Rheinische Friedrich-Wilhelms-Universität Bonn
Institut für Informatik, Abteilung I
Prof. Dr. Rolf Klein, Dr. Elmar Langetepe

Seminar Randomisierte Algorithmen
Wintersemester 2002/2003

Thema: Online-Algorithmen

Ausarbeitung von Duc Luu

1 Vorwort/Rückblick

Dieses ist die Ausarbeitung zum zweiten von drei Vorträgen über Randomisierte Online-Algorithmen. Die Inhalte basieren auf folgendem Buch:

> Rajeev Motwani & Prabhakar Raghavan
> Randomized Algorithms
> Cambridge University Press, 1995

Zunächst stichpunktartig eine Liste der im ersten Teil abgehandelten Themen:

- Paging-Problem
- Algorithmen für Offline- und Online-Paging
- optimaler Offline-Algorithmus MIN
- randomisierte Paging-Algorithmen
- Kompetivität
- Gegner-Modelle:
  - unwissender Gegner (oblivious adversary)
  - adaptiver Offline-Gegner (adaptive Offline adversary)
  - adaptiver Online-Gegner (adaptive online adversary)
- Kompetivitätskoeffizienten für Gegnermodelle
- untere Schranke für Kompetivität von deterministischen Online-Algorithmen

Im zweiten Teil untersuchen wir die Leistungsfähigkeit randomisierter Algorithmen gegen unwissende Gegner. Danach betrachten wir die Zusammenhänge zwischen den Gegnermodellen.

2 Unwissende Gegner

Als erstes wollen wir die Kompetivität randomisierter Algorithmen gegen unwissende Gegner untersuchen. Im ersten Teil wurde bereits der Kompetivitätskoeffizient $\mathcal{C}_R^{obl}$ eines randomisierten Online-Algorithmus R gegen einen unwissenden Gegner definiert. Für diesen gilt es nun, eine untere Schranke zu finden.

Mit einem kleinen Trick wollen wir uns die Untersuchung etwas vereinfachen. Betrachten wir also zunächst eine Wahrscheinlichkeitsverteilung P über die Eingabesequenz $\rho = (\rho_1, \ldots, \rho_N)$, $\rho_i \in I$. D.h. P gibt an, mit welcher Wahrscheinlichkeit ein ρ_i aus der Menge I der Speicher-Items gewählt wird. Die Wahl eines ρ_i darf dabei von der Wahl der $\rho_1, \ldots, \rho_{i-1}$ abhängen. Die Kostenfunktion *cost* unter der Eingabe ρ ist nun eine Zufallsvariable, da die Sequenz ρ zufällig ist.

Definition.

Sei P eine Wahrscheinlichkeitsverteilung über die Eingabesequenz $\rho = (\rho_1, \ldots, \rho_N)$ und A ein deterministischer Online-Algorithmus für Paging. Dann ist $\mathcal{C}_A^P$ das Infimum aller $\mathcal{C}$, so dass gilt:

$$E[cost_A(\rho)] - \mathcal{C} \times E[cost_O(\rho)] \leq b$$

A heißt dann $\mathcal{C}_A^P$-kompetitiv unter P. Die Konstante b darf wieder von der Größe k des Caches, aber nicht von der Länge N der Eingabesequenz abhängen.

Wir nehmen nun an, dass A die Verteilung von P kennt. A weiß also, wie P „würfelt“, d.h. mit welcher Wahrscheinlichkeit ein bestimmtes ρ_i gewählt wird, ohne aber das Ergebnis des Würfelns zu kennen.

Dann gilt folgende Gleichheit:

$$\inf_R \mathcal{C}_R^{obl} = \sup_P \inf_A \mathcal{C}_A^P.$$

Diese Gleichung lässt sich aus Yaos Minimax-Prinzip und dem Loomis-Theorem aus der Spieltheorie herleiten. Unter dem Stichwort „distributional complexity“ wird folgendes erklärt: Die Laufzeit des besten deterministischen Algorithmus unter der schlechtesten Verteilung gleicht der Laufzeit eines randomisierten Algorithmus gegen einen unwissenden Gegner.

