Seminar Randomisierte Algorithmen
WiSe 2002/2003

Online Algorithmen (III) - Adaptive Online-Gegner

Von Oliver Spata

Inhalt

- 1 Einleitung/Wiederholung
- 2 Gewichtetes Paging
- 3 Das k -Server Problem
- 4 Literatur

1 Einleitung/Wiederholung

In den vorangehenden Vorträgen zum Themenbereich Online Algorithmen wurden das Paging-Problem, die verschiedenen Typen von Gegenspielern und der Begriff des Kompetitivitätskoeffizienten eingeführt und besprochen. Zentral ist hierbei die Frage, inwieweit sich mithilfe von Randomisierung die erwarteten Kosten eines Algorithmus verbessern lassen. Die wichtigsten Begriffe sollen noch einmal kurz wiederholt werden.

Um die Güte von Online-Algorithmen zu messen, verwendet man verschiedene Arten von Gegenspielern. Den zu testenden randomisierten Algorithmus R und seinen Gegenspieler lässt man dieselbe Anfragesequenz bearbeiten, woraufhin die dabei entstandenen Kosten verglichen werden. Sogenannte Kompetitivitätskoeffizienten C stellen ein Verhältnis zwischen diesen Kosten her. Man verwendet sie, wenn erwiesen ist, dass der Aufwand des Algorithmus R mindestens um den Faktor C höher liegt als der Aufwand, der dem Gegenspieler entsteht.

Der unwissende Gegenspieler (oblivious adversary) ist der einfachste Gegner, er erzeugt die Anfragesequenz vollständig im Voraus, um sie dann dem Algorithmus zu übergeben. Die Adaptiven Gegner erzeugen demgegenüber die Sequenz in Reaktion auf das Verhalten von R während dessen Laufzeit. Handelt es sich um einen Adaptiven Online-Gegner, so beantwortet dieser gleichzeitig online dieselbe Sequenz. Noch mächtiger ist der Adaptive Offline-Gegner, welcher die entstandene Anfrage nachher mit einem optimalen Offline-Algorithmus bearbeiten kann. Es spiegelt sich im zugehörigen Koeffizienten C wieder, wie mächtig ein Gegenspieler gegenüber einem Algorithmus R ist. Ein stärkerer Gegner erzeugt einen größeren Koeffizienten, denn er zwingt R höhere Kosten auf. Im weiteren soll es um Adaptive Online-Gegner bzw. deren Kompetitivitätskoeffizienten C^{aon} gehen:

$$E[f_R] - C_R^{aon} \cdot E[f_{ADV}] \leq b,$$

wobei f_R Kosten von R , f_{ADV} Kosten des Gegenspielers und b additive Konstante (unabhängig von der Länge der Abfragesequenz). Zu beachten ist, dass beide Kostenbegriffe durch Erwartungswerte dargestellt sind, denn das Verhalten von R ist randomisiert, und da die Anfrage adaptiv während der Laufzeit von R entsteht, unterliegen auch die Kosten des Gegenspielers dieser Randomisierung.

Welche Aussagen lassen sich für den Kompetitivitätskoeffizienten C^{aon} machen? Hierfür werden wir Verallgemeinerungen des bisherigen Paging kennen lernen, zunächst das gewichtete Paging und dann das k -Server Problem.

2 Gewichtetes Paging

Jedem Item x wird ein zugehöriges Gewicht $w(x)$ zugeordnet. Dieses Gewicht gibt die entstehenden Kosten an, wenn dieses Item in den Cache geholt werden muss. Dementsprechend betragen die Gesamtkosten beim Paging $\sum_i w(x_i)$. Setzt man die Gewichte

aller Items einheitlich auf eins, so erhält man wieder das Szenario des Pagings ohne Gewichte. Das Gewichtete Paging stellt damit eine Verallgemeinerung des bisherigen Pagings dar. Schnell wird klar, dass bisherige Paging-Algorithmen wie z.B. FIFO nicht hierfür angemessen sind, denn das Ziel beim Paging mit Gewichten sollte es sein, „schwere“ Items eher im Cache zu behalten und „leichtere“ schneller wieder zu entfernen, um somit die hohen Kosten dieser schweren Items nicht erneut tragen zu müssen. Ein Paging-Algorithmus sollte also bei der Entscheidung, welches Item den Cache zu verlassen hat, auf die Gewichte der Items Rücksicht nehmen.

