Kapitel 7: Online Algorithms

Notation

Online Problem:

- Statt einer vollständig gegebenen Eingabe I haben wir jetzt eine Eingabesequenz $\sigma=(\sigma(1), \sigma(2),\ldots, \sigma(t))$, in der die Eingabeteile $\sigma(i)$ erst nach und nach an den Online Algorithmus übergeben werden.
- Nach jedem $\sigma(i)$ muss der Online Algorithmus eine Entscheidung treffen, bevor er $\sigma(i+1)\ldots\sigma(t)$ gesehen hat. Diese Entscheidung kann (im Standard-Online-Modell) nicht wieder zurück-genommen werden.

Notation

Definition: Sei Π ein Minimierungsproblem und A ein Online Algorithmus für Π. Für eine beliebige Eingabesequenz $\sigma=(\sigma(1), \sigma(2),\ldots, \sigma(t))$ seien $A(\sigma)$ die Kosten von A für σ.

A heißt c-kompetitiv, wenn es einen von t unabhängigen Parameter a gibt, so dass für alle Eingabesequenzen σ gilt:

$$A(\sigma) \leq c \cdot OPT(\sigma) + a$$

Kapitel 7: Online Algorithms

Inhalt:

- Deterministische Online Algorithmen
  - Einführung
  - Paging
  - Scheduling
- Randomisierte Online Algorithmen
  - Paging
  - Finanzielle Spiele

Paging

Problem: Eine online eintreffende Folge von Seitenanfragen $\sigma=(x_1,\ldots,x_n)$ muss durch ein zweistufiges Speichersystem (Cache und Hauptspeicher) bearbeitet werden. Ist eine Seite nicht im Cache vorhanden (Cache-Miss), muss sie vom Hauptspeicher dorthin geladen werden (und verdrängt bei vollem Cache eine dort vorhandene Seite). Minimiere die Anzahl der Cache-Misses.

Offline Fall: alle Seitenanfragen im vorhinein gegeben.

Offline-Algorithmus: MIN

Entferne stets diejenige Seite aus dem Cache, deren nächste Anfrage am weitesten in der Zukunft liegt.

Paging

Satz: MIN ist ein optimaler Offline-Algorithmus, d.h. er erzeugt für jede Anfragesequenz σ die minimale Anzahl an Seitenfehlern.

Beweis:

- σ: beliebige Anfragefolge
- A: optimaler Algorithmus auf σ
- Wir bauen A so um, dass er wie MIN arbeitet. Die Anzahl der Seitenersetzungen wird dabei nicht erhöht.

Paging

Beweis (Fortsetzung):

- Behauptung: angenommen, A und MIN arbeiten auf den ersten i-1 Anfragen identisch, aber auf der i-ten Anfrage verschieden. Dann kann A in Algorithmus A´ transformiert werden, so dass gilt:
  - A´ und MIN arbeiten auf der ersten i Anfragen identisch
  - $A'(\sigma) \leq A(\sigma)$
- Ist diese Behauptung gezeigt, dann können wir A schrittweise in MIN umformen, ohne die Kosten zu erhöhen, und damit wäre der Satz bewiesen.

Paging

Beweis der Behauptung:

- Die i-te Anfrage referenziere die Seite x.

  A entfernt dafür die Seite u aus dem Cache und MIN die Seite v.
- Algorithmus A´: arbeitet auf den ersten i-1 Anfragen wie A und entfernt bei Anfrage i die Seite v. A´ simuliert dann A, bis eines der folgenden Ereignisse eintrifft:
  - a) Ersetzt A die Seite v, dann ersetzt A´ zum gleichen Zeitpunkt die Seite u.
  - b) u wird angefragt und A ersetzt z. In diesem Fall ersetzt A´ die Seite z durch v.

  v kann in σ nicht vor u angefragt werden, da sonst die MIN-Regel verletzt wird.
- In beiden Fällen sind A und A´ dann wieder in identischen Konfigurationen und A´ arbeitet wie A weiter. Die Anzahl der Seitenersetzungen hat sich dabei nicht erhöht. q.e.d.

Paging

Leider ist MIN ein offline Algorithmus (d.h. er muss die gesamte Anfragefolge σ im vorhinein kennen). Ein oft verwendeter Online Algorithmus ist der folgende:

Algorithmus LRU (Least-Recently-Used):
Entferne stets die Seite, die am längsten nicht mehr benutzt wurde.

Im worst case ist LRU (scheinbar) recht schlecht, wie das folgende Resultat zeigt.

Paging

Satz: Für einen Cache der Größe k ist LRU k-kompetitiv.

