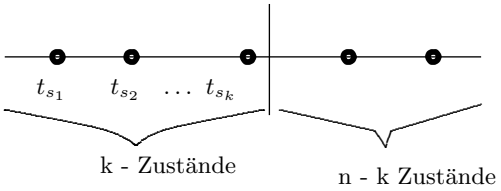


Abbildung 56: Phasen bei ALG

$X_n = 1 + X_{N-k}$ Jede Wahl des ersten Zustands ist gleich wahrscheinlich.
$\Rightarrow E(X_n) = \frac{1}{N} \sum_{k=1}^{N} (1 + E(X_{n-k})$
$= 1 + \frac{1}{N} \sum_{j=0}^{N-1} E(X_j)$ **Trick:** Historie loswerden
$N \cdot E(X_n) - (N-1)E(X_{n-1}) = N + \sum_{j=0}^{N-1} E(X_j) - (N-1) \sum_{j=0}^{N-2} E(X_j)$
$= 1 + E(X_{n-1})$
$\Rightarrow N(E(X_n) - E(X_{n-1})) = 1$
$\Rightarrow E(X_n) = \frac{1}{N} + E(X_{n-1}) \Rightarrow E(X_n) = H_n$
$\Rightarrow$ Pro Phase ALG hat erwartete Kosten $\leq 2 \cdot H_n$, OPT hat pro Phase Kosten ≥ 1 $\square$
Zu (3) (Randomisierung in allgemeinen metrischen Räumen)

Idee: Bette gegebenen metrischen Raum (M, d) in einen andern (M, d') so ein, dass gilt:
$\forall s, t \in M : d(s,t) \leq D \cdot d'(s,t)$
Falls ein c-kompetitiver ALG für (M, d') existiert, so hat man $c \cdot D$-kompetitiven Algorithmus in (M, d).

6 Das k-Server-Problem

Beispiel: ADAC Pannenhelfer (Siehe Abbildung 57 auf der nächsten Seite)
Minimiere die Gesamtfahrtkosten? Tuts Greedy?

09.07.03

6.1 Grundlagen

Gegeben: $M = (M, d)$, metrischer Raum. k Server, die an Punkten von M stehen. Anforderung im Punkt r wird dadurch bedient, dass einer der k-

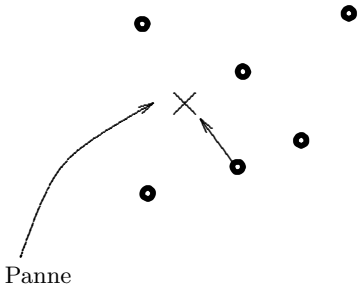


Abbildung 57: ADAC Pannenhelfer

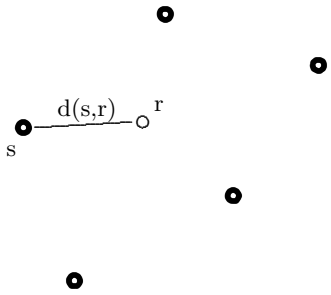


Abbildung 58: Anforderungen beim k-Server-Modell

Server nach r geht. Falls dieser vorher in s war: (Übergangs-)kosten $d(s, r)$. (Siehe Abbildung 58)
$\sigma = r_1, r_2, \ldots, r_n$ mit $r_i \in M$ Anforderungsfolge.
Beim Eintreffen von Anforderung r_i :

- Falls schon ein Server in r_i steht: NOP.
- Sonst entscheide, welcher der k-Server nach r_i bewegt wird.

Ziel: Gesamtkosten (nur Übergangskosten) zu minimieren.

Lemma 6.1.1 *Man darf faul sein und Server nur dann an einen neuen Ort bewegen, wenn dort Anforderung vorliegt.*

Beweis 6.1.1: Sei ALG ein k-Server, der "vorwegeilend" Server bewegt. Konstruiere einen faulen ALG ' folgendermaßen:
ALG ' merkt sich für jeden der k Server alle Züge, die ALG mit diesem Server macht in der richtigen Reihenfolge. Er führt diese Züge (bzw. nur den Zug

an den letzten Standort) erst aus, wenn dort tatsächlich eine Anforderung ist.
$\Rightarrow \forall\sigma : \text{ALG}'(\sigma) \leq \text{ALG}(\sigma)$
Ähnlichkeit zu Lemma 5.1.3 auf Seite 66 (Demand Paging) nicht zufällig, denn:
Paging ist ein Spezialfall des k-Server-Problems. k Server $\Leftrightarrow$ k Slots im Cache.
Punkte in $M \Leftrightarrow$ insgesamt vorhandene Anfrageseiten.
Server i bewegt sich von s nach $t \Leftrightarrow$ In Slot i wird Seite s durch Seite t ersetzt.
$d \Leftrightarrow$ Einheitsmaß.

Zweite Anwendung: Festplatte mit k Köpfen (Siehe Abbildung 59) ist k-Server-Problem auf einer Geraden.

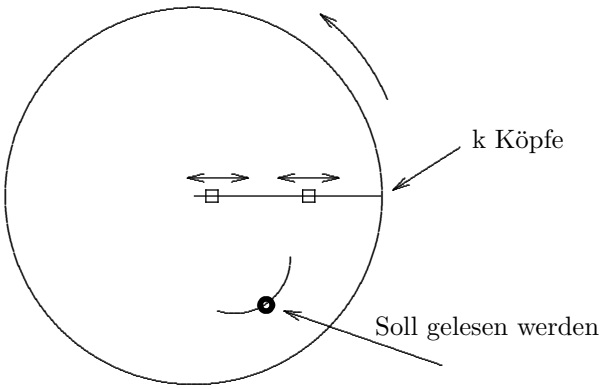


Abbildung 59: Festplatte mit k Köpfen

Lemma 6.1.2 *Das k-Server-Problem endlicher metrischer Räume ist ein Spezialfall metrischer Task-Systeme.*

Beweis 6.1.2: Sei (M, d) der metrische Raum in dem die k-Server leben.
Konstruiere $(\tilde{M}, \tilde{d})$ für ein metrisches Task-System folgendermassen:
$\tilde{M} = \binom{M}{k} = \{S \subset M, |S| = k\}$ mögliche Serverpositionen.
Für Teilmengen S, T von M mit je k Elementen: $\tilde{d}(S, T)$ = Gewicht eines minimalen Matching von S und T. (Siehe Abbildung 60 auf der nächsten Seite) = Überführungskosten der k-Server von S nach T. Aufgabenfunktion:
Für $p \in M$ (Aufgabe):

$$p(S) = \begin{cases} 0 & , p \in S \\ \infty & , \text{sonst} \end{cases}$$

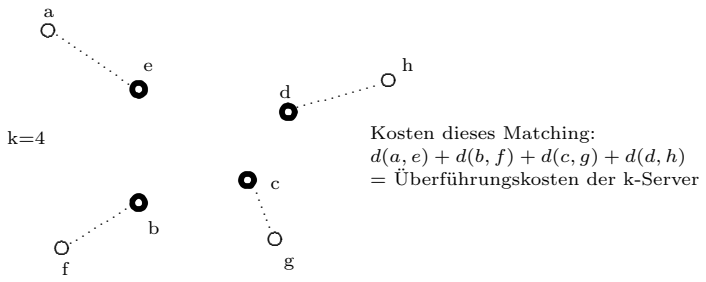


Abbildung 60: Minimales Matching

Folgerung: Aus Lemma 6.1.2 auf der vorherigen Seite und Theorem 5.3.4 auf Seite 73 folgt: Das k-Server-Problem in einem beliebigen metrischen Raum mit $m = |M|$ Punkten lässt sich $\left(2\binom{m}{k} - 1\right)$-kompetitiv lösen. Frage: Geht das besser?