Der **Reciprocal**-Algorithmus leistet dies, indem er jedem im Cache befindlichen Item eine Wahrscheinlichkeit zuordnet, mit der dieses Item entfernt wird. Die Wahrscheinlichkeit p_i wird definiert durch:

$$p_i = \frac{1/w(x_i)}{\sum_{j=1}^k 1/w(x_j)}$$

Der Name des Algorithmus weist darauf hin, dass sich die einem Item zugewiesene Wahrscheinlichkeit gegenläufig zum Gewicht des Items verhält. Es werden also mit höchster Wahrscheinlichkeit die Items mit dem geringsten Gewicht aus dem Cache entfernt. Für diesen Algorithmus gilt das

Theorem: Reciprocal ist k -kompetitiv gegenüber adaptiven Online-Gegnern.

Beweis: Für den Beweis verwendet man die sogenannte Potentialfunktion, die den aktuellen Unterschied zwischen dem Cache von R (Reciprocal) und dem Gegenspieler misst. Dies sei an einem Beispiel verdeutlicht: Zu einem Zeitpunkt sei

$\{a,b,c,d,e\}$ der Cacheinhalt S^R von R ,

$\{a,b,c,r,s\}$ der Cacheinhalt S^{ADV} des Gegenspielers,

$\{d,e\}$ ergibt sich daraus als die Differenzmenge $S^R \setminus S^{ADV}$.

Die Potentialfunktion hat nun die Form

$$\Phi_i = \sum_{x \in S_i^R} w(x) - k \cdot \sum_{x \in S_i^R \setminus S_i^{ADV}} w(x).$$

Untersucht werden soll die Veränderung in der Potentialfunktion nach jedem Schritt i der Anfragesequenz. Hierfür definiert man $\Delta\Phi_i = \Phi_i - \Phi_{i-1}$. Um die Veränderung der Potentialfunktion mit den entstehenden Kosten im i -ten Schritt gegenüberzustellen, wird noch definiert

$$X_i = f_i^R - k \cdot f_i^{ADV} - \Delta\Phi_i.$$

Hierbei werden die Kosten wie weiter oben bereits mit f bezeichnet. Die Potentialfunktion kann sich nur verändern, wenn einer der beiden Konkurrenten einen Fehler begeht, wenn also Elemente im Cache ausgetauscht werden.

Wir wollen den Fall untersuchen, dass R im i -ten Schritt einen Fehler begeht. In diesem Fall muss gelten, dass $|S^R \setminus S^{ADV}| \geq 1$, im Cache von R ist also mindestens ein Item, welches im Cache des Gegenspielers fehlt. Denn wenn die Differenzmenge leer wäre, würden sich beide Caches exakt gleichen, welches der Annahme widerspricht, dass nur R einen Fehler begeht.

Die Frage ist also: Wie sieht X_i aus? Es ist klar, dass R ein neues Item x in den Cache holen muss, was bedeutet dass seine Kosten $f_i^R = w(x)$ betragen. Gleichzeitig entstehen keinerlei Kosten für den Gegenspieler, also $f_i^{ADV} = 0$.

Wichtig ist jetzt der Wert von $\Delta\Phi_i$. Er wird im folgenden durch den Erwartungswert $E[\Delta\Phi_i]$ formal ausgedrückt. Betrachtet man die Definition der Potentialfunktion, so wird klar dass das Gesamtgewicht des Caches von S^R auf jeden Fall um das Gewicht des neuen Items $w(x)$ erhöht wird. Gleichzeitig wird das Gesamtgewicht um das Gewicht des entfernten Items x' verringert. Zum Schluss muss noch berücksichtigt werden, dass das entfernte Item aus der Menge $S^R \setminus S^{ADV}$ stammen könnte. Zunächst ermitteln wir den Erwartungswert für das Gewicht des Items x' , welches von Reciprocal entfernt wird:

$$E[w(x')] = \sum_{i=1}^k w(x_i) \cdot p_i = \sum_{i=1}^k w(x_i) \cdot \frac{1/w(x_i)}{\sum_{j=1}^k 1/w(x_j)} = \frac{k}{\sum_{j=1}^k 1/w(x_j)}$$

Man sieht, dass durch die Definition von p_i bei Reciprocal die Summationsvariable verschwindet. Die Summe enthält k viele identische Terme.