Beweis:

- Wir betrachten eine beliebige Anfragefolge σ.
- LRU und OPT starten mit denselben Seiten im Cache.
- Wir zerlegen σ in Phasen P(1), P(2),…, so dass LRU in jeder P(l) mit $l \geq 2$ jeweils exakt k Seitenfehler hat und in der Phase P(1) höchstens einen Seitenfehler.
- Können wir zeigen, dass OPT in jeder dieser Phasen mindestens einen Seitenfehler hat, so folgt der Satz.
- Für Phase P(1) ist das sicherlich wahr, da LRU und OPT anfangs dieselben Seiten haben.

Paging

Beweis (Fortsetzung):

- Seien $p_1,\ldots,p_k$ die Fehlerstellen (= Seitenfehler) von LRU in Phase P(l) mit $l \geq 2$ und q die letzte vorangegangene Anfrage vor Beginn der Phase P(l). Wir unterscheiden die folgenden Fälle:

1. $\forall i \neq j\ p_i \neq p_j$ und $\forall i\ p_i \neq q$: Beim Phasenwechsel hin zu Phase P(l) hat OPT q im Cache und kann somit höchstens $k-1$ der Seiten $p_1,\ldots,p_k$ im Cache haben. Also tritt bei OPT in P(l) mindestens ein Seitenfehler auf.
2. $\exists i \neq j\ p_i = p_j$: Zwischen den Anfragen p_i und p_j müssen mindestens k andere verschiedene Seiten angefragt worden sein, sonst hätte LRU die Seite p_i ja nicht aus dem Cache entfernt. Also gibt es $k+1$ verschiedene Seiten in der Phase P(l) und OPT muss wiederum mindestens einen Seitenfehler haben.
3. $\forall i \neq j\ p_i \neq p_j$, aber $\exists i\ p_i = q$: Analog zur Argumentation bei Punkt 2. müssten zwischen q und p_i mindestens k andere verschiedene Seiten angefragt worden sein. Also muss OPT auch hier wieder mindestens einen Seitenfehler haben. q.e.d.

Paging

Satz: Sei A ein deterministischer online Paging-Algorithmus. Ist A c-kompetitiv, dann ist $c \geq k$.

Beweis:

- Wir geben eine Folge an, die $k+1$ verschiedene Seiten $p_1,\ldots,p_{k+1}$ verwendet. Anfangs seien die Seiten $p_1,\ldots,p_k$ im Cache.
- Der „Gegner" fragt nun stets diejenige Seite an, die bei A gerade nicht im Cache vorhanden ist. Bei A tritt also bei jeder Anfrage ein Seitenfehler auf, d.h. $A(\sigma)=|\sigma|$.
- OPT muss für σ höchstens alle k Anfragen eine Seite ersetzen, da nur $k+1$ Seiten vorhanden sind, und also nach einem Seitenfehler auf jeden Fall die nächsten k Seiten im Speicher sind.
- Damit ist $c=A(\sigma)/OPT(\sigma) \geq |\sigma|/(|\sigma|/k) = k$. q.e.d.

Paging

Bemerkung: Auch wenn LRU im worst case eine schlechte Kompetitivität hat, so gibt es dennoch keinen besseren deterministischen Algorithmus. In der Tat gibt es sogar deterministische Algorithmen mit unbeschränkter Kompetitivität:

Algorithmus MRU (Most-Recently-Used):
Entferne diejenige Seite, die zuletzt gefragt wurde

- Betrachte die Sequenz $\sigma=p_1,\ldots, p_k, p_{k+1}, p_k, p_{k+1},\ldots$
- Bei MRU sind die Seitenfehler unbeschränkt während OPT nur einen Seitenfehler hat.

Kapitel 7: Online Algorithms

Inhalt:

- Deterministische Online Algorithmen
  - Einführung
  - Paging
  - Scheduling
- Randomisierte Online Algorithmen
  - Paging
  - Finanzielle Spiele

Scheduling

Problem: Eine online eintreffende Folge von Jobs $\sigma=(J_1,\ldots,J_n)$ soll auf m identischen Maschinen verteilt werden. Jeder Job J_i sei spezifiziert durch eine Ausgabezeit $t_i \in \mathbb{N}$ und eine Bearbeitungszeit $p_i \in \mathbb{N}$, wobei $t_i < t_{i+1}$ für alle i gilt. Der Online Algorithmus muss nun jeden Job J_i zum Zeitpunkt t_i einer Maschine zuweisen.

Ziel: Minimiere den Makespan, d.h. die Zeit bis der letzte Job bearbeitet ist.