Lemma 6.1.3 *Schon für 2 Server auf euklidischen Liniensegment ist der Greedy-Algorithmus nicht kompetitiv.*

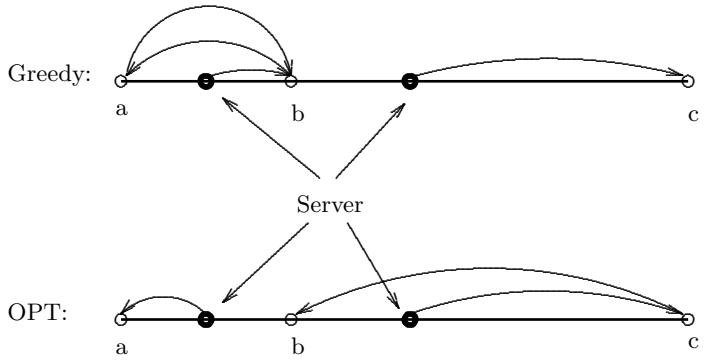


Abbildung 61: Verhalten von Greedy und OPT

Beweis 6.1.3: Aufgabenfolge $\sigma = c(ba)^m$
Kosten der Algorithmen: (Siehe Abbildung 61) Greedy$(\sigma) \geq 2(m+1)d(a,b)$
OPT $(\sigma) \leq d(a,c) + d(b,c)$ □

6.2 Untere Schranke

Betrache Modell, in dem ALG über k Server verfügt, OPT nur über $h \leq k$ Server verfügt. (wie beim Paging).

Theorem 6.2.1 *Sei $h \leq k$ und (M, d) ein beliebiger metrischer Raum mit $|M| \geq k+1$- Dann gibt es keinen Algorithmus für das (k, h)-Server Problem mit kompetitiven Faktor $c < \frac{k}{k-h+1}$*

Beweis 6.2.1: Sei ALG ein fauler k-Server Algorithmus: Sei M' Teilraum von M mit genau $k+1$ Punkten.
$\Rightarrow$ Ein Punkt ist immer unbesetzt.
Der böse Gegner generiert die nächste Anforderung stets am unbesetzten Punkt.
$\rightarrow$ Anforderungsfolge $\sigma = r_1, r_2, r_3, \ldots, r_n$
ALG bedient r_i mit dem Server, jetzt noch in r_{i+1} steht (denn r_{i+1} ist der nächste freie Punkt)
$\Rightarrow$ Kosten ALG $(r_i) = d(r_{i+1}, r_i) = d(r_i, r_{i+1})$
$\text{ALG}(\sigma) = \sum_{i=1}^{n-1} d(r_i, r_{i+1})$
$\text{OPT}(\sigma) \leq$? Alter Trick: Mittelwertbildung: r_1 = der Anfangs unbesetzte Punkt von M'. Betrachte alle h-elementigen Teilmengen S von M', die r_1 enthalten: $\binom{k}{h-1}$ viele. Für jedes dieser S definiere einen k-Server-Algorithmus B_S folgendermaßen:

- Zu Beginn sind alle Server in den Punkten von S ($\Rightarrow r_1$ ist abgedeckt)
- bei Anforderung r_i:
  - Falls bei r_i einer der h-Server steht: NOP.
  - Sonst ziehe den Server r_{i-1} (dort steht einer) nach r_i

Behauptung 1: Sei $S \neq T$. Dann belegen B_S und B_T zu jedem Zeitpunkt unterschiedliche Mengen mit ihren Servern.

Beweis 1: Seien S^i, T^i die von B_S und B_T nach Bearbeitung von r_i belegten Serverpositionen. Zeige: $\forall i : S^i \neq T^i$.
$i = 0 : S^i = S \neq T = T^i \surd$
Gelte $S^{i-1} \neq T^{i-1}$

1. Fall: $r_i \in S^{i-1} \cap T^{i-1}$ Beide Algorithmen B_S und B_T lassen ihre Server stehen
   $\Rightarrow S^i = S^{i-1} \neq T^{s-i} = T^i$

2. Fall: $r_i \in S^{i-1}, \notin T^{i-1} \Rightarrow B_S$ lässt Server in r_{i-1} stehen $\Rightarrow r_{i-1} \in S^i$
   B_T zieht von r_{i-1} nach $r_i \Rightarrow r_{i-1} \notin T^i$

3. Fall: $r_i \notin S^{i-1}, \notin T^{i-1}$
$\Rightarrow S^{i-1} \neq T^{i-1} \Rightarrow S^{i-1} - \{r_{i-1}\} \neq T^{i-1} - \{r_{i-1}\}$
$\Rightarrow \underbrace{S^{i-1} - \{r_{i-1}\} + \{r_i\}}_{S^i} \neq \underbrace{T^{i-1} - \{r_{i-1}\} + \{r_i\}}_{T^i}$

$\boxed{1}$
Gesamtkosten aller B_S bei Bearbeitung von σ:

- Aus Behauptung 1 $\Rightarrow$ nach jeder Anforderung r_{i-1} sind genau $\binom{k}{h-1}$ viele verschiedene h-elementigen Teilmengen von M' durch B_S belegt.
- Jedes S^{i-1} enthält r_{i-1} (nach Def. der B_S).
- Genau diejenigen B_S müssen bei Anforderung r_i bezahlten.
  - und zwar $d(r_{i-1}, r_i)$ deren Menge S^{i-1} den Punkt r_i nicht enthält.
- Das sind $\binom{k-1}{h-1}$ viele der insgesamt $\binom{k}{h-1}$ Algorithmen.

$\Rightarrow$ Gesamtkosten der Bearbeitung von r_i : $\binom{k-1}{h-1} d(r_{i-1}, r_i)$

$\Rightarrow$ Gesamtkosten der Bearbeitung von von σ : $\binom{k-1}{h-1} \cdot \underbrace{\sum_{i=1}^{n} d(r_{i-1}, r_i)}_{=\mathrm{ALG}(\sigma)} (*)$

Jetzt alter Trick: $\mathrm{OPT}(\sigma) \leq \min_S B_S(\sigma)$
$\leq \frac{1}{\binom{k}{h-1}} \sum_S B_S(\sigma)$
$\overset{*}{=} \frac{\binom{k-1}{h-1}}{\binom{k}{h-1}} \mathrm{ALG}(\sigma) = \frac{(k-1)!\cdot(h-1)!\cdot(k-h+1)!}{(h-1)!\cdot(k-h)!\cdot k!} \mathrm{ALG}(\sigma) = \frac{k-h+1}{k} \mathrm{ALG}(\sigma)$
$\Rightarrow \frac{\mathrm{ALG}(\sigma)}{\mathrm{OPT}(\sigma)} \geq \frac{k}{k-h+1}$ □

Korollar 6.2.2 *(h,k)-Paging hat die Komplexität $\frac{k}{k-h+1}$ (nur gezeigt für $h = k$)*

Für $h = k$ sagt Theorem 6.2.1 auf der vorherigen Seite: k ist untere Schranke für k-Server Problem.

Frage: Gibt es allgemeine k-kompetitive Algorithmen? Offen!

14.07.03 Wollen jetzt zunächst Spezialfälle betrachten.

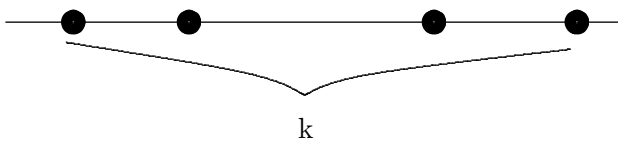


Abbildung 62: k Server auf euklidischer Gerade

6.3 k Server auf Bäumen

Zunächst: k-Server auf euklidischer Gerade. (Siehe Abbildung 62) Schon gesehen: Greedy tuts nicht (Siehe Abbildung 61 auf Seite 80)

Besser: Double Coverage (DC)

1. Fall: Anforderungspunkt a liegt "ausserhalb", gehe mit nächst gelegenem Server zu a. (Siehe Abbildung 63)

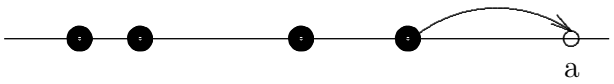


Abbildung 63: Anforderungspunkt a ausserhalb

2. Fall: Anforderungspunkt a liegt "innerhalb", bewege die beiden zu a benachbarten Server gleichschnell auf a zu, bis der erste dort ankommt. (Siehe Abbildung 64)

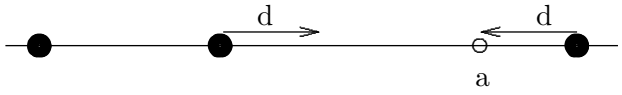


Abbildung 64: Anforderungspunkt a innerhalb

Klar:

- DC ist nicht lazy. (lässt sich lazyfizieren, ohne Kosten zu erhöhen)
- Kann vorkommen, dass mehrere Server am gleichen Ort stehen.