Durch dieselbe Argumentation ergibt sich für den Fall, dass x' aus der Differenzmenge $S^R \setminus S^{ADV}$ stammt, eine erwartete Veränderung der Potentialfunktion in Höhe von

$$E[x' \in S^R \setminus S^{ADV}] = \frac{|S^R \setminus S^{ADV}| \cdot k}{\sum_{j=1}^k 1/w(x_j)}$$

Der Faktor k im Zähler ist der k -fach höheren Gewichtung der Elemente, die ausschließlich im Cache von R liegen, innerhalb der Potentialfunktion entsprungen. Damit ergibt sich für $E[\Delta\Phi_i]$ also:

$$E[\Delta\Phi_i] = w(x) - \frac{k}{\sum_{j=1}^k 1/w(x_j)} + \frac{|S^R \setminus S^{ADV}| \cdot k}{\sum_{j=1}^k 1/w(x_j)} \geq w(x).$$

Daraus folgt die Abschätzung $E[X_i] \leq 0$, und damit ebenso $E[\sum X_i] \leq 0$. Weiterhin verschwinden bei der Aufsummierung aller X_i fast alle Φ_i (Teleskopsumme). Übrig bleiben einzig die Terme Φ_0 und Φ_n , welche Ausdrücke für die Unterschiede zwischen den Caches vor und nach Bearbeitung der Anfragesequenz sind. Diese sind abhängig von den Gewichten der Items in diesen Caches, nicht jedoch abhängig von der Länge der bearbeiteten Sequenz. Somit kann der Einfluss dieser Terme unter der additiven Konstante b in der Definition der Kompetitivität verbucht werden. Damit bleibt festzustellen, dass eine Schranke für

$$\sum_i (E[f_i^R] - k \cdot E[f_i^{ADV}])$$

existiert. Und dies beweist die Aussage des Theorems. □

Wenn man den Reciprocal auf das Paging ohne Gewichte überträgt, so entsteht daraus der **Random**-Algorithmus, welcher durch simples „Würfeln“ über das Entfernen eines Items aus dem Cache entscheidet. Auch Random ist k -kompetitiv gegenüber adaptiven Online-Gegnern. Lässt sich für das Paging ein besseres Ergebnis erzielen, oder ist k der bestmögliche Kompetitivitätskoeffizient? Antwort liefert eine weitere Verallgemeinerung.

3 Das k -Server Problem

Die Umgebung eines k -Server Problems ist ein Maßraum, der sich aus Punkten zusammensetzt. Es existiert also ein Abstandsmaß für die Punkte. Ein Algorithmus verwaltet k Server, und jeder Server besetzt einen Punkt des Raumes (dabei nicht mehr als ein Server gleichzeitig auf einem Punkt). Falls ein Server von einem Punkt u zu einem anderen Punkt v bewegt wird, entstehen dadurch Kosten c_{uv} , welche der Entfernung zwischen u und v entsprechen.

Das Paging kann man als Spezialfall oder anders ausgedrückt als eine Anwendung des k -Server Problems betrachten (es lassen sich auch verschiedene andere Problemstellungen durch ein k -Server Problem formulieren). Für das Paging entsprechen die k Server also den k Speicherzellen des Caches. Wenn ein Server einen Punkt besetzt, so ist dieser Punkt Bestandteil des momentanen Caches. Um ein Item in den Cache zu holen, muss dementsprechend ein Server bewegt werden. Beim Paging ohne Gewichte ist die Entfernung zwischen allen Punkten des Raumes einheitlich eins, während beim gewichteten Paging ein Punkt zu allen übrigen Punkten des Raumes denselben Abstand besitzt, und zwar den seines Gewichtes. Vorstellen kann man sich dies anhand eines Graphen, in dem alle Kanten zu einem bestimmten Punkt dieselbe Länge haben. Dies bedeutet, dass es keine Rolle spielt, welchen Server man auf einen Punkt bewegt, da für alle Server dieselbe Entfernung gilt.

Die unterschiedlichen Gegner und die dazugehörigen Kompetitivitätskoeffizienten lassen sich für das k -Server Problem so definieren wie vorher (für die Kosten stehe ab jetzt M):

$$M_R - C_R^{aon} \cdot M_{ADV} \leq b$$

Theorem: R sei ein randomisierter Algorithmus, welcher k Server verwaltet. Dann gilt: R ist k -kompetitiv gegenüber adaptiven Online-Gegnern.