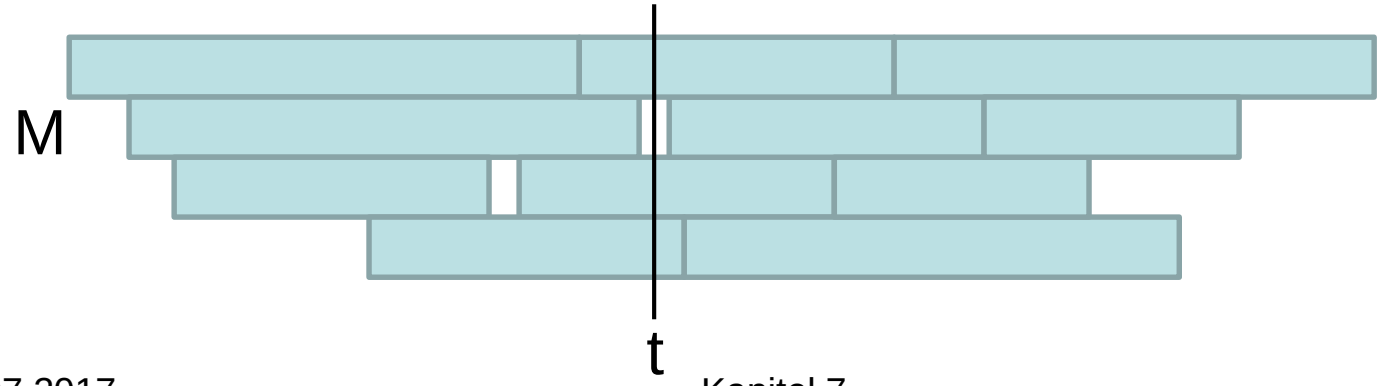
Algorithmus: Greedy-Scheduling
Weise jeden Job der Maschine zu, die zurzeit am wenigsten ausgelastet ist.

Scheduling

Satz: Greedy ist 3-kompetitiv.

Beweis:

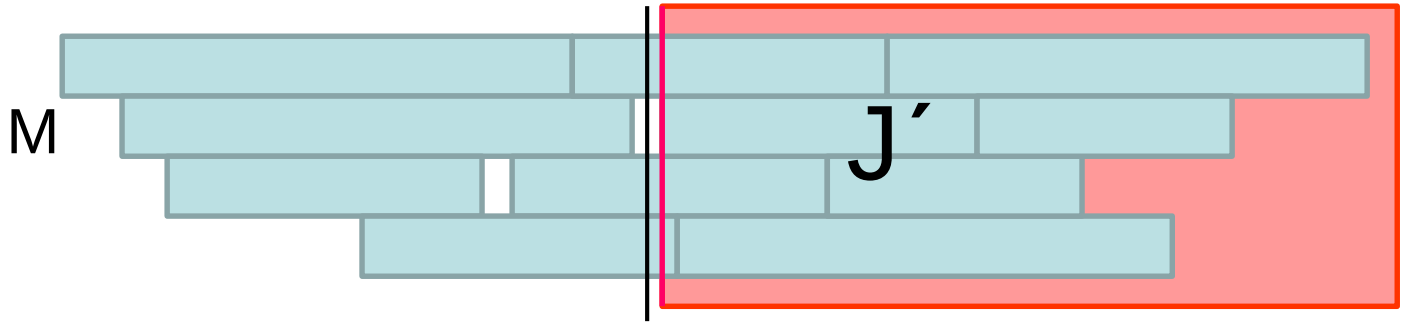
- Sei t der letzte Zeitpunkt, zu dem eine Maschine unbeschäftigt ist, bevor sie bis zum Ende beschäftigt ist, und sei M diese Maschine.
- Für alle Jobs, die vor dem Zeitpunkt t generiert wurden, gilt, dass diese vor t mit der Bearbeitung begonnen haben, denn ansonsten hätte Greedy diese M zugewiesen.



Scheduling

Beweis (Fortsetzung):

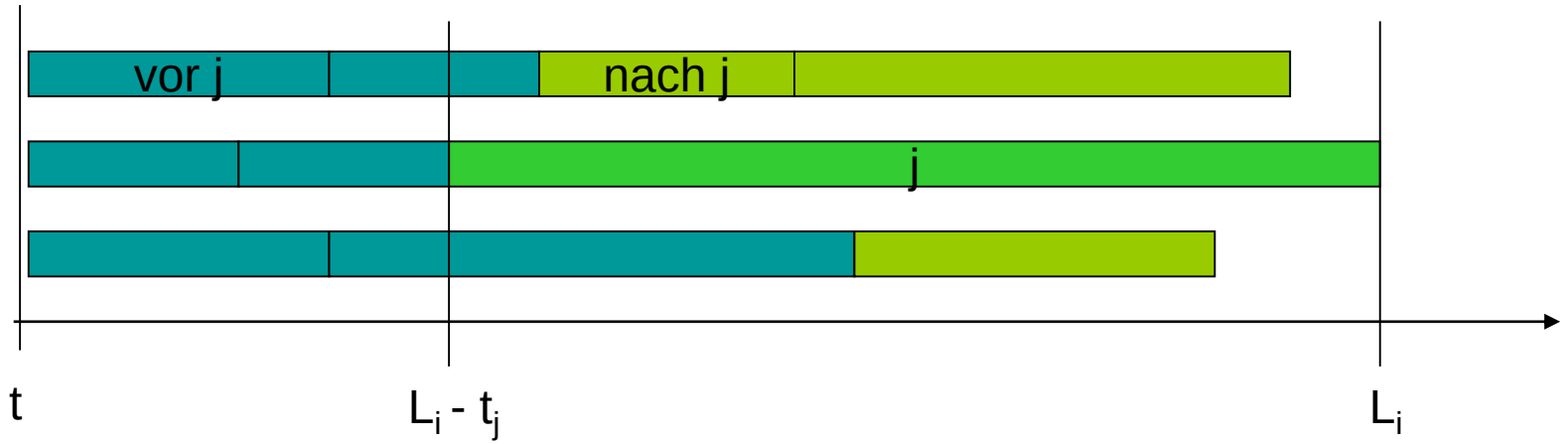
- Sei J die Menge der Jobs, die nach t generiert werden.
- Ergänzen wir J um die nach t noch abzuarbeitenden Reste der Jobs, die vor t generiert worden sind, auf J´, so ergibt sich für diese Jobmenge ein Greedy Schedule wie für den Greedy Approximationsalgorithmus.