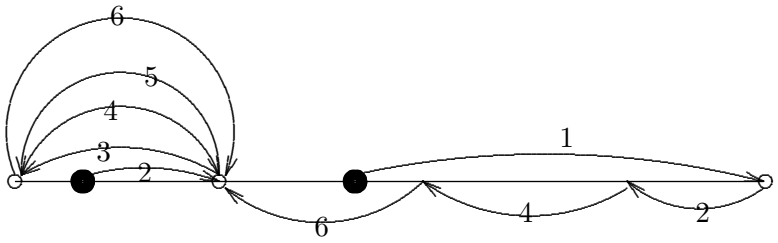


Abbildung 65: DC bei Greedys Albtraum

Beispiel: (Greedy's Albtraum) (Siehe Abbildung 65) Nach endlich vielen Schritten (abhängig von $\frac{d(b,c)}{d(a,b)}$) endet der rechte Server in b

Theorem 6.3.1 DC *ist k-kompetitiv, also optimal*[38]

Lemma 6.3.2 (event-gesteuerte Online-Analyse) *Ein deterministischer* ALG *spielt gegen* OPT *. Den aktuellen Zuständen von* ALG *und* OPT *seien durch:*

$$\Phi : Z_{\text{ALG}} \times Z_{\text{OPT}} \to \mathbb{R}$$

Werte zugeordnet (="Potential")
(z.B. Anzahl der Inversionen beim Listenproblem)
ALG *und* OPT *machen (vielleicht abwechselnd) Züge, die mit Kosten* ≥ 0 *verbunden sind.*
Sei $\sigma = e_1, \ldots, e_n$ *die gesamte Zugfolge. Es gelte:*

(i) e_i *Zug von* ALG *: Potential fällt mindestens um* $\text{ALG}(e_i)$*, d.h.* $\text{ALG}(e_i) \leq \Phi_{i-1} - \Phi_i = \Delta\Phi$

(ii) e_i *Zug von* OPT *: Potential steigt höchstens um* $c \cdot \text{OPT}(e_i)$*, d.h.* $\text{OPT}(e_i) \geq \frac{\Phi_i - \Phi_{i-1}}{c}$ *oder* $\Delta\Phi \leq c \cdot \text{OPT}(e_i)$

(iii) $\Phi_i \geq U$ *für Konstante* U *und alle* i*.*

Dann ist ALG *c-Kompetitiv.*

Beweis 6.3.2: Sei $A \subseteq \{1, 2, \ldots m\}$ die Menge der Indizes der Züge von ALG . Nach Voraussetzung $\text{ALG}(\sigma) \leq \sum_{i \in A} (\Phi_{i-1} - \Phi_i)$ und
$c \cdot \text{OPT}(\sigma) \geq \sum_{i \in A^c} (\Phi_i - \Phi_{i-1})$.

[38] Theorem 6.2.1 auf Seite 81 gilt für beliebige metrische Räume mit mindestens $k+1$ Punkten, also auch für Geraden

Genügt zu zeigen: $\exists$ Konstante V mit: $\forall n, \forall \sigma$ der Länge n:

$\text{ALG}(\sigma) \leq \sum_{i \in A} (\Phi_{i-1} - \Phi_i) \overset{!}{\leq} \sum_{i \in A^c} (\Phi_i - \Phi_{i-1}) + V \leq c \cdot \text{OPT}(\sigma) + V$

$\Leftrightarrow \sum_{i \in A} \Phi_{i-1} + \sum_{i \in A^c} \Phi_{i-1} \overset{!}{\leq} \sum_{i \in A^c} \Phi_i + \sum_{i \in A} \Phi_i + V$

$\Leftrightarrow \Phi_0 \overset{!}{\leq} \Phi_n + V$ (Φ_0 konstant und nach (iii) $U \leq \Phi_n$)

$\Rightarrow \Phi_0 \leq \Phi_n + \underbrace{(\Phi_0 - U)}_{=V}$ 6.3.2 Wollen jetzt Lemma 6.3.2 zum Beweis der k-Kompetivität von DC verwenden. Dazu Definition einer geeigneten Potentialfunktion Φ :

$\Phi : k \cdot M_{\min} + \sum_{\text{DC}}$, wobei:

$M_{\min}$ = Kosten eines minimalen Matchings zwischen den Servern von DC und OPT

$\sum_{\text{DC}}$ = Summe aller absoluten Abstände zwischen allen Servern von DC .

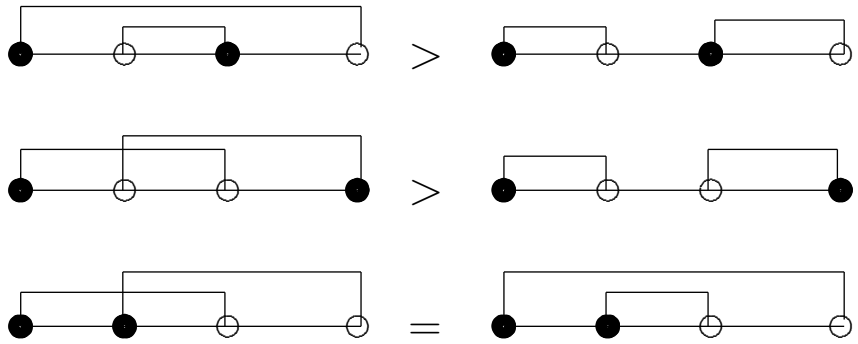


Abbildung 66: Matching “Gleichungen“

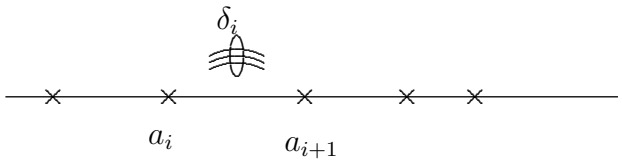


Abbildung 67: Brücken beim Matching

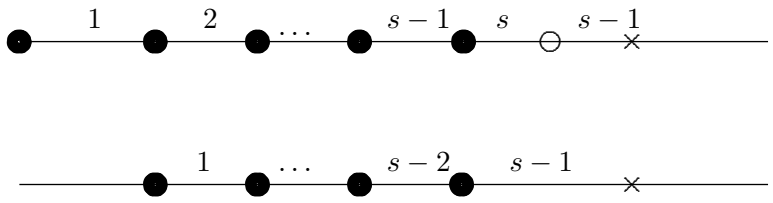


Abbildung 68: Konstruktion eines minimalen Matchings

Einschub: Minimales Matching auf Gerade: Klar sind folgende Gleichungen: (Siehe Abbildung 66 auf der vorherigen Seite) Es muss über $[a_i, a_{i+1}]$ mindestens δ_i viele "Brücken" geben. (Siehe Abbildung 67 auf der vorherigen Seite) $\delta_i = |\#\text{Schwarz} \leq a_i - \#\text{Weiß} \leq a_i|$

$\Rightarrow$ Kosten eines beliebigen Matching $\geq \sum_{i=1}^{2n-1} (\delta_i \cdot (a_{i+1} - a_i))$

Behauptung: Die untere Schranke kann (durch minimales Matching) erreicht werden.

Beweis: Per Induktion über n. $n = 1 \surd$

$n > 1$ Betrache den linksten Punkt, oBdA ist er schwarz. Matche diesen schwarzen Punkt auf den linksten Weißen (Siehe Abbildung 68) und entferne die Punkte. $\surd$

Müssen zeigen, dass Φ die Eigenschaften (i) bis (iii) von Lemma 6.3.2 erfülllt.

(iii): $\Phi > 0$

(ii): OPT zieht immer vor DC und OPT ist lazy. Angenommen OPT zieht einen Server um Abstand d zu neuen Anfrageort.