Spieltheoretisch lässt sich das auch so ausdrücken: Einerseits haben wir einen Spieler R mit randomisierter Strategie. Ihm gegenüber sitzt ein Gegenspieler, der die Randomisierung von R kennt und ihn, unter Ausnutzung seiner Kenntnis, mit einer möglichst schlechten Eingabe versucht auszutricksen. Der Gegenspieler kennt aber nur die Wahrscheinlichkeit, mit der R seine Strategien auswählt, das Ergebnis von Rs Wahl kennt er nicht. Auf der anderen Seite haben wir Spieler A mit einer festen Strategie. Der Gegenspieler versucht A

durch Würfeln über die Eingabe hereinzulegen. A kennt die Wahrscheinlichkeit, mit der sein Gegenspieler würfelt, aber nicht das Ergebnis des Würfelns.

Wir machen uns diese Eigenschaft zu Nutze und untersuchen nicht randomisierte Algorithmen gegen unwissende Gegner, sondern stattdessen deterministische Algorithmen unter zufälliger Eingabe. Können wir eine untere Schranke für $\mathcal{C}_A^P$ finden, so gilt dieselbe auch für $\mathcal{C}_R^{obl}$: $\exists P\ \forall A:\ \mathcal{C}_A^P \geq \mathcal{C} \Longrightarrow \forall R:\ \mathcal{C}_R^{obl} \geq \mathcal{C}$.

Satz.

Sei R ein randomisierter Algorithmus für Paging. Dann gilt: $\mathcal{C}_R^{obl} \geq H_k$, wobei $H_k = \sum_{j=1}^{k} 1/j$ die k-te Harmonische Zahl ist.

Beweis.

Wir beweisen diese Eigenschaft für $\mathcal{C}_A^P$. Gegeben sein ein Cache der Größe k und eine Menge I von $(k+1)$ Speicher-Items, $I = \{I_1,\ \ldots,\ I_{k+1}\}$. Das bedeutet, dass ein Algorithmus pro Zeitpunkt ein Item außerhalb des Caches lassen muss. Gegeben sei weiter eine Wahrscheinlichkeitsverteilung P auf der Eingabesequenz ρ. Dabei werde ρ wie folgt erzeugt: Wähle ρ_1 gleichverteilt aus I und ρ_i gleichverteilt aus $I \backslash \{\rho_{i-1}\}$ für $i > 1$. Es darf also nicht dasselbe Item zweimal hintereinander angefragt werden.

Jetzt lassen wir zwei Algorithmen auf einer solchen Sequenz laufen: Einen optimalen Offline-Algorithmus und einen deterministischen Online-Algorithmus. Dabei analysieren wir die erwartete minimale Fehlerzahl des Offline- und maximale Fehlerzahl des Online-Algorithmus. Das Verhältnis dieser beiden Zahlen ist per Definition die Kompetitivität.

Analysieren wir zuerst das Verhalten des Offline-Agorithmus. Dieser teilt die Eingabesequenz in Runden auf. Die erste Runde beginnt mit der ersten Anfrage und endet unmittelbar nach der ersten Anfrage des $(k+1)$-ten Items. Alle anderen Runden beginnen nach Beendigung der vorhergehenden Runde und enden wie die erste Runde. Eine Runde enthält also $(k+1)$ verschiedene Items, und das $(k+1)$-te Item tritt genau einmal auf, nämlich bei der letzten Anfrage. Setzt der Offline-Agorithmus auf jede Runde den MIN-Algorithmus an, so lässt er das Item außerhalb des Caches, dessen Anfrage am weitesten in der Zukunft liegt, und zwar das letzte der Runde. Damit verfehlt er einmal pro Runde.

Kommen wir nun zum Verhalten des Online-Algorithmus. Wir wollen wissen, wie häufig dieser in einer Runde verfehlt. Auch er muss zu jedem Zeitpunkt ein Item außerhalb des Caches lassen. Ein Fehler tritt auf, wenn genau dieses Items angefragt wird. Da ein Item nicht zweimal angefragt werden darf, wird das fehlende Item zum Zeitpunkt i aus der k-elementigen Menge $I \backslash \{\rho_{i-1}\}$ gleichverteilt gewählt. Daher ist die Wahrscheinlichkeit für einen Fehler $1/k$ pro Anfrage.

