Kompetitive Analysen von Online-Algorithmen

jonas echterhoff

16. Juli 2004

1 Einführung

1.1 Terminologie

Online-Algorithmen sind Algorithmen, die Probleme lösen sollen, bei denen Entscheidungen getroffen werden müssen, bevor alle Informationen auf denen diese Entscheidungen basieren verfügbar sind. Typische Beispiele für solche Probleme in der Informatik sind Roboter-Bewegungsplanung, Datenkompression und Paging-Systeme (auf die hier später noch genauer eingegangen wird).
Im allgemeinen lässt sich ein Online-Problem wie folgt beschreiben: Einem Online-Algorithmus A wird eine Folge von Anfragen $\sigma = \sigma(1), \sigma(2), \ldots, \sigma(m)$ vorgelegt, die er *online* abarbeiten muss, d.h. wenn $\sigma(t)$ vorgelegt wird, sind dem Algoritmus noch keine $\sigma(t')$ mit $t' > t$ bekannt. Das abarbeiten von Anfragen erfordet Kosten, und Zielsetzung soll es sein, die Gesamtkosten für die Folge σ zu minimieren.
In diesem Vortrag soll nun das Verfahren zur *kompetitiven Analyse* von Online-Algorithmen vorgestellt werden. Dabei wird ein Online-Algorithmus A gegen einen *optimalen Offline-Algorithmus* verglichen. Ein *optimaler Offline-Algorithmus* ist dabei ein Algorithmus, der die vollständige Anfragefolge σ kennt, und diese mit minimalen Kosten abarbeiten kann. Sein nun $C_A(\sigma)$ die Kosten die A benötigt um σ abzuarbeiten, und $C_{OPT}(\sigma)$ die Kosten die der optimale Offline-Algorithmus OPT benötigt. Man bezeichnet nun A als *c-kompetitiv*, wenn eine Konstante a existiert, so dass für alle Anfragefolgen σ gilt: $C_A(\sigma) \leqslant c \cdot C_{OPT}(\sigma) + a$.

1.2 Beispiel: das Paging-Problem

Wir betrachten hier das Paging-Problem, wie es z.B. bei virtueller Speicherverwaltung und Cache-Architekturen vorkommt. Gegeben sei eine Speicherhierachie mit zwei Ebenen, also einen kleinen, schnellen Cache-Speicher und einen grossen, langsamen Hintergrundspeicher. Wenn ein Speicherblock angefordert wird, muss der Block im Cache-Speicher liegen um abgearbeitet zu werden. Wenn nicht muss der Block geladen werden, und es tritt ein *page fault* auf. Eine Paging-Algorithmus muss nun entscheiden welcher Block aus dem Cache-Speicher verworfen werden kann. Ein Online-Algoritmus weiß dabei jedoch nicht, welche Blöcke als nächstes angefragt werden. Als Kosten be-

trachten wir hier die Gesamtanzahl von page faults für eine Anfragefolge.
Wir betrachten hier drei bekannte Paging-Algorithmen:

- **LRU** (Least Recently Used): Den Block verwerfen, der am längsten nicht mehr angefragt wurde.
- **FIFO** (First-In First-Out): Den Block verwerfen, der am längsten im Cache-Speicher lag.
- **LFU** (Least Frequently Used): Den Block verwerfen, der am wenigsten häufig angefragt wurde.

Als Vergleich betrachten wird den offensichtlich optimalen Offline-Algorithmus OPT:

- **MIN**: Den Block verwerfen, der am spätesten in der Zukunft wieder angefragt wird.

Satz: LFU ist nicht kompetitiv.

Beweis: Gehen wir einfachhaltshalber von einem Cache-Speicher der Grösse 2 aus, der zu Beginn die Blöcke $\{1, 2\}$ enthält. Dann gibt es zu jedem c eine Anfragesequenz, so dass LFU c page faults hat, und OPT genau einen: Zuerst wird $c/2$ mal Block 1 aufgerufen, danach $c/2$ mal abwechselnd Block 2 und Block 3.

Sei im folgenden k die Grösse des Cache-Speichers.

Satz: LRU und FIFO sind k-kompetitiv.