$\Rightarrow \sum_{\text{DC}}$ ändert sich nicht, $M_{\min}$ wächst um höchstens d

$\Rightarrow \Delta\Phi \leq k \cdot d$

(i)

1. Fall DC bewegt nur einen (und zwar den äußersten) Server. (Siehe Abbildung 69) $\sum_{\text{DC}}$ wächst um $(k-1) \cdot d$, $M_{\min}$ fällt um mindestens d

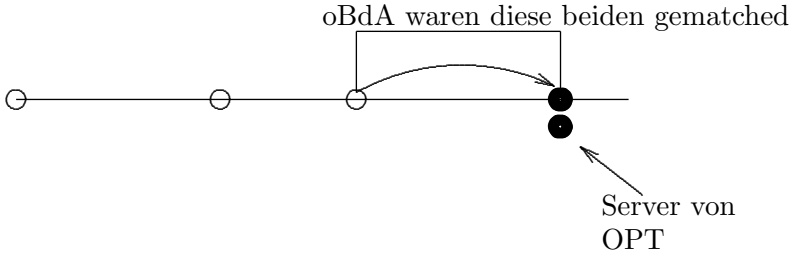


Abbildung 69: DC zieht nur einen Server

$\Rightarrow \Delta\Phi \leq -d$

2. Fall DC bewegt 2 Server um jeweils Abstand d. (Siehe Abbildung 70) In

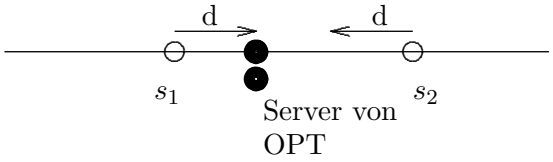


Abbildung 70: DC zieht 2 Server

$\sum_{\text{DC}} : \forall s \neq s_1, s_2 : |s_1 - s| + |s_2 - s|$ bleibt konstant, $|s_1 - s_2|$ sinkt um $2d$.
Sei $M_{\min}$ das alte minimale Matching. s_1 und s_2 können nicht beide zu Partnern ausserhalb gematched sein (Siehe Abbildung 71).

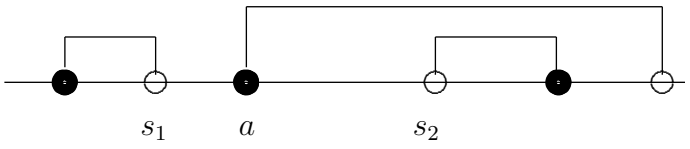


Abbildung 71: Nicht minimales Matching

$\Rightarrow$ einer von beiden muss vorher auf a gematched gewesen sein, dieser kommt näher, aber der andere entfernt sich von seinem alten Partner $\Rightarrow$ $M_{\min}$ wächst nicht $\Rightarrow$ Φ fällt mindestens um $2d$ = Kosten ALG . $\square$.

16.07.03

Verallgemeinerung: T Baum. DC-Tree: Bewege alle zu a unmittelbar benachbarten Server gleichschnell auf a zu, bis der erste dort ankommt. (Siehe Abbildung 72 auf der nächsten Seite).

- DC-TREE kann mehere Server gleichzeitig ziehen.
- Anzahl kann sich zur Laufzeit ändern.

Klar: DC-TREE ist Verallgemeinerung von DC . (Siehe Abbildung 73 auf der nächsten Seite)

Theorem 6.3.5 DC-TREE *ist k-kompetitiv. (Optimal)*

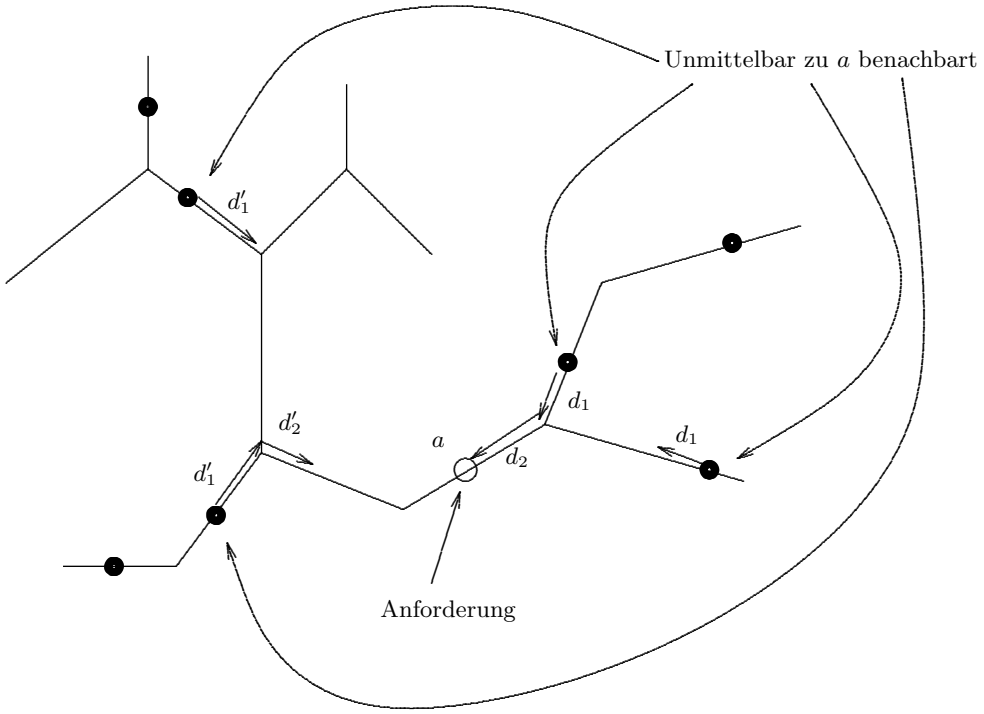


Abbildung 72: Baum T mit Servern

Abbildung 73: DC-TREE auf Gerade

Beweis 6.3.5: Ähnlich wie bei DC , mit Lemma 6.3.2 auf Seite 84. Potentialfunktion: $\Phi = k \cdot M_{\min} + \sum_{\text{DC-TREE}}$, wobei $M_{\min}$ = Kosten eines minimalen Matchings der DC-TREE Server auf die OPT Server ist und $\sum_{\text{DC-TREE}} = \sum_{s_i, s_j \text{ Server von DC-TREE}, i<j} |s_i - s_j|$ (Entlang des Baums)
Betrachte Zug von OPT , lazy: 1 Server wird um Abstand d bewegt:
$\Rightarrow \sum_{\text{DC-TREE}}$ bleibt fest,
$M_{\min}$ kann höchstens um d wachsen
$\Rightarrow \Delta\Phi \leq k \cdot d$.
Betrachte Zug von DC-TREE , zerlege ihn in Phasen, bei denen die Anzahl der sich bewegenden Server konstant bleibt. Behandle jede Phase einzeln. Angenommen in der akutellen Phase sind m der k Server unterwegs, alle

direkt zu a benachbart. Nicht alleder m Server können im alten minimalen Matching einen OPT -Partner "ausserhalb" haben. Gleiches Argument wie bei DC . (Siehe Abbildung 74)
⇒ Einer derunmittelbaren Nachbarn war vor dem Zug von DC-TREE auf

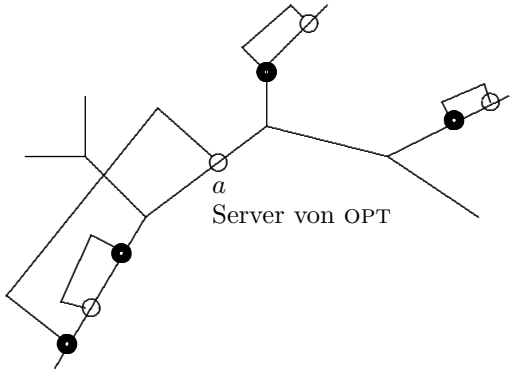


Abbildung 74: Nicht minimales Matching auf Baum

den OPT Server in a abgebildet. (OPT zieht vor DC-TREE)
⇒ Beitragdieses Servers nimmtum d ab, alle übrigen $m-1$ sich bewegenden Server können ihren Beitrag zu $M_{\min}$ höchstens um d vergrößeren. $M_{\min}$ wächst höchstens um $(m-2)d$. Was macht $\sum_{\text{DC-TREE}}$? Siehe Abbildung 75.
Sei s' ein nicht unmittelbar zu a benachbarter Server (der sich demnach in