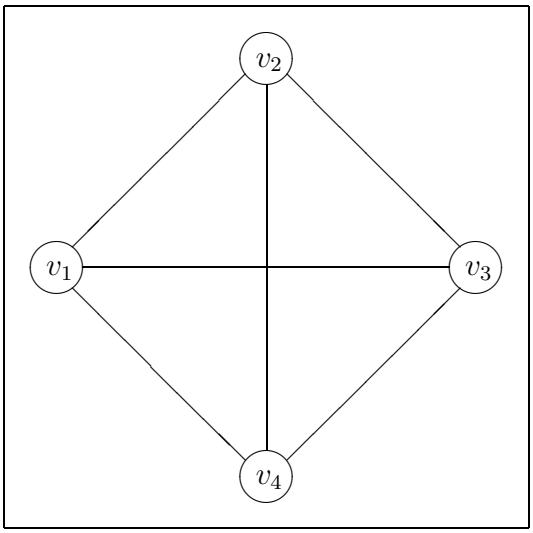

Abbildung 1: *Vollständiger Graph mit 4 Knoten.*

Nun stellt sich die Frage, wie viele Anfragen in einer Runde gestellt werden. Das führt uns zu der Frage nach der erwarteten Länge einer Runde. Formen wir das Problem um und betrachten einen vollständigen (ungerichteten) Graphen[1] mit n Knoten (vgl. Abb. 1). Wir führen einen Random Walk[2] darauf aus und betrachten die erwartete Anzahl an Schritten, um n Knoten zu besuchen. Setzen wir $n = k+1$, so ist dieses Problem äquivalent zum Problem, eine erwartete Rundenlänge mit $(k+1)$ verschiedenen Items zu bestimmen. Anstatt n verschiedene Knoten zu besuchen, wollen wir $(k+1)$ verschiedene Items anfragen.

Sei X_i die Anzahl der Schritte, um i verschiedene Knoten zu besuchen. Haben wir bereits i Knoten besucht, dann ist $(X_i - X_{i-1})$ die Anzahl der Schritte, um einen neuen Knoten zu besuchen. Wir haben n Knoten und befinden uns auf einem dieser Knoten, also können wir als nächstes nur einen der $(n-1)$ Knoten besuchen. (Deshalb haben wir vorausgesetzt, dass nicht zwei Items hintereinander angefragt werden dürfen: Man kann einen Knoten nicht zweimal hintereinander besuchen.) Nach i bereits besuchten Knoten kann man $(n-i)$ neue Knoten besuchen. Die Wahrscheinlichkeit für $(X_i - X_{i-1})$ ist also $\frac{n-i}{n-1}$. Diese Zufallsvariable ist geometrisch verteilt, denn wir besuchen solange einen der i bereits besuchten Knoten, bis wir zum ersten Mal auf einen neuen Knoten kommen. Bei der geometrischen Verteilung ist der Erwartungswert einer Zufallsvariablen der Kehrwert ihrer Wahrscheinlichkeit, womit folgt: $E[X_i - X_{i-1}] = \frac{n-1}{n-i}$.

[1] Vollständiger Graph: Graph, bei dem alle Knoten miteinander verbunden sind (außer mit sich selbst).
[2] Random Walk: Zufälliges Durchlaufen des Graphen, d.h. der nächste zu besuchende Knoten ist zufällig.

Wir suchen X_n und können dies nun durch Aufsummieren der $(X_i - X_{i-1})$ berechnen:

$$E[X_n] = \sum_{i=1}^{n-1} E[X_{i+1} - X_i] = \sum_{i=1}^{n-1} \frac{n-1}{n-i} \overset{(*)}{=} (n-1) \sum_{i=1}^{n-1} \frac{1}{n-i}$$

$$\overset{(**)}{=} (n-1) \sum_{i=1}^{n-1} \frac{1}{i} = (n-1) H_{n-1}$$

Eine kurze Erklärung zur obigen Rechnung: Im Schritt $(*)$ wurde der Zähler $(n-1)$ vor die Summe gezogen, da er keinen Summationsindex enthält, während im Schritt $(**)$ die Summe rückwärts hingeschrieben wurde.

Kommen wir auf unser ursprüngliches Problem zurück. Wir suchen die erwartete Rundenlänge bei Anfragen nach $(k+1)$ verschiedenen Items und erhalten mit der obigen Formel $X_{k+1} = k \cdot H_k$. Erwartungsgemäß werden also pro Runde $k \cdot H_k$ Anfragen gestellt. Bei jeder dieser Anfragen ist die Fehlerwahrscheinlichkeit $1/k$. Damit haben wir eine erwartete Fehlerzahl von $(1/k) \cdot k \cdot H_k = H_k$ in einer Runde. Da der optimale Algorithmus genau einen Fehler pro Runde macht, impliziert das die Ungleichung $E[cost_A(\rho)] - (H_k) \times E[cost_O(\rho)] \leq b$. Damit ist A H_k-kompetitiv unter P und gleichzeitig R H_k-kompetitiv gegen unwissende Gegner, also $\mathcal{C}_R^{obl} \geq H_k$. □