Scheduling

Beweis (Fortsetzung):

- J bestehe aus n Jobs mit Aufwand $p_1,\ldots, p_n$
- Betrache Maschine i mit höchster Last L_i.
- Sei j der letzte Job auf Maschine i.
- Da Job j Maschine i zugeordnet wurde, hatte i vorher die kleinste Last. Es gilt also $L_i - p_j \leq L_k$ für alle k.



Scheduling

Beweis (Fortsetzung):

- Es gilt: $L_i - p_j \leq L_k$ für alle $k \in \{1,\ldots,m\}$ und damit $L_k \geq L_i - p_{max}$.
- Wir wissen: $\Sigma_{j=1}^{m} L_j \leq \Sigma_{j=1}^{n} p_j + (m-1)\, p_{max}$ (Alte Jobs vor t)
- Also ist $L_i + (m-1)(L_i - 2p_{max}) \leq \Sigma_{j=1}^{n} p_j$
- Daraus folgt:

$$m \cdot L_i \leq \Sigma_{j=1}^{n} p_j + (m-1) \cdot 2p_{max}$$

$$\Rightarrow \quad L_i \leq (1/m)\, \Sigma_{j=1}^{n} p_j + 2((m-1)/m) \cdot p_{max}$$

$$\leq OPT(\sigma) + 2(1-1/m) \cdot OPT(\sigma)$$

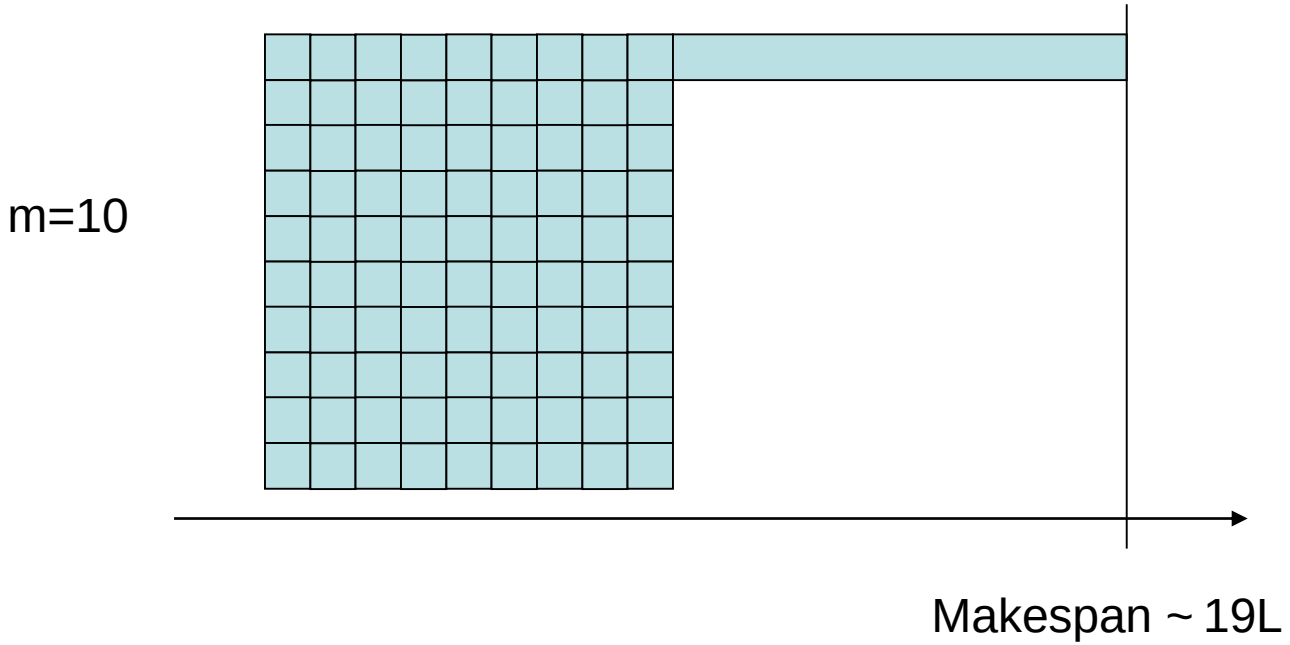
$$\leq 3\, OPT(\sigma)$$

Scheduling

Genauere Analyse: Greedy ist 2-1/m-kompetitiv.

Ist das scharf? Ja!

Beispiel: m Maschinen, m(m-1) Jobs der Länge L<<m(m-1), (so dass Zeit für Genierung der Jobs vernachlässigbar) ein Job der Länge m·L

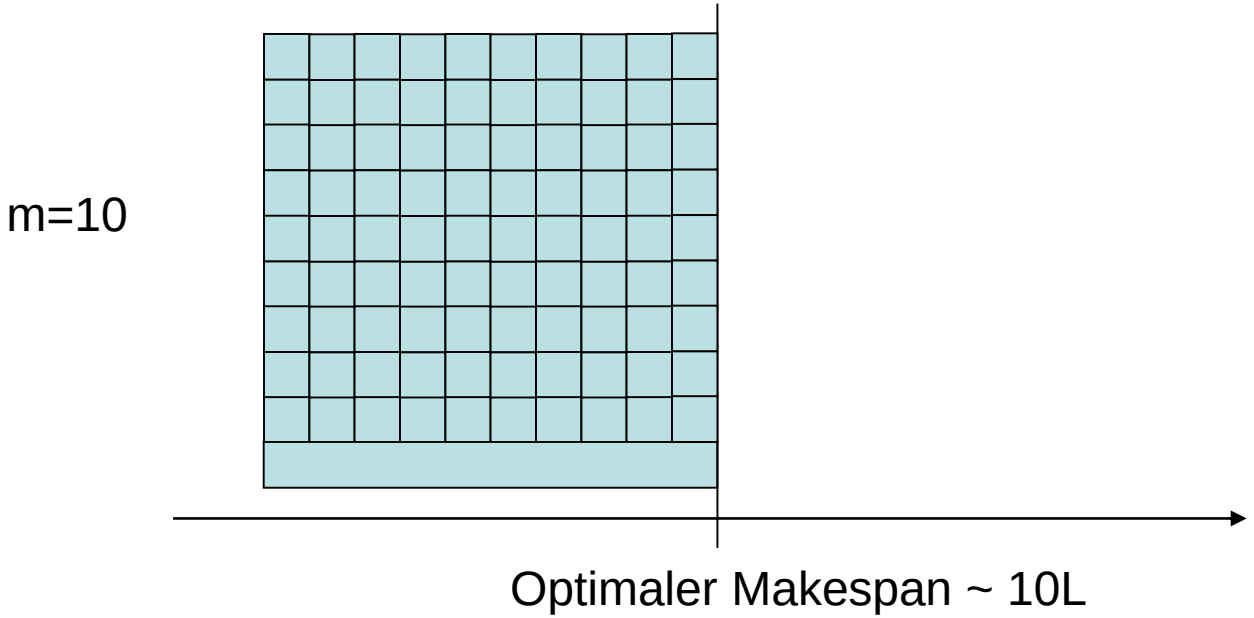


Scheduling