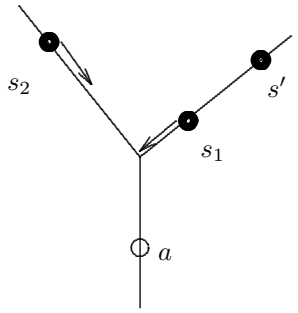


Abbildung 75: Verhalten von $\sum_{\text{DC-TREE}}$

dieser Phase nicht bewegt). Wie verhält sich s' zu den bewegenden Servern?
Einer entfernt sich von s'um d, die übrigen $m-1$ kommen s' um d näher
⇒ Änderungsbeitrag zu $\sum_{\text{DC-TREE}}$: $(k-m)(d-(m-d)d)$.
Netrache nun zwei sich bewegende Server, s_1 und s_2: Abstand fällt um $2d$
⇒ Änderungsbeitragzu $\sum_{\text{DC-TREE}}$: $\binom{m}{2}(-2d)$

$\Rightarrow$ Änderung von Φ pro Phase (eines Zuges von von DC-TREE)
$\leq k \underbrace{(m-2)d}_{\Delta M_{\min}} - \underbrace{(k-m)(m-2)d - (m-1)md}_{\Delta \sum_{\text{DC-TREE}}}$
$= kmd - 2kd - kmd + 2kd + m^2d - 2md - m^2d + md = -md = -$DC-TREE (Phase) $\square$.

Anwendungen von dc-tree :

1. k Server auf Graph G.

   Ansatz: Sei T ein minimaler Spannbaum von G. (Siehe Abbildung 76) Wende Algorithmus DC-TREE auf T an. Gerade gezeigt: DC-TREE ist k-kompetitiv, d.h DC-TREE $\leq k \cdot$ OPT-T, aber OPT-G könnte auch solche Kanten von G benutzen, die nicht in T vorkommen. Sei e eine Kante

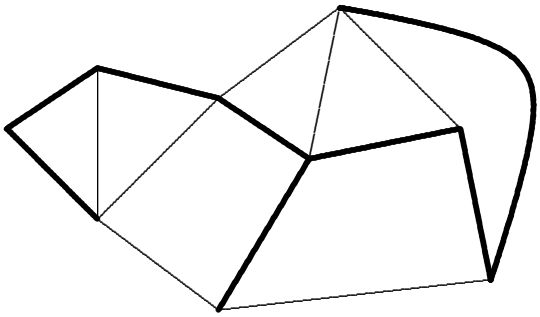
   

   Abbildung 76: Minimaler Spannbaum

   $\in G-T$. Pfad in T von v nach w: Siehe Abbildung 77 auf der nächsten Seite.
   $\Rightarrow$ Pfad hat Länge $\leq (N-1)$Länge von e.
   $\Rightarrow$ Kosten von OPT-T$\leq (N-1)$ Kosten von OPT-G
   $\Rightarrow$ DC-TREE angwenedet auf T von G ist $(N-1)k$-kompetitiv. (Gilt nur für Anforderungen auf Knoten, wegen Abbildung 78 auf der nächsten Seite). Besser als $2\binom{N}{k} - 1$ aus Reduktion auf metrische Tasksystem (Theorem 5.3.4 auf Seite 73).

   Trotzdem: $(N-1)k$ ist schwach!

2. Paging: (Abbildung 79 auf Seite 92) Ein Server geht von P_i nach $P_j \Leftrightarrow P_i$ wird im Cache durch P_j verdrängt. Jeder k Server ALG für T ist ein Paging Algorithmus. DC-TREE entspricht Flush-When-Full.

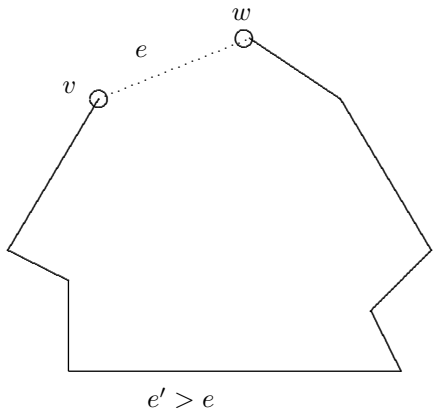


Abbildung 77: Unmöglicher pfad von v nach w in T

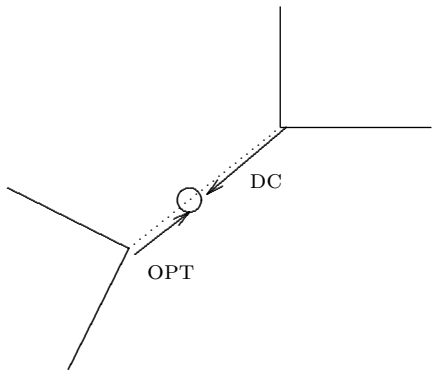


Abbildung 78: Problematische Anforderungen

6.4 2 Server im $\mathbb{R}^d$

, $d = 2$, euklidische Metrik. Definiere:
$\text{slack}(x, y, r) = |xy| + |xr| - |ry| \geq 0 (\triangle \neq)$ (Siehe Abbildung 80 auf Seite 93)
Algorithmus Slack-Coverage (SC). Falls die Server in x und y stehen und Anforderung bei r kommt, $r \notin \{x, y\}$ Sei $|xr| \leq |yr|$:

- Bewege Server in y um $\frac{1}{2}\text{slack}(x, y, r)$ auf x zu.
- Ziehe Server in x nach r

(Siehe Abildung 81 auf Seite 94) Angenommen $d = 1$ (2 Server auf Gerade)

1. Fall a liegt "ausserhalb" $\frac{1}{2}\text{slack}(x, y, r) = \frac{1}{2}(|xy| + |xr| - |yr|) = 0$ (Abbildung 82 auf Seite 94)

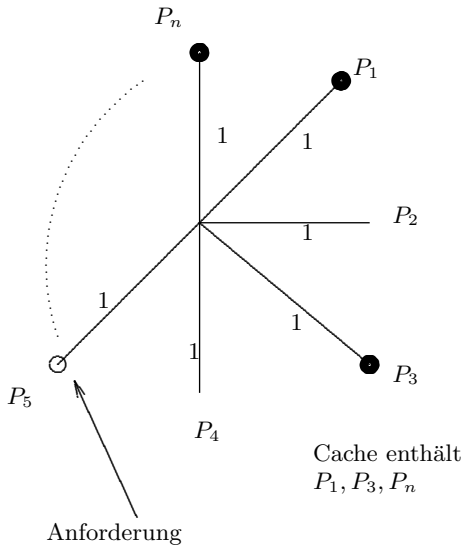


Abbildung 79: Paging Graph

2. Fall a liegt "innerhalb" $\frac{1}{2}\text{slack}(x,y,r) = \frac{1}{2}\left(|xy| + |xr| - |yr|\right) = |xr|$ (Abbildung 83 auf Seite 95)

$\Rightarrow$ SC auf Gerade = DC .

Lemma 6.4.1 *Sei y' der neue Standort des Server, der vorher in y stand. Dann ist $|y'r| \leq |yr|$*

Beweis 6.4.1: $\frac{1}{2}\text{slack}(x,y,r) = \frac{1}{2}(|xy| + \underbrace{|xr| - |yr|}_{\leq 0(|xr| \leq |yr)}) \leq \frac{1}{2}|xy|$
$\Rightarrow y'$ liegt auf derselben Seite des Bisektors $B(x,y) = \{z \in \mathbb{R}^d, |zx| = |zy|\}$ wie y, aber r liegt auf der anderen Seite.
$\Rightarrow |y'r| \leq |yr|$ (benutzt Eigenschaften des Euklidischen Raums).