Nun also wissen wir, dass randomisierte Algorithmen gegen unwissende Gegner keine kleineren Kompetivitätskoeffizienten als H_k haben. Definieren wir $\mathcal{C}^{obl}$ als kleinsten Kompetitivitätskoeffizienten über alle randomisierten Algorithmen gegen unwissende Gegner und auf dieselbe Weise $\mathcal{C}^{aof}$ und $\mathcal{C}^{aon}$. Der kleinste Kompetitivitätskoeffizient über alle deterministischen Algorithmen sei bezeichnet als $\mathcal{C}^{det}$. Somit gilt die Ungleichung $\mathcal{C}^{obl} \geq H_k$. Im ersten Teil haben wir festgestellt, dass $\mathcal{C}^{det} = k$ ist (alle deterministischen Algorithmen machen mindestens k Fehler, und mindestens einer erreicht diese untere Schranke). Vergleicht man diese unteren Schranken, so stellen wir fest, dass randomisierte Algorithmen (gegen unwissende Gegner) gemessen an den Kompetitivitätskoeffizienten bessere Ergebnisse erzielen als die deterministischen, da $k \geq H_k$. Gegen unwissende Gegner bringt Randomisierung also Vorteile gegenüber Determinismus.

Dass es einen Algorithmus gibt, der die untere Schranke von H_k auch erreicht, haben L.A. McGeoch und D.D. Sleator gezeigt. Damit gilt also die Gleichheit $\mathcal{C}^{obl} = H_k$. Diesen Algorithmus wollen wir hier jedoch nicht untersuchen. Stattdessen stellen wir hier den *Marker*-Algortihmus vor, der diese untere Schranke zwar nicht erreicht, ihr aber sehr nahe kommt. Er arbeitet wie folgt: Jede der k Cache-Speicherzellen hat ein Marker-Bit, womit sich im Cache befindliche Items markieren lassen. Bei ihrer Bearbeitung wird die Anfragesequenz in Runden aufgeteilt. Am Anfang jeder Runde werden die Marker-Bits auf 0 (null) zurückgesetzt. Befindet sich ein angefragtes Item im Cache, wird es markiert. Falls nicht, so muss es aus dem Hauptspeicher geholt werden und ersetzt dabei ein zufällig gewähltes

unmarkiertes Item im Cache. Nach dem Tausch wird das angefragte Item markiert. Eine Runde ist beendet, sobald alle Cache-Zellen markiert wurden und ein Fehlgriff erfolgt. Die nächste Runde beginnt, alle Marker-Bits werden zurückgesetzt, und die erste Anfrage ist ein Fehlgriff.

Satz.

Der Marker-Algorithmus ist $(2H_k)$-kompetitiv gegen unwissende Gegner.

Beweis.

Wir lassen wieder den optimalen Offline-Algorithmus und den Marker-Algorithmus auf derselben Eingabesequenz laufen und vergleichen deren Fehlerzahl. Dabei nehmen wir an, dass beide Algorithmen mit denselben Items im Cache starten und dass die erste Anfrage einen Fehler verursacht.

Der Marker-Algorithmus teilt die Eingabesequenz wie oben beschrieben in Runden auf, wobei die erste Runde mit der ersten Anfrage beginnt. Ganz allgemein beginnt eine Runde mit Anfrage ρ_i und endet mit Anfrage ρ_j, wobei ρ_{j+1} die erste Anfrage nach dem $(k+1)$-ten verschiedenen Item ist. Jede Runde beginnt also mit einem Fehlgriff, und am Ende jeder Runde befinden sich alle angefragten Items im Cache, wobei alle markiert sind.