Genauere Analyse: Greedy ist 2-1/m-kompetitiv.

Ist das scharf? Ja!

Beispiel: m Maschinen, m(m-1) Jobs der Länge L<<m(m-1), (so dass Zeit für Genierung der Jobs vernachlässigbar) ein Job der Länge m·L



Kapitel 7: Online Algorithms

Inhalt:

- Deterministische Online Algorithmen
  - Einführung
  - Paging
  - Scheduling
- Randomisierte Online Algorithmen
  - Paging
  - Finanzielle Spiele

Notation

Definition: Sei Π ein Optimierungsproblem und A ein randomisierter Online Algorithmus für Π. Für eine beliebige Eingabesequenz $\sigma=(\sigma(1), \sigma(2),\ldots, \sigma(t))$ seien $E[A(\sigma)]$ die erwarteten Kosten von A für σ.

A heißt c-kompetitiv, wenn es einen von t unabhängigen Parameter a gibt, so dass für alle Eingabesequenzen σ gilt:

$$E[A(\sigma)] \leq c \cdot OPT(\sigma) + a$$

Paging

Problem: Eine online eintreffende Folge von Seitenanfragen $\sigma=(x_1,\ldots,x_n)$ muss durch ein zweistufiges Speichersystem (Cache und Hauptspeicher) bearbeitet werden. Ist eine Seite nicht im Cache vorhanden (Cache-Miss), muss sie vom Hauptspeicher dorthin geladen werden (und verdrängt bei vollem Cache eine dort vorhandene Seite).