Beweis: Zunächst wird für den Beweis gezeigt, welche Kosten einem (beliebigen) Algorithmus R aufgezwungen werden. Hierfür definiert man die Menge H , welche alle Punkte enthält, die zu Beginn von R abgedeckt werden, plus einem weiteren beliebigen Punkt, also $|H| = k + 1$. Der adaptive Gegner sorgt dafür, dass bei jedem Schritt der Anfragesequenz

ρ_i der Punkt aus H verlangt wird, welcher nicht durch R abgedeckt ist. Dies bedeutet, dass im i -ten Schritt für das abgefragte ρ_i ein Server bewegt werden muss. Der freigewordene Punkt wird dann im nächsten Schritt zu ρ_{i+1} . Dadurch entstehen bei jedem Schritt i Kosten für R in Höhe von $c_{\rho_{i+1}\rho_i}$. Die sicheren Gesamtkosten für R lassen sich dann ausdrücken durch

$$M_R(\rho_1, \rho_2, \dots, \rho_N) = \sum_{i=1}^N c_{\rho_{i+1}\rho_i} = \sum_{i=1}^N c_{\rho_i\rho_{i+1}}.$$

Um die Behauptung des Theorems zu zeigen, konstruiert man k verschiedene Online-Gegner B_j , deren summierte Kosten in der Höhe von M_R liegen. Die B_j bedecken in der Ausgangslage dieselben Punkte wie R , mit einer Ausnahme: Jeder B_j lässt genau einen Punkt aus der Startkonfiguration von R aus, und jeder B_j einen anderen. Den dadurch freigewordenen Server setzt jeder B_j auf den Punkt aus der Menge H , der nicht zur Konfiguration von R zählt. Dieser Punkt wird auch das erste Element ρ_1 der Anfrage sein.

Diese erste Anfrage bereitet noch keinem der k Online-Gegner Kosten (im Gegensatz zu R). Im folgenden aber bewegt ein B_j , falls er im i -ten Schritt einen Fehler begeht, den Server von ρ_{i-1} auf ρ_i . Durch diese Vorschrift wird sichergestellt, dass sich alle B_j zu jedem Zeitpunkt voneinander unterscheiden: Zu Beginn ist dies per Konstruktion so. Gilt dies für alle Konfigurationen vor dem i -ten Schritt und ein B_j begeht einen Fehler, so ist der Punkt ρ_{i-1} nach dem i -ten Schritt nicht mehr in der Konfiguration des B_j . Da aber alle anderen Online-Gegner keinen Fehler begangen haben, decken sie noch alle mit einem Server den Punkt ρ_{i-1} ab. Also unterscheidet sich B_j auch nach diesem Schritt weiterhin von allen übrigen Gegnern. Und weil sich die übrigen $k-1$ Gegner vor Schritt i untereinander unterschieden und keiner von ihnen handeln musste, unterscheiden sie sich auch nach Schritt i noch.

Dies führt zu der Aussage, dass pro Schritt i (mit Ausnahme des ersten Schrittes, der einen Sonderfall darstellt) immer genau ein B_j einen Fehler begeht, denn nur genau einem B_j kann der angefragte Punkt ρ_i fehlen. Zu jedem Fehler von R gibt es also einen zugehörigen Fehler eines der B_j , daher gilt

$$\sum_{j=1}^k M_{B_j}(\rho_1, \rho_2, \dots, \rho_N) \leq M_R(\rho_1, \rho_2, \dots, \rho_N).$$

Man kann nicht genau sagen, welche exakten Kosten für einen der B_j entstehen, aber es lässt sich auf jeden Fall mindestens ein Online-Gegner finden, dessen Kosten mindestens um den Faktor k geringer ausfallen als die Kosten von R . $\square$

Da dieses Ergebnis für die allgemeinere k -Server Umgebung gilt, lässt sich schlussfolgern: Der Reciprocal-Algorithmus ist für gewichtetes Paging optimal!

Bei unwissenden Gegenspielern war der Kompetitivitätskoeffizient H_k noch geringer. Dass sich bei adaptiven Gegnern keine solch geringe Schranke zeigen lässt, ist ein Ausdruck dafür,

wie mächtig diese Art von Gegenspieler ist (bei adaptiven Offline-Gegnern dürfte ein noch schlechteres Ergebnis zu erwarten sein).

4 Literatur

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