Für den weiteren Beweis führen wir nun ein paar Begriffe ein. Alle definierten Begriffe beziehen sich auf eine gerade betrachtete Runde. Ein *stale Item* ist ein Item, das sich zu Anfang einer Runde (vor der Bearbeitung der ersten Anfrage) im Marker-Cache befindet, während ein *clean Item* ein Item ist, welches zu Anfang einer Runde *nicht* im Marker-Cache anzufinden ist. Sei l die Anzahl der Anfragen nach clean Items in einer Runde. Falls wir uns nicht gerade in der ersten Runde befinden, so wurde ein stale Item in der vorhergehenden Runde angefragt. Bezeichne S_M den Cache des Marker-Algorithmus und S_O den Cache des optimalen Offline-Algorithmus. Dann ist $d_A = |S_O \backslash S_M| = |S_M \backslash S_O|$ die Kardinalität der Differenzmenge der beiden Caches am Anfang einer Runde. Analog definiere man d_E für das Ende einer Runde. d_A bzw. d_E ist die Anzahl der nicht übereinstimmenden Items in beiden Caches. Sei beispielsweise $S_O = [\underline{1}, \underline{2}, 3, 4, 5]$ und $S_M = [3, 4, 5, \underline{6}, \underline{7}]$. Sind das die Zustände am Anfang einer Runde, so ist $d_A = |S_O \backslash S_M| = |\{1, 2\}| = 2$. Werden im Laufe der Runde die Items $1, 2, 3, 4, 8$ angefragt, so sind $1, 2, 8$ clean Items und somit $l = 3$. Obwohl die Items 1 und 2 sich zu Anfang der Runde in S_O befinden, sind sie clean, da die Begriffe stale und clean nur über den Marker-Cache definiert sind. Am Ende der Runde seien $S_O = [1, 2, 4, \underline{5}, 8]$ und $S_M = [1, 2, 3, \underline{4}, 8]$. Folglich ist $d_E = |\{5\}| = 1$. Da beim Übergang von einer Runde zur nächsten nur die Marker-Bits zurückgesetzt werden, stimmen die Caches am Ende einer Runde mit denen am Anfang der nächsten Runde überein, so dass das d_E am Ende einer Runde dem d_A der darauffolgenden Runde gleicht.

Nun kommen wir zur Fehleranalyse der beiden Algorithmen und beginnen mit dem Offline-

Algorithmus. In einer betrachteten Runde gibt es l Anfragen nach clean Items und $(k-l)$ Anfragen nach stale Items. Da die stale Items in der vorherigen Runde angefragt wurden, befinden sie sich möglicherweise im Offline-Cache und produzieren möglicherweise keine Fehler (im Best Case). Also müssen wir die Fehler bei den Anfagen nach clean Items suchen. Nehmen wir zu Beginn der Runde aus dem Offline-Cache S_O die Items heraus, die in der vorherigen Runde angefragt wurden (also alle Items in S_M und somit alle stale Items), so bleiben in der Differenzmenge $S_O \backslash S_M$ nur clean Items übrig, und deren Anzahl ist d_A. Werden diese angefragt, so sind diese im besten Fall noch im Cache, so dass diese Anfragen keine Fehler produzieren. Es bleiben also $(l - d_A)$ Items, die sich zu Anfang der Runde in jedem Fall nicht im Cache befinden und somit Fehler produzieren. Wie bereits festgestellt, befinden sich am Ende der Runde im Marker-Cache nur Items, die angefragt wurden. Bilden wir die Differenzmenge $S_M \backslash S_O$, so bleiben nur Items übrig, die angefragt wurden, sich jedoch nicht mehr im Offline-Cache befinden (d_E Stück). Da sie angefragt wurden, müssen sie im Offline-Cache gewesen und (vom Offline-Algorithmus) wieder entfernt worden sein, was einen Fehler impliziert. Somit haben wir in einer Runde mindestens $\max\{l - d_A, d_E\} \geq \frac{l-d_A+d_E}{2}$ Fehler. (Die Ungleichung gilt, da das Maximum zweier Zahlen immer größer oder gleich dem arithmetischen Mittel ist, so dass wir eine untere Schranke für die Mindestfehlerzahl haben.)

Nun Amortisieren wir die Fehlerzahl, indem wir die Gesamtfehlerzahl auf alle Runden verteilen. Summieren wir die Fehler $\frac{l-d_A+d_E}{2}$ aller Runden auf, so erhalten wir im Zähler eine Teleskopsumme, da die d_E und d_A zweier aufeinanderfolgenden Runden sich aufheben, so dass nur noch das d_A der ersten und das d_E der letzten Runde übrig bleiben. Da wir vorausgesetzt haben, dass beide Algorithmen mit denselben Items im Cache starten, ist in der ersten Runde $d_A = 0$. Folglich bleibt nur das d_E der letzten Runde übrig. Diese Konstante, die nicht von der Länge der Eingabesequenz abhängt, ist zu vernachlässigen, da wir sie auf alle Runden verteilen und der Anteil einer Runde an diesem Fehler somit sehr klein wird. Bei länger werdenden Eingabesequenzen wird dieser Anteil immer kleiner. Es bleiben also die $l/2$ Fehler pro Runde, die der Offline-Algorithmus mindestens macht, also gilt $cost_O(\rho) \geq l/2$.