Theorem 6.4.2 SC *ist 3-kompetitiv.*

Beweis 6.4.2:

- Verwendet Lemma 6.3.2 auf Seite 84.
- Potentialfunktion $\Phi = 3 \cdot M_{\min} + |xy|$

21.07.03

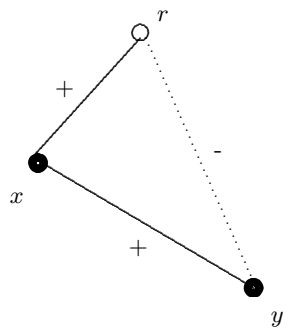


Abbildung 80: slack(x, y, r)

6.5 Ein $2k-1$ kompetitiver Algorithmus für metrische Räume

Theorem 6.5.1 (Borodin, Linial, Saks, Kontsupicas, Papadimitrion '95) *Es gibt einen $(2k-1)$-kompetitiven k-Server Algorithmus für beliebige metrische Räume.*

Beweis 6.5.1: Idee: Arbeitsfunktionen.

- Wie bei metrischen Task-System.
- Einfacher, da keine Aufgaben berechnet werden.

Situation: Anfangskonfiguration der Server: C_0 (k-Multimenge), Anforderungsfolge $\sigma = r_1, r_2, \ldots, r_n$ mit $r_i \in M$.
Für eine beliebige Konfiguration C:
$w_i(C)$ = minimale Kosten einer Bearbeitung von $\sigma_i = r_1, r_2, \ldots, r_i$ die in C_0 startet und in C endet.
$w_0(C)$ = Kosten eines minimalen Matchings von C_0 auf C

Lemma 6.5.2 *a)* $\text{OPT}(\sigma_i) = \min_C(C)$, $\forall\sigma, \forall i$ *(Klar)*

b) $w_i(C) = \min_{x \in C} (w_{i-1}(C - x + r_i) + d(r_i, x))$

c) $w_i(C) = w_{i-1}(C)$, *wenn* $r_i \in C$ *(Klar)*

d) $w_i(C) \geq w_{i-1}(C)$ *("Mehr Moves machen mehr Mühe")*

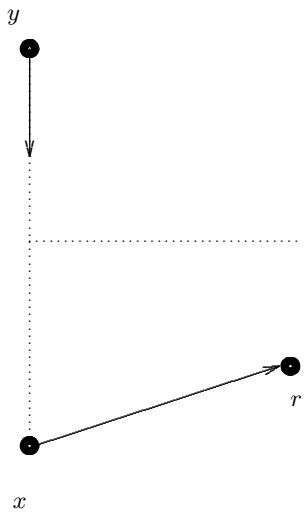


Abbildung 81: Ein Zug von SC

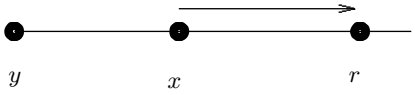


Abbildung 82: SC auf Gerade, Fall 1

Beweis 6.5.2 b: "$\leq$" klar: $w_{i-1}(C - x + r_i) = w_i(C - x + r_i)$ (6.5.2 c), denn rechts steht für jedes $x \in C$ eine Sepzielle Lösung, der auf der linken Seite verlangen Minimierungsaufgabe.
"$\geq$":

1. Fall: $r_i \in C$ Setze $x = r_i$ (6.5.2 c)

2. Fall: $r_i \notin C$. Sei C' die Konfiguration nach Bearbeitung von r_i
$\Rightarrow w_i(C) = w_i(C') + \underbrace{D(C', C)}_{39}$

$DC(C', C) = d(r_i, x) + \sum_{j=1}^{m} d(c_j, f(c_j))$ (Siehe Abbildung 84 auf der nächsten Seite.)

$\geq w_i(C - x + r_i) + d(r_i, x)$

[39] Kosten eines minimalen Matchings von C' auf C

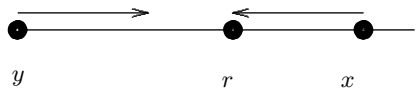


Abbildung 83: SC auf Gerade, Fall 2

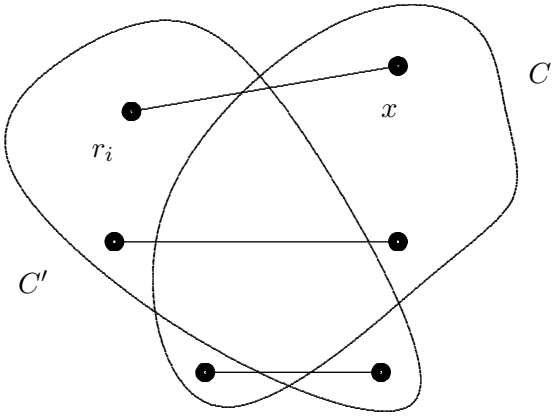


Abbildung 84: Matching zwischen C' und C

denn, $w_i(C - x + r_i) \leq w_i(C') + \underbrace{D(C', C - x + r_i)}_{=\sum_{j=1}^{m} d(c_j, f(c_j))}$

$= w_{i-1}(C - x + r_i) + d(r_i, x)$

Beweisen jetzt ein technisches Lemma.

Lemma 6.5.3 ("Quasikonvexität") $\forall i \forall$ *Konfigurationsmengen* $A, B : \forall a \in A \, \exists b \in B : w_i(A - a + b) + w_i(B - b + a) \leq w_i(A) + w_i(B)$ *(Klar* $a \in A \cap B$*, setze* $b = a$*)*

Beweis 6.5.3: Induktion $i = 0$. Sei $a \in A$ beliebig, aber fest gewählt.
Sei $M_A \subset A \times C_0$ minimales Matching von A auf C_0
Sei $M_B \subset B \times C_0$ minimales Matching von B auf C_0
Sei $c \in C_0$ der Partner von a in M_A
Sei $b \in B$ der Partner von c in M_B (Siehe Abbildung 85 auf der nächsten Seite) Dann ist:

$$\begin{array}{ccccc} w_0(A) & & D(A, C_0) & & D(A - a, C_0 - c) + d(a, c) \\ + & = & + & + & \\ w_0(B) & & D(B, C_0) & & D(B - b, C_0 - c) + d(b, c) \end{array}$$

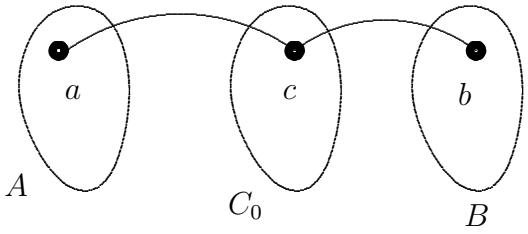


Abbildung 85: Matching Partner von a und c

$$\begin{array}{ccccc} D(A-a,C_0-c)+d(b,c) & & D(A-a+b,C_0) & & w_0(A-a+b) \\ = \quad + \quad & \geq & + & = & + \\ D(B-b,C_0-c)+d(a,c) & & D(B-b+a,C_0) & & w_0(B-b+a) \end{array}$$

$\boxed{i=0}$
(Siehe Abbildung 86)

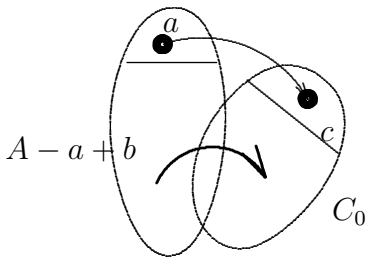


Abbildung 86: Konstruktion im Fall $i = 0$

Jetzt $i > 0$**:** Nach Lemma 6.5.2 auf Seite 93 (b):
$\exists x \in A : w_i(A) = w_{i-1}(A - x + r_i) + d(r_i, x)$
$\exists y \in B : w_i(B) = w_{i-1}(B - y + r_i) + d(r_i, y)(*)$

1. Fall: $a = x$ Setze $b = y$. Es folgt:
$w_i(A-a+b) \overset{6.5.2}{\leq} w_{i-1}\underbrace{((A-a+b)-b+r_i)}_{A-a+r_i} + d(r_i, b)$
ebenso: $w_i(B-b+a) \overset{6.5.2}{\leq} w_{i-1}\underbrace{((B-b+a)-a+r_i)}_{B-b+r_i} + d(r_i, a)$

$$\begin{array}{ccccccc} & w_i(A-a+b) & & w_{i-1}(A-a+r_i)+d(r_i,b) & & (*) & w_i(A) \\ \Rightarrow & + & \leq & + & & = & + \\ & w_i(B-b+a) & & w_{i-1}(B-b+r_i)+d(r_i,a) & x=a, y=b & & w_i(B) \end{array}$$