Ziel: minimiere die Anzahl der Cache-Misses

Algorithmus RANDOM:

Entferne bei jedem Seitenfehler eine zufällig gemäß der Gleichverteilung gewählte Seite aus dem Cache.

Paging

Satz: RANDOM ist nicht besser als k-kompetitiv im Erwartungswert.

Beweis:

- Wähle für $N>>k$ die folgende Eingabesequenz:

$$\sigma=\underbrace{b_0\ a_1\ldots a_{k-1}}_{P_0}\ \underbrace{(b_1\ a_1\ldots a_{k-1})^N}_{P_1}\underbrace{(b_2\ a_1\ldots a_{k-1})^N}_{P_2}\ldots$$

- In jeder Phase $P_i, i\geq 1$, ist die Wkeit für das Auslagern der Seite b_{i-1} bei jedem Cache-Miss $1/k$.
- Also macht RANDOM in jeder Phase im Erwartungswert k Fehler, bis er sich von b_{i-1} trennt.

Paging

Algorithmus MARKING:

- Die Anfragesequenz σ wird in Phasen bedient. Zu Beginn einer Phase sind alle Seiten unmarkiert.
- Wird eine Seite angefragt, so wird sie markiert.
- Bei einem Seitenfehler wird eine zufällig gleichverteilt ausgewählte unmarkierte Seite entfernt.
- Eine Phase endet unmittelbar vor einem Seitenfehler, wenn alle Seiten im schnellen Speicher markiert sind.

Satz: MARKING ist $2H_k$-kompetitiv, wobei $H_k = \Sigma_{i=1}^{k} 1/i$ die k-te harmonische Zahl ist.

Bemerkung: Es gilt $\ln(k+1) \leq H_k \leq \ln(k) + 1$.

Paging

Algorithmus MARKING:

σ: 7 4 8 3 1 6 4 2

Cache

Hauptspeicher

Paging

Algorithmus MARKING:

σ: 

Cache 

Hauptspeicher 

Paging

Algorithmus MARKING:

σ: 

Cache 

Hauptspeicher 

Paging

Algorithmus MARKING:

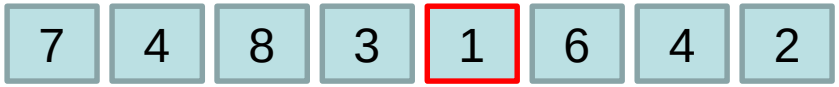
σ: 

Cache 

Hauptspeicher 

Paging

Algorithmus MARKING:

σ: 

Cache 

Hauptspeicher 

Paging

Algorithmus MARKING:

σ: 

Cache 

Hauptspeicher 

Paging

Algorithmus MARKING:

σ: 

Cache 

Hauptspeicher 

Paging

Algorithmus MARKING:

σ: 

Cache 

Hauptspeicher 

Paging

Algorithmus MARKING:

σ: 

Cache 

neue Phase

Hauptspeicher 

Paging

Beweis:

- Sei σ eine beliebige Anfragesequenz.
- MARKING zerlegt σ in Phasen $P_1, P_2, \ldots, P_m$.
- Jede Phase P_i enthält genau k verschiedene Seitenanfragen, und die erste Anfrage von P_i ist verschieden von allen Seitenanfragen in P_{i-1}.
- n_i: Anzahl der neuen Seitenanfragen in P_i (d.h. Seiten, die nicht in P_{i-1} angefragt wurden)
- Wir zeigen, dass OPT amortisierte Kosten von mindestens $n_i/2$ in P_i erzeugt und MARKING mit erwarteten Kosten von $n_i \cdot H_k$ auskommt.
- Dann ergibt sich als Kompetitivitätsfaktor

$$c \leq n_i \cdot H_k / (n_i/2) = 2H_k$$

Paging

Beweis (Fortsetzung):

- Wir zeigen zunächst, dass OPT amortisierte Kosten von mindestens $n_i/2$ pro Phase P_i erzeugt.
- Wir betrachten zwei aufeinanderfolgende Phasen P_i und P_{i+1}.
- P_i und P_{i+1} enthalten zusammen genau $k+n_i$ paarweise verschiedene Seiten.
- OPT macht also in diesem Phasenpaar mindestens n_i Fehler.
- Es gilt also:

$$OPT \geq n_2+n_4+n_6+\ldots = \Sigma_{i \text{ gerade}} n_i$$

sowie

$$OPT \geq n_1+n_3+n_4+\ldots = \Sigma_{i \text{ ungerade}} n_i$$

- Insgesamt erhalten wir als amortisierte Kosten

$$2OPT \geq \Sigma_{i \text{ gerade}} n_i + \Sigma_{i \text{ ungerade}} n_i$$

$$\Rightarrow OPT \geq (1/2) \Sigma_i n_i$$

Paging

Beweis (Fortsetzung):