Kommen wir nun zum Marker-Algorithmus. Dessen Cache enthält zu Anfang einer Runde nur stale Items, so dass er bei jeder der l Anfragen nach clean Items einen Fehler macht. Es bleiben noch die Fehler, die Marker zusätzlich bei den $(k-l)$ Anfragen nach stale Items produziert. Werden diese vor den clean Items angefragt, so befinden sie sich noch im Cache, so dass sie keine Fehler produzieren. Werden jedoch zuerst die clean Items angefragt, so können dadurch stale Items, die später noch angefragt werden, aus dem Cache genommen werden, so dass auch die Anfragen nach diesen stale Items Fehler produzieren. Untersuchen wir also letzteren Fall genauer.

Nehmen wir zur Worst-Case-Analyse an, dass bereits alle l clean Items angefragt wurden und nun die Anfragen nach den $(k-l)$ stale Items folgen. Bei der Anfrage nach dem ersten stale Item sind es erwartete l/k Fehler aus oben genannten Gründen (maximal l stale Items

mussten bei Anfrage nach l clean Items aus dem Cache befördert werden). Bei der zweiten Anfrage sind es $\frac{l+(l/k)}{k}$ erwartete Fehler. (Im Zähler: l Fehler bedingt durch clean Items sowie l/k Fehler bedingt durch das erste stale Item.) Durch Umformungen erhalten wir also

$$\frac{l+\frac{l}{k}}{k} = \frac{l\left(1+\frac{1}{k}\right)}{k} = \frac{l}{k}\cdot\frac{k+1}{k} \leq \frac{l}{k}\cdot\frac{k}{k-1} = \frac{l}{k-1}$$

erwartete Fehler, die bei der dritten Anfrage wieder zum Zähler addiert werden: $\frac{l+\frac{l}{k}+\frac{l}{k-1}}{k}$. Das setzt sich weiter fort, so dass wir allgemein bei der i-ten Anfrage (nach einem stale Item)

$$\begin{aligned}
& \frac{l+\frac{l}{k}+\frac{l}{k-1}+\ldots+\frac{l}{k-(i-2)}}{k} \\
\overset{(*1)}{=} \quad & \frac{l\left(1+\frac{1}{k}+\frac{1}{k-1}+\ldots+\frac{1}{k-(i-2)}\right)}{k} \\
\overset{(*2)}{=} \quad & \frac{l\left(1+\frac{1}{k}+\frac{1}{k-1}+\ldots+\frac{1}{k-(i-2)}\right)\cdot(k-(i-1))}{k\cdot(k-(i-1))} \\
\overset{(*3)}{=} \quad & \frac{l\left((k-(i-1))+\frac{k-(i-1)}{k}+\frac{k-(i-1)}{k-1}+\ldots+\frac{k-(i-1)}{k-(i-2)}\right)}{k(k-(i-1))} \\
\overset{(*4)}{\leq} \quad & \frac{l\cdot k}{k\cdot(k-(i-1))} = \frac{l}{k-i+1}
\end{aligned}$$

erwartete Fehler haben. Zur Erklärung: Im Schritt $(*1)$ wurde l ausgeklammert, im Schritt $(*2)$ wurde der Bruch erweitert und im Schritt $(*3)$ wurde im Zähler der Term $(k-(i-1))$ in die Klammer multpliziert. Um Schritt $(*4)$ nachzuvollziehen, überlegen wir uns, dass alle Summanden in $\frac{k-(i-1)}{k}+\frac{k-(i-1)}{k-1}+\ldots+\frac{k-(i-1)}{k-(i-2)}$ kleiner 1 sind, so dass die Summe kleiner $(i-1)$ ist. Also ist der Zähler kleiner $l(k-(i-1)+(i-1)) = l\cdot k$, womit die Ungleichung $(*4)$ erfüllt ist.