Beweis: Gehen wir oBdA davon aus, dass bei LRU und OPT zu Beginn diesselben Daten im Cache-Speicher liegen. Wir zerteilen nun σ in Phasen $P(0), P(1), P(2), \ldots$, so dass LRU höchstens k page faults in $P(0)$ und genau k page faults in $P(i), i \geqslant 1$ hat, indem wir immer nach k page faults einen neue Phase beginnen. Zu einer Phase $P(i)$ sein $\sigma(t_i)$ die erste und $\sigma(t_{i+1} - 1)$ die letzte Anfrage.
Nun ist noch zu zeigen, dass OPT in jeder Phase mindestens einen page fault hat. Für $P(0)$ ist dies offensichtlich: da OPT und LRU zu Beginn den selben Cache-Inhalt haben haben OPT und LRU zur gleichen Zeit den ersten page fault.
Sei p der letzte Block, der in einer Phase $P(i-1), i \geqslant 1$ angefordert wurde. Wenn nun $P(i)$ Anfragen zu k verschiedenen Blöcken ungleich p enthält, dann muss OPT mindestens einen page fault haben, und die Kompetitivität ist gezeigt. Dies ist eindeutig der Fall, wenn die k Anfragen zu denen LRU einen page fault hat alle zu verschiedenen Blöcken ungleich p sind. Was passiert aber wenn LRU in der Phase zweimal zu demselben Block q einen page fault hat, wenn es also s_1, s_2 gibt mit $\sigma(s_1) = \sigma(s_2) = q$ und $t_i \leqslant s_1 < s_2 \leqslant t_{i+1} - 1$? q wird also zu einem Zeitpunkt t mit $s_1 < t < s_2$ verdrängt, ist also der am längsten nicht mehr angefragte Block im Cache. Also enthält die Teilfolge

$\sigma(s_1), \ldots, \sigma(t)$ Anfragen nach $k+1$ verschiedenen Blöcken, von denen mindestens k ungleich p sein müssen.
Also hat OPT mindestens einen page fault und somit ist LRU k-kompetitiv.

Der Beweis für FIFO ist sehr ähnlich.

1.2.1 Untere Abschätzung zum Paging-Problem für deterministische Online-Algorithmen

Satz: Sei A ein deterministischer Online-Algorithmus für das Paging-Problem. Wenn A c-kompetitiv ist, so ist $c \geqslant k$.

Beweis: Sei $S = \{p_1, p_2, \ldots, p_{k+1}\}$ eine Menge von $k+1$ verschiedenen Blöcken. Gehen wir oBdA davon aus, dass bei A und OPT zu Beginn $p_,1 \ldots, p_k$ im Cache liegen. Betrachten wir nun die Anfragefolge, in der in jeder Anfrage der Block aus S angefordert wird, der nicht im Cache von A liegt. Also hat A bei jeder Anfrage einen page fault. Angenommen OPT hat bei $\sigma(t)$ ein page fault. Dann wird OPT den Block verdrängen, der in $\sigma(t+1), \ldots, \sigma(t+k-1)$ nicht angefragt wird. Also hat OPT für k beliebige aufeinanderfolgende Anfragen höchstens einen page fault.

Der Kompetitivitätsfaktor c ist für deterministische Paging-Algorithmen nicht unbedingt besonders Aussagekräftig. Wie gezeigt wurde steigt c zwangsläufig für grössere Cache-Speicher. In der Realität arbeitet ein Paging-Algoritmus aber um so effiziente, desto grösser der Speicher ist. Ausserdem arbeitet LRU in der Realität wesentlich effizienter als FIFO, hat aber denselben Kompetitivitätsfaktor.

2 Randomisierte Online-Algorithmen

2.1 Terminologie

Der Kompetitivitätsfaktor für eine randomisierten Online-Algorithmus A wird im Verhältniss zu einem *Gegenspieler* der die Anfragefolge σ erzeugt und auch abarbeiten muss definiert. Die Frage ist nun, wieviel der Gegner über die Zufallsentscheidungen die A trifft weiss. Es werden drei Arten von Gegnern definiert:

- **Blinder Gegenspieler** Ein blinder Gegner erzeugt σ komplett im Vorraus. Dem Gegner werden die Kosten des optimalen Offline-Algorithmus angerechnet.
- **Adaptiver Online-Gegenspieler** Ein adaptiver Online-Gegner kann die Reaktionen von A beobachten, und kann eine neue Anfrage basierend auf dem vorhergehenden Verhalten von A stellen. Der Gegner muss seine Anfragen online abarbeiten, also ohne zu das Verhalten von A in der Zukunft zu kennen.

- **Adaptiver Offline-Gegenspieler** Wie ein adaptiver Online-Gegner. Dem Gegner werden jedoch die Kosten des optimalen Offline-Algorithmus angerechnet.

A heisst c-kompetitiv gegen einen Blinden Gegenspieler, wenn eine Konstante a existiert, so dass $E[C_A(\sigma)] \leqslant c \cdot C_{OPT}(\sigma) + a$, wobei $E[C_A(\sigma)]$ die zu erwartenden Kosten von A mit Eingabe σ sind.

A heisst c-kompetitiv gegen einen adaptiven Gegenspieler, wenn eine Konstante a existiert, so dass $E[C_A] \leqslant c \cdot E[C_{ADV}] + a$, wobei $E[C_A]$ und $E[C_{ADV}]$ die zu erwartenden Kosten von A und dem Gegenspieler sind.

> **Satz:** Wenn es zu einem Problem einen c-kompetitiven randomisierten Online-Algorithmus gegen jeden adaptiven Offline-Gegenspieler gibt, dann gibt es auch einen c-kompetitiven deterministischen Algorithmus.