$\boxed{i>0 \text{ 1. Fall}}$

2. Fall: $a \neq x \Rightarrow a \in A-x \subseteq A-x+r_i$ Wende Induktionsvoraussetzung auf die Menge $A-x+r_i$ und $B-y+r_i$ und auf a an. $\Rightarrow \exists b \in B-y+r_i : w_{i-1}(A-x+r_i-a+b)+w_{i-1}(B-y+r_i-b+a) \leq w_{i-1}(A-x+r_i)+w_{i-1}B-y+r_i(**)$
$\Rightarrow w_i(A-a+b)+w_i(B-b+a)$

$$\overset{6.5.2b}{\leq} \begin{array}{c} w_{i-1}(A-a+b-x+r_i)+d(x,r_i) \\ + \\ w_{i-1}(B-b+a-y+r_i)+d(y,r_i) \end{array} \quad (**) \quad \leq \begin{array}{c} w_{i-1}(A-x+r_i)+d(x,r_i) \\ + \\ w_{i-1}(B-y+r_i)+d(y,r_i) \end{array}$$

$\overset{(*)}{=} w_i(A)+w_i(B)$ $\boxed{6.5.3}$
Damit technische Vorbereitung abgeschlossen. Kommen nun zur Definition des Work Funktion Algorithmus WFA.

Situtation: Server in Konfiguration $C_{i-1}, \sigma_{i-1} = r_1, \ldots, r_{i-1}$ schon bearbeitet. Anforderung r_i kommt an. Was tun?

Residerata:

- So sein wie OPT
- Greedy

$\Rightarrow$ Kombination.
WFA zieht seine Server in Konfiguration C mit:

- $r_i \in C$
- C minimiert $\underbrace{w_i(X)}_{\text{OPT}} + \underbrace{D(C_{i-1},x)}_{\text{Greedy}}$

Tatsächlich braucht WFA zur Minimierung von $f(x) = w_i(X)+D(C_{i-1},X)$ nur einen Server zu bewegen.

Lemma 6.5.4 $\min\limits_{C:r_i \in C} \underbrace{w_i(C)+D(C_{i-1},C)}_{=f(C)} \overset{(1)}{=} w_i(C_{i-1}$
$\overset{(2)}{=} \min\limits_{s \in C_{i-1}} (w_{i-1}(C_{i-1}-s+r_i)+d(s,r_i))$

Beweis 6.5.4: (1) "$\geq$" trival.
(2) "=" wegen Lemma 6.5.2 auf Seite 93 (b).
(1) "$\leq$": Sei $s \in C_{i-1}$ beliebig. Setze $C = C_{i-1} - s + r_i$ Dann gilt $w_i(C) + D(C_{i-1}, C) = w_i(\underbrace{C_{i-1} - s + r_i}_{C} + d(s, r_i)$ $\boxed{6.5.4}$

Das bedeutet: Mann kann WFA auch dadurch definieren, dass bei Anforderung von r_i ein Server von demjenigen $s \in C_{i-1}$ auf r_i gezogen wird, des das $w_{i-1}(C_{i-1} - s + r_i) + d(s, r_i)$ minimiert.

Neue Konfiguration: $C_{i-1} - s + r_i = C_i$
Kosten: $\text{WFA}(r_i) = d(s, r_i)$

23.07.03 "Betrache die "Offline-Pseudokosten" der Konfiguration $C_0, C_1, \ldots, C_n$ von WFA :
$w_i(C_i) - w_{i-1}(C_{i-1})$ $\boxed{\boxed{\text{teleskopiert}}}$
Addiere die echten Kosten $\text{WFA}(r_i) = d(s, r_i)$
Nun ist: $w_i(C - i) \overset{r_i \in C_i}{=} \overset{6.5.2(c)}{} w_{i-1}(C_i)$
$\overset{6.5.4}{=} \overset{\text{Def.} C_i, s}{} 0 w_i(C_{i-1}) - d(s, r_i)$, also:
$w_i(C_i) - w_{i-1}(C_{i-1}) + d(s, r_i) = w_i(C_{i-1}) - w_{i-1}(C_{i-1})$
$\leq \max_C (w_i(C) - w_{i-1}(C))$

Vorteil: Das hngt nicht mehr von WFA ab, nur von (M, d) und σ_i.
Nachteil: Verschwendung.

Zum Beweis von Theorem 6.5.1 auf Seite 93:

Hauptlemma 6.5.5 $\exists A \forall n \forall \sigma$ *der Lnge* n : $\sum_{i=1}^{n} \max_C (w_i(C) - w_{i-1}(C)) \leq 2k \cdot \text{OPT}(\sigma) + A$

Aus Hauptlemma 6.5.5 folgt Theorem 6.5.1 auf Seite 93:
$\underbrace{W_n(C_n)}_{\geq \text{OPT}(\sigma)} - \underbrace{W_0(C_0)}_{=0} + \text{WFA}(\sigma)$
$= \sum_{i=1}^{n} [w_i(C_i) - w_{i-1}(C_{i-1}) + \text{WFA}(r_i)] \leq \sum_{i=1}^{n} \max_C (w_i(C) - w_{i-1}(C))$
$\overset{6.5.5}{\leq} 2k \cdot \text{OPT}(\sigma) + A$
$\Rightarrow \text{WFA}(\sigma) \leq (2k-1)\text{OPT}(\sigma) + A$ $\boxed{6.5.1}$

Beweis 6.5.5: Überlegung: $w_i(C) - w_{i-1}(C) =?$ (0 falls $r_i \in C$) Wird maximal, wenn $w_{i-1}(C)$ mglichst klein und alle Punkte von C weit von r_i entfernt sind.

Definition C heit Minimierer fr r_i bezglich w_{i-1} falls die Funktion $f(C) = w_{i-1}(C) - \sum_{a \in C} d(a, r_i)$ fr C minimal wird. Das tuts:

Lemma 6.5.6 *(a) Jeder Minimierer fr r_i bezglich w_{i-1} ist Maximier fr $w_i(X) - w_{i-1}(X)$.*

(b) Jeder Minimierer fr r_i bezglich w_{i-1} ist auch Minimierer fr r_i bezglich w_i.

Beweis 6.5.6: Sei A ein Minimierer fr r_i bezglich w_{i-1}, d.h.:
$\forall X : w_{i-1}(A) - \sum_{a \in A} d(a, r_i) \leq w_{i-1}(X) - \sum_{x \in X} d(x, r_i) (*)$

(a): Zu zeigen: $\forall B : w_i(A) - w_{i-1}(A) \geq w_i(B) - w_{i-1}(B)$
$\Leftrightarrow \forall B : w_i(A) + w_{i-1}(B) \geq w_{i-1}(A) + w_i(B)$
$\Leftrightarrow \forall B : \min_{a \in A} (w_{i-1}(A - a + r_i) + d(a, r_i)) + w_{i-1}(B)$
$\geq w_{i-1}(A) + \min_{b \in B} (w_{i-1}(B - br_i) + d(b, r_i))$

$\Leftrightarrow \forall B : \forall a \in A \, \exists b \in B : w_{i-1}(A - a + r_i) + d(a, r_i) + w_{i-1}(B) \overset{!}{\geq} w_{i-1}(A) + w_{i-1}(B - b + r_i) + d(b, r_i)$
Sei $a \in A$ beliebig, aber fest gewhlt. Wende Lemma 6.5.3 auf Seite 95 (Quasikonvexitt) an auf:
$\mathfrak{A} = A - a + r_i, \mathfrak{a} = r_i, \mathfrak{B} = B, \mathfrak{i} = i - 1$ an:

$$\boxed{\exists b \in B : w_{i-1}(A - a + r_i - r_i + b) + w_{i-1}(B - b + r_i) \leq w_{i-1}(A - a - r_i) + w_{i-1}(B)} \quad \text{(I)}$$

Aus Voraussetzung (*) folgt fr $X = A - a + b$:
$w_{i-1}(A) - \sum_{x \in A} d(x, r_i) \leq w_{i-1}(A - a + b) - \sum_{x \in A - a + b} d(x, r_i)$

$$\Leftrightarrow \boxed{w_{i-1}(A) - d(a, r_i) \leq w_{i-1}(A - a + b) - d(b, r_i)} \quad \text{(II)}$$

Aus (I) und (II) folgt Behauptung.