Für die i-te Anfrage nach einem stale Item haben wir also erwartungsgemäß $\frac{l}{k-i+1}$ Fehler. Nimmt man alle $(k-l)$ Anfragen zusammen, erhalten wir

$$\begin{aligned}
\sum_{i=1}^{k-l}\frac{l}{k-i+1} &= l\left(\frac{1}{k}+\frac{1}{k-1}+\ldots+\frac{1}{l+2}+\frac{1}{l+1}\right) \\
&= l\left(\sum_{j=1}^{k}\frac{1}{j}-\sum_{j=1}^{l}\frac{1}{j}\right) = l(H_k-H_l)
\end{aligned}$$

Fehler bei Anfragen nach stale Items. Nehmen wir die l Fehler bei clean Items hinzu, so ist die gesamte Fehlerzahl $l + l(H_k - H_l) \leq l(H_k + \underbrace{1 - H_l}_{\leq 0}) \leq lH_k$ in einer Runde.

Daraus können wir schließen:

$$E[cost_M(\rho)] \leq lH_k = 2H_k \cdot \frac{l}{2} \leq 2H_k \cdot cost_O(\rho)$$

was folgende Ungleichung impliziert:

$$E[cost_M(\rho)] - (2H_k) \cdot cost_O(\rho) \leq b$$

Somit haben wir die $(2H_k)$-Kompetivität des Marker-Algorithmus gegen unwissende Gegner bewiesen. □

3 Vergleich der Gegnermodelle.

Nachdem wir nun die unwissenden Gegner kennengelernt haben, untersuchen wir als nächstes adaptive Gegner. Dabei wollen wir wieder die Leistungsfähigkeit randomisierter Algorithmen gegen diese Gegner betrachen. Zunächst führen wir jedoch einige Begriffe ein.

Definition.

Ein deterministischer Online-Algorithmus A für Paging heißt α-kompetitiv, wenn für alle Eingabesequenzen ρ gilt: $cost_A(\rho) \leq \alpha[cost_O(\rho)]$.

Wir betrachten in Bezug auf adaptive Gegner also keine multiplikativen Kompetitivitätskoeffizienten mehr, sondern Funktionen. Jedoch ist die multiplikative Kompetitivität in der funktionalen enthalten, da man die Funktionen entsprechend konstruieren kann. Bsp.: Ist ein Algorithmus A α-kompetitiv bzgl. der Funktion $\alpha[n] = \mathcal{C} \cdot n + b$, so ist sie auch $\mathcal{C}$-kompetitiv, denn es ist $cost_A(\rho) \leq \mathcal{C} \cdot cost_O(\rho) + b \iff cost_A(\rho) - \mathcal{C} \cdot cost_O(\rho) \leq b$.

Auf dieselbe Weise kann man die Kompetitivität randomisierter Algorithmen gegen unwissende Gegner definieren. Die Unterschiede zu deterministischen Algorithmen sind, dass die Kosten Zufallsvariable sind, so dass wir den Erwartungswert der Kosten betrachten müssen, und dass der unwissende Gegner die Anfragen erzeugt.

Definition.

Ein randomisierter Online-Algorithmus R für Paging heißt α-kompetitiv gegen unwissende Gegner, wenn für alle Eingabesequenzen ρ gilt: $E[cost_R(\rho)] \leq \alpha[cost_O(\rho)]$.

Beim Paging gegen adaptive Gegner ist die Eingabesequenz nicht mehr fest, da sie in Abhängigkeit der randomisierten Antworten erzeugt wird. Zur Erinnerung: Der Gegner generiert eine Anfrage, worauf R antwortet. Abhängig von den bisherigen Antworten generiert der Gegner die nächste Anfrage. Daher bezeichnen wir die zufälligen Eingaben ρ_R. Der adaptive Online-Gegner beantwortet selber die aktuelle Anfrage, bevor er die nächste generiert, während der adaptive Offline-Gegner zuerst die komplette Eingabesequenz erzeugt und dann seine (optimale) Strategie anwendet. Daher definieren wir die Kompetitivität nicht mehr über die Kosten des optimalen Offline-Algorithmus, sondern über die des jeweiligen Gegners.

Definition.

Ein randomisierter Online-Algorithmus R für Paging heißt α-kompetitiv gegen adaptive Offline-Gegner, wenn für alle Eingabesequenzen ρ_R gilt: $E[cost_R(\rho_R)] \leq \alpha[cost_Q(\rho_R)]$.

Definition.

Ein randomisierter Online-Algorithmus R für Paging heißt α-kompetitiv gegen adaptive Online-Gegner, wenn für alle Eingabesequenzen ρ_R gilt: $E[cost_R(\rho_R)] \leq \alpha[cost_S(\rho_R)]$.

Dabei seien Q bzw. S die jeweiligen adaptiven Offline- bzw. Online-Gegner, die die Eingabesequenz erzeugen. Auch können wir $cost_Q$ als optimal annehmen, da Q i.a. eine optimale Offline-Strategie zur Anfragebearbeitung anwendet.