- MARKING erzeugt in Phase P_i genau n_i Seitenfehler für die Anfragen auf neue Seiten.
- Es kann aber auch Seitenfehler für Seiten geben, die nicht neu in Pi sind!
- Die Anzahl der in P_i angefragten alten Seiten sei o_i. Es gilt $o_i+n_i=k$.
- Wir betrachten die erste Anfrage auf die j-te alte Seite, $j \leq o_i$.
- Unmittelbar vorher enthält der Cache n_i^j neue Seiten, die bisher angefragt worden sind, $n_i^j \leq n_i$.
- Es gibt $k-(j-1)$ alte Seiten, die vor der Anfrage auf die j-te alte Seite noch nicht angefragt wurden. Von diesen alten Seiten sind n_i^j nicht mehr im Cache.
- Die Wahrscheinlichkeit für einen Seitenfehler ist also
$$n_i^j/(k-(j-1)) \leq n_i/(k-(j-1))$$
- Da $n_i \geq 1$ ist, gilt $o_i \leq k-1$ und damit $1/(k-o_i+1) \leq ½$. Als erwartete Kosten für die alten Seiten ergibt sich also
$$\Sigma_{j=1}^{o_i} n_i/(k-(j-1)) \leq n_i(1/k+\ldots+1/2) = n_i(H_k-1)$$
- Die Gesamtkosten in Phase P_i sind also $cost(P_i) \leq n_i+n_i(H_k-1) = n_i \cdot H_k$.

Paging

Satz: Sei A ein randomisierter Online-Paging-Algorithmus, der c-kompetitiv ist. Dann ist $c \geq H_k$.

Beweis:

- Sei A ein Online-Paging-Algorithmus.
- Die Menge der angefragten Seiten sei $\{p_1,\ldots,p_{k+1}\}$.
- Der Gegner verwaltet einen Vektor von Wahrscheinlichkeitswerten $Q=(q_1,\ldots,q_{k+1})$. q_i ist die Wahrscheinlichkeit, dass die Seite p_i nicht im Cache von A ist.
- Es gilt $\Sigma_{i=1}^{k+1} q_i = 1$. Die Einträge im Vektor Q werden vom Gegner in jedem Schritt aktualisiert.
- Wir unterteilen die Anfragesequenz σ in Phasen, die jeweils aus k Subphasen bestehen. Der Gegner markiert die angefragten Seiten wie bei MARKING und unterteilt σ in entsprechende Phasen. Mit jeder neu markierten Seite beginnt eine neue Subphase.
- In jeder Subphase j wollen wir für A erwartete Kosten von $1/(k+1-j)$ erzeugen, so dass sich die Kosten pro Phase für A berechnen zu
  $$\Sigma_{j=1}^{k} 1/(k+1-j) = 1/k + 1/(k-1) + \ldots + 1 = H_k$$
- Der Gegner soll pro Phase Kosten von 1 erzeugen. Dann folgt Satz 2.4.

Paging

Beweis (Fortsetzung):

- Zu Beginn einer Subphase j gibt es j markierte Seiten.
- Sei M die Menge der markierten Seiten. Wir setzen $\gamma = \Sigma_{p_i \in M} q_i$ und unterscheiden zwei Fälle.
- Ist $\gamma=0$, dann gibt es $k+1-j$ unmarkierte Seiten, von denen wir eine unmarkierte Seite p_i mit $q_i \geq 1/(k+1-j)$ auswählen.
- Diese Seite wird vom Gegner angefragt und markiert. Damit endet Subphase j.
- Ist $\gamma>0$, dann gibt es ein $p_i \in M$ mit $q_i=\varepsilon>0$. Der Gegner fragt p_i an und erzeugt somit bei A erwartete Kosten ε.
- Solange die erwarteten Kosten für Subphase j kleiner als $1/(k+1-j)$ sind und $\gamma>\varepsilon$ ist, fragt der Gegner die markierte Seite $p_i \in M$ mit dem größten q_i an.
- Nach dem Ende dieser Schleife fragen wir die unmarkierte Seite p_i mit größtem q_i an und markieren sie, womit Subphase j beendet wird.