> **Satz:** Wenn A zu einem Problem ein c-kompetitiven randomisierter Online-Algorithmus gegen einen adaptiven Online-Gegenspieler ist und es einen d-kompetitiven Online-Algorithmus gegen einen blinden Gegner gibt, dann ist A $(c \cdot d)$-kompetitiv gegen jeden adaptiven Offline-Gegenspieler.

Aus den beiden Sätzen folgt, dass wenn es zu einem Problem einen c-kompetitiven randomisierten Online-Algorithmus gegen jeden adaptiven Online-Gegenspieler gibt, dann gibt es auch einen c^2-kompetitiven deterministischen Algorithmus.

2.2 Beispiel: das Paging-Problem

Betracheten wir nun folgenden randomisierten Algorithmus für das Paging-Problem:

- **MARKING**: Eine Anfragesequenz wird in Phasen bearbeitet. Zu Beginn jeder Phase sind alle Speicherblöcke unmarkiert. Wenn ein Block angefragt wird, wird er markiert. Bei einem page fault wird per Zufall eine der unmarkierten Blöcke verdrängt. Eine Phase endet, wenn alle Blöcke markiert sind, dann werden alle Markierungen gelöscht und eine neue Phase beginnt.

> **Satz:** MARKING ist $2H_k$-kompetitiv gegen jeden blinden Gegner, wobei $H_k = \sum_{i=1}^{k} \frac{1}{i} \approx ln\ k$ die k-te harmonische Zahl ist.

> **Beweis:** Nehmen wir oBdA an, dass MARKING schon in der ersten Anfrage $\sigma(1)$ einen page fault hat. Wir bezeichnen nun einen Block der unmarkiert ist, aber in der letzten Phase markiert wurde als *stale*. Ein Block der weder markiert noch stale ist bezeichnen wir als *clean*. Sei nun c die anzahl der cleanen Blöcke, die in einer Phase angefordert werden und S_{OPT} und S_M die Inhalte der Cache-Speichers von OPT bzw. MARKING.

Sein ausserdem d_I bzw. d_F der Wert von $|S_{OPT} \backslash S_M|$ zu Beginn und Ende einer Phase. OPT hat wärend dieser Phase mindestens $c - d_I$ page faults, weil mindestens $c - d_I$ der c cleanen Blöcke nicht im Cache-Speicher von OPT sind. Weiterhin hat OPT mindestens d_F page faults wärende der Phase, da d_F der angeforderten Blöcke zum Ende der Phase nicht im Cache-Speicher von OPT liegen und, da sie im Speicher von MARKING liegen, offensichtlich verdrängt wurden.
Also hat OPT in einer Phase mindestens $max(c - d_I, d_F) \geqslant \frac{1}{2}(c - d_I + d_F) = \frac{c}{2} - \frac{d_I}{2} + \frac{d_F}{2}$ page faults. Über alle Phasen gesehen gleichen sich die $\frac{d_I}{2}$'s und $\frac{d_F}{2}$'s gegenseitig aus, ausser für die erste und letzte Phase. Also sind die amortisierten Kosten von OPT pro Phase mindestens $\frac{c}{2}$.
Nun wollen wir die zu erwartenden Kosten von MARKING analysieren. Die c cleanen Blöcke zu bedienen verursacht Kosten von c. Es gibt weiterhin $s = k - c \leqslant k - 1$ Anfragen an stale Blöcke. Für $i = 1, \ldots, s$ berechnen wir die erwarteten Kosten für die i-te Anfrage nach einem stale Block. Sein dann $c(i)$ die Anzahl der cleanen Blöcke, die in der Phase schon angefragt wurden, und $s(i)$ die Anzahl der stale Blöcke, die noch nicht nicht angefragt wurden.
Nun sind noch genau $s(i) - c(i)$ der $s(i)$ stale Blöcke im Cache-Speicher, gleichwarscheinlich verteilt. Somit sind die erwarteten Kosten der Anfrage $\frac{s(i)-c(i)}{s(i)} \cdot 0 + \frac{c(i)}{si} \cdot 1 \leqslant \frac{c}{s(i)} = \frac{c}{k-i+1}$. Dann sind die zu erwartenden Gesamtkosten für eine Phase $c + \sum_{i=1}^{s} \frac{c}{k-i+1} \leqslant c + \sum_{i=2}^{k} \frac{c}{i} = c + c(H_k - 1) = cH_k$.
Nun kann man an den Kosten von MARKING und OPT erkennen, das MARKING $2H_k$-kompetitiv gegen jeden blinden Gegner ist.

2.2.1 Untere Abschätzung zum Paging-Problem für randomisierte Online-Algorithmen

Satz: Ist R ein randomisierter Online-Algorithmus für das Paging-Problem, der c-kompetitiv gegen jeden blinden Gegenspieler ist, so ist $c \geqslant H_k$.

.