Nun kennen wir also auch den Begriff der Kompetitivität gegen adaptive Gegner und können nun zum ersten Vergleich zwischen den Gegnern kommen. Dazu formulieren wir folgenden

Satz (ohne Beweis).

Sei R ein randomisierter Online-Algorithmus, der α-kompetitiv ist gegen alle adaptiven Offline-Gegner. Dann gibt es einen α-kompetitiven deterministischen Algorithmus.

Mit anderen Worten bedeutet das: Sobald ein adaptiver Offline-Gegner die Eingabesequenz für R erzeugt, gibt es immer einen deterministischen Algorithmus, der genauso effizient arbeitet wie R. Während Randomisierung gegen unwissende Gegner also Effizienzgewinne bringt, können adaptive Offline-Gegner die Vorteile der Randomisierung aushebeln. Hier sieht man, wie mächtig adaptive Offline-Gegner gegenüber unwissenden Gegnern sind.

Des Weiteren kann man aus obigem Satz schließen: Es gibt zu jedem randomisierten einen deterministischen Algorithmus mit derselben Kompetitivität, aber möglicherweise gibt es auch deterministische Algorithmen mit besseren Kompetitivitätskoeffizienten, so dass wir folgern können: $\mathcal{C}^{det} \leq \mathcal{C}^{aof}$.

Um schließlich die adaptiven Online-Gegner in unsere Untersuchungen einzubeziehen, benötigen wir noch folgenden

Satz (ohne Beweis).

Sei R α-kompetitiv gegen alle adaptiven Online-Gegner, und es gebe einen β-kompetitiven randomisierten Algorithmus gegen alle unwissenden Gegner. R ist dann $(\alpha \circ \beta)$-kompetitiv gegen alle adaptiven Offline-Gegner.

Dabei ist $\circ$ die Komposition zweier Funktionen, also $(\alpha \circ \beta)[n] = \alpha[\beta[n]]$.

Hier sehen wir, dass die Kompetitivität randomisierter Algorithmen gegen adaptive Offline-Gegner von der Kompetitivität gegen unwissende und adaptive Online-Gegner abhängt.

Vorhin hatten wir bereits festgestellt, dass sich die Kompetitivitätskoeffizienten durch Funktionen darstellen lassen. Wenden wir das auf obigen Satz an, so können wir insbesondere aus obigem Satz folgern: $\mathcal{C}^{aof} \leq \mathcal{C}^{aon}\mathcal{C}^{obl}$. Nach Kombination beider Sätze erhalten wir $\mathcal{C}^{det} \leq \mathcal{C}^{aof} \leq \mathcal{C}^{aon}\mathcal{C}^{obl}$. Mit den bereits hergeleiteten Schranken $\mathcal{C}^{det} = k$ und $\mathcal{C}^{obl} = H_k$ können wir eine untere Schranke für $\mathcal{C}^{aon}$ bestimmen:

$$\mathcal{C}^{aon} \geq \frac{\mathcal{C}^{det}}{\mathcal{C}^{obl}} = \Omega(k/H_k).$$

Diese Kompetitivität ist wieder besser als die deterministischer Algorithmen.

4 Zusammenfassung.

Die hier gemachten Analysen der Gegnermodelle haben folgendes ergeben: Randomisierte Algorithmen erreichen gegen unwissende Gegner eine Kompetitivität von H_k und schlagen damit die deterministischen Algorithmen, die immer mindestens k Fehler produzieren. Gegen adaptive Offline-Gegner bringt Randomisierung keine Vorteile mehr, da es immer einen deterministischen Algorithmus mit gleicher Kompetitivität gibt. Die Kompetitivität randomisierter Algorithmen gegen adaptive Offline-Gegner hängt mit der gegen unwissende und adaptive Online-Gegner zusammen. Daraus kann man wiederum schließen, dass randomisierte Algorithmen gegen adaptive Online-Gegner immer mindestens k/H_k Fehler produzieren und damit wieder die deterministischen schlagen.

Zur Erinnerung: Sobald wir die Größe des Caches festlegen und damit ein konkretes k haben, können wir sofort H_k und k/H_k ausrechnen.

Damit ist der zweite Teil abgeschlossen. Im dritten Teil werden adaptive Online-Gegner genauer behandelt. Dazu werden das gewichtete Paging und die k-Server betrachtet.