Paging

Beweis (Fortsetzung):

- Wird die Schleife verlassen, dann sind die erwarteten Kosten entweder mindestens $1/(k+1-j)$ oder es gilt $\gamma \leq \varepsilon$.
- Die erwarteten Kosten am Ende der Schleife sind also mindestens

$$\varepsilon + \frac{1 - \sum_{p_i \in M} q_i}{k+1-j} = \varepsilon + \frac{1-\gamma}{k+1-j} \geq \varepsilon + \frac{1-\varepsilon}{k+1-j} \geq \frac{1}{k+1-j}$$

- Am Phasenende bleibt die zuletzt markierte Seite für die neue Phase markiert.
- Am Anfang einer Phase entfernt der Gegner gerade die Seite aus dem Cache, die zu Beginn der folgenden Phase angefragt wird. So erreicht er Kosten 1 pro Phase.

Kapitel 7: Online Algorithms

Inhalt:

- Deterministische Online Algorithmen
  - Einführung
  - Paging
  - Scheduling
- Randomisierte Online Algorithmen
  - Paging
  - Finanzielle Spiele

Finanzielle Spiele

Sei $\sigma = p_1, p_2, p_3, \ldots$ eine Sequenz von Preisen für ein Gut.

Problem: Wähle das Maximum (Minimum) in σ.
Akzeptieren wir den i-ten Preis, dann ist unser Ertrag p_i.
Andernfalls spielen wir weiter und das Angebot p_i ist verfallen.

Beispiele: Hausverkauf, Gehaltsverhandlung, Aktienkauf

Einschränkung: Wir beschränken uns auf Suchprobleme mit einer beschränkten Preisspanne $[m,M]$ mit $0<m\leq M$, die dem Online-Spieler bekannt ist.

Finanzielle Spiele

Algorithmus RPP (Reservation-Price-Policy):

Wir setzen einen Reservation Price $p^* = \sqrt{m \cdot M}$ und akzeptieren den ersten Preis p mit $p \geq p^*$. Falls wir am Ende noch kein Angebot angenommen haben, geben wir uns mit dem letzten Angebot zufrieden.

Satz: RPP ist $\sqrt{\varphi}$-kompetitiv, wobei $\varphi = M/m$ ist.

Beweis:

- Sei σ beliebig und p_{max} der maximale Preis der Sequenz.
- Ist $p^* \geq p_{max}$, dann ist
  $OPT(\sigma)/RPP(\sigma) \leq p_{max}/m \leq p^*/m = \sqrt{Mm}/m = \sqrt{\varphi}$
- Ist $p^* < p_{max}$, dann ist
  $OPT(\sigma)/RPP(\sigma) \leq p_{max}/p^* \leq M/\sqrt{Mm} = \sqrt{\varphi}$

In beiden Fällen ist also RPP $\sqrt{\varphi}$–kompetitiv.

Finanzielle Spiele

Satz: Kein deterministischer Online-Algorithmus A ist besser als $\sqrt{\varphi}$-kompetitiv.

Beweis:

- Der Gegner bietet als ersten Preis $\sqrt{mM}$.
- Akzeptiert A das Angebot, dann bieten wir als nächstes M und erhalten als Verhältnis $\sqrt{\varphi}$.
- Akzeptiert A das Angebot nicht, dann bieten wir als nächstes m und erhalten ebenfalls als Verhältnis $\sqrt{\varphi}$.

Finanzielle Spiele

Wie wir sehen werden, können randomisierte online Algorithmen deutlich besser sein.

Algorithmus EXPO:
Sei $M=m\cdot 2^k$ mit $k\in IN$, also $\varphi=2^k$. Sei RPP_i der RPP Algorithmus mit $p^*=m\cdot 2^i$. Wähle mit Wahrscheinlichkeit $1/k$ die Strategie RPP_i für $i\in\{1,\ldots,k\}$.

Satz: EXPO ist $c(\varphi)\log \varphi$–kompetitiv mit $c(\varphi)\to 1$ für $\varphi\to\infty$.

Finanzielle Spiele

Beweis:

- Sei eine beliebige Preisfolge σ gegeben und $j \in IN$ so, dass $m2^j \leq p_{max} < m2^{j+1}$.
- Hat EXPO die Strategie RPP_i gewählt, dann ist für $i \leq j$ der Ertrag mindestens $m2^i$. Für $i>j$ und damit $p^*>p_{max}$ ist der Ertrag von EXPO mindestens m. Als erwarteter Ertrag von EXPO ergibt sich daraus
  $E[EXPO(\sigma)] = (1/k)(\Sigma_{i=1}^{j} m2^i + (k-j)m)$
  $= (m/k)(2^{j+1}-2+k-j)$
- Also ist
  $OPT(\sigma)/E[EXPO(\sigma)] \leq m2^{j+1}/[(m/k)(2^{j+1}-2+k-j)]$
  $\leq k/(1+(k-j-2)/2^{j+1})$
  $= \log \varphi /(1+(k-j-2)/2^{j+1})$
- Der Term $1/(1+(k-j-2)/2^{j+1})$ wird maximiert für $j^*=k-2+1/\log 2$, wodurch dieser für $k \to \infty$ gegen 1 konvergiert.

Fragen?