Beweis (b): $\forall B : w_i(A) - \sum_{a \in A} d(a, r_i)$
$\overset{!}{\leq} w_i(B) - \sum_{b \in B} d(b, r_i)$

$\Leftrightarrow \forall B \, \forall y \in B \, \exists x \in A : w_{i-1}(A - x + r_i) + d(x, r_i) - \sum_{a \in A} d(a, r_i) \overset{!}{\leq} w_{i-1}(B -$

$y + r_i) + d(y, r_i) - \sum_{b \in B} d(b, r_i)$
Sei $y \in B$ beliebig, aber fest gewhlt
$\Rightarrow x \in A$: Wenne Lemma 6.5.3 auf Seite 95 auf $\mathfrak{A} = B - y + r_i, \mathfrak{a} = r_i, \mathfrak{B} = A$
$\exists x \in A : w_{i-1}(B-y+r_i-r_i+x)+w_{i-1}(A-x+r_i) \leq w_{i-1}(B-y+r_i)+w_{i-1}(A)$
Also:

$$\boxed{w_{i-1}(B - y + x) + w_{i-1}(A - x + r_i) - w_{i-1}(A) \leq w_{i-1}(B - y + r_i)} \quad \text{(I)}$$

Andererseits folgt aus (*) f+r $X = B - y + x$

$$w_{i-1}(A) - \sum_{a \in A} d(a, r_i) \leq w_{i-1}(B - y + x) - \underbrace{\sum_{b \in B-y+x} d(b, r_i)}_{-\sum_{b \in B} d(b,r_i)-d(y,r_i)+d(x,r_i)}$$

$$\boxed{w_{i-1}(A) - \sum_{a \in A} d(a, r_i) - w_{i-1}(B - y + x) + d(x, r_i) \leq d(y, r_i) - \sum_{b \in B} d(b, r_i)} \quad \text{(II)}$$

Aus (I) + (II) folgt Behauptung $\boxed{6.5.6}$

Zur Erinnerung: Mssen 6.5.5 auf Seite 98 zeigen, also
$\sum_{i=1}^{n} \max_C (w_i(C) - w_{i-1}(C)) \leq 2k \cdot \text{OPT}(\sigma) + A$
Haben gerade Menge C betrachtet, welche die Differenzen $w_i(C) - w_{i-1}(C)$ maximieren.
Sei A_i ein Minimierer von r_i bezglich w_{i-1}, d.h.:

$$w_{i-1}(A_i) - \sum_{a \in A_i} d(a, r_i) = \underbrace{\min_X \left(w_i(X) - \sum_{x \in X} d(x, r_i) \right)}_{=:\text{MIN}_{i-1}(r_i)}$$

$$\Rightarrow \max_C (w_i(C) - w_{i-1}(C)) \overset{6.5.6(a)}{=} w_i(A_i) - w_{i-1}(A_i)$$

$$= \underbrace{\left(w_i(A_i) - \sum_{a \in A_i} d(a, r_i) \right)}_{=\text{MIN}_i(r_i)} - \underbrace{\left(w_{i-1}(A_i) - \sum_{a \in A_i} d(a, r_i) \right)}_{=\text{MIN}_{i-1}(r_i)}$$

$\Rightarrow \max_C (w_i(C) - w_{i-1}(C)) = \text{MIN}_i(r_i) - \text{MIN}_{i-1}(r_i)$

Bis jetzt nur $(M, d), \sigma$ betrachtet.

Bringe jetzt OPT ins Spiel: Sei U_i Konfiguration von OPT nach Erledigung von $\sigma_i = r_1, \ldots, r_i$
$\text{MIN}_i(r_i) - \text{MIN}_{i-1}(r_i)$

[40] da nach 6.5.6 auf der vorherigen Seite (b) A_i auch Minimierer fr r_i bezglich w_i ist.
[41] da nach 6.5.6 auf der vorherigen Seite (a) A_i Minimierer fr r_i bezglich w_{i-1}

$$\leq \sum_{u \in U_i - r_i} \underbrace{(\mathrm{MIN}_i(u) - \mathrm{MIN}_{i-1}(u))}_{>0}$$

$$\mathrm{MIN}_i(u) \stackrel{\text{def}}{=} \min_X \Big(\underbrace{w_i(X)}_{\geq w_{i-1}(X)} - \sum_{a \in X} d(a,u) \Big)$$
$$\geq \min_X (w_{i-1}(X) - \sum_{a \in X} d(a,u)) \stackrel{\text{def}}{=} \mathrm{MIN}_{i-1}(u)$$

Zwischenergebnis: $\max_C (w_i(C) - w_{i-1}(C)) \leq \sum_{u \in U_i} \mathrm{MIN}_i(u) - \sum_{u \in U_i} \mathrm{MIN}_{i-1}(u) (**)$

Charakter von Potentialfunktionen, teleskopiert nicht

Angenommen, OPT zieht von Server von z nach $r_i \Rightarrow U_i = U_{i-1} - z + r_i$

$$\sum_{u \in U_i} \mathrm{MIN}_{i-1}(i) - \sum_{u \in U_{i-1}} \mathrm{MIN}_{i-1}(u) = \mathrm{MIN}_{i-1}(r_i) - \mathrm{MIN}_{i-1}(z)$$

$$\mathrm{MIN}_{i-1}(r_i) = \min_X \left(w_{i-1}(X) - \sum_{a \in X} d(a, r_i) \right)$$
$$\stackrel{\triangle \neq}{\geq} \min_X \left(w_{i-1}(X) - \sum_{a \in X} (d(a,z) + d(z, r_i) \right)$$
$$= \min_X \left(w_{i-1}(X) - \sum_{a \in X} d(a,z) \right) \stackrel{def}{=} \mathrm{MIN}_{i-1}(z) - k d(z, r_i)$$
$$\Rightarrow \sum_{u \in U_i} \mathrm{MIN}_{i-1}(u) - \sum_{u \in U_{i-1} \mathrm{MIN}_{i-1}(u) \geq k \cdot d(z, r_i} (***)$$
$$(**) + (***) : \max_C (w_i(C) - w_{i-1}(C)) - k \cdot \underbrace{d(z, r_i)}_{\mathrm{OPT}(r_i)}$$
$$\leq \sum_{u \in U_i} \mathrm{MIN}_i(u) - \sum_{u \in U_i} \mathrm{MIN}_{i-1}(u) + \sum_{u \in U_i} \mathrm{MIN}_{i-1}(u) - \sum_{u \in U_{i-1}} \mathrm{MIN}_{i-1}(u)$$
$$= \sum_{u \in U_i} \mathrm{MIN}_i(u) - \sum_{u \in U_{i-1}} \text{ Jetzt teleskopiert es.}$$

Es folgt: $\sum_{i=1}^{n} \max_C (w_i(C) - w_{i-1}(C)) - k \cdot \mathrm{OPT}(\sigma) = \sum_{i=1}^{n} \Big(\max_C (w_i(C) - w_{i-1}(C)) - k \cdot \mathrm{OPT}(r_i) \Big)$

$$\leq \underbrace{\sum_{u \in U_n} w_n(u)}_{\leq k \cdot \mathrm{OPT}(\sigma)} - \underbrace{\sum_{u \in U_0} \mathrm{MIN}_0(u)}_{=A}$$

Sei $y \notin U_n$ Dann $\mathrm{MIN}_n(u) \stackrel{\text{Def}}{=} \min_X (w_n(X) - \sum_{a \in X} d(a,u)) \leq w_n(U_n - u + y) -$

$d(y,a) \leq w_n(U_n) = \mathrm{OPT}(\sigma)$.

6.5.1

Index