Approximations- und Online-Algorithmen

thgoebel@ethz.ch

ETH Zürich, FS 2022

This documents is a **short** summary for the course *Approximations- und Online-Algorithmen* at ETH Zurich. It is intended as a document for quick lookup, e.g. during revision, and as such does not replace attending the lecture, reading the slides or reading a proper book.

We do not guarantee correctness or completeness, nor is this document endorsed by the lecturers. Feel free to point out any errata, either by mail or on Github.

Contents

Part I.
Approximations-Algorithmen

1. Approximations-Algorithmen

TODO. Siehe das Skript von letzem Jahr.

Part II.
Online-Algorithmen

2. Einführung und das Paging-Problem

Konzepte

- Online-Problem, Online-Algorithmus, kompetitiver Faktor
- Skirental-Problem
- Paging-Problem
- Randomisierte Online-Algorithmen
- Yaos Prinzip

Motivation Probleme lösen und Entscheidungen fällen ohne alle für eine optimale Lösung relevanten Informationen zu haben. Stattdessen werden die Informationen stückweise zur Laufzeit bekannt.

Beispiel: Skirental-Problem Unendlich langer Urlaub, nur an schönen Tagen Ski fahren. Skier mieten für 1 CHF pro Tag, oder kaufen für k CHF. Erst am Tag selbst wird bekannt ob ein Tag schön ist.

Optimale Lösung: Sei s die Anzahl schöner Tag. Miete bei $s < k$, kaufe bei $s > k$, bei $s = k$ egal.

Problem: s nicht bekannt, erst am Tag selber wird bekannt ob ein Tag schön ist.

Szenario	Worst Case	Approximationsgüte
An Tag 1 kaufen	Ab Tag 2 schlechtes Wetter	$\frac{k}{1}$
Immer mieten	An $x >> k$ Tagen schönes Wetter	$\frac{x}{k}$
An $k-1$ Tagen mieten, dann kaufen	Ab Tag $k+1$ schlechtes Wetter	$\frac{2k-1}{k} = 2 - \frac{1}{k}$

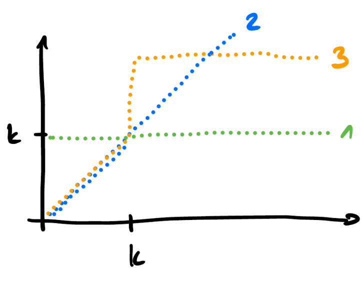

Figure 1: Skirental Szenarios

Online-Problem Ein *Online-Minimierungsproblem* ist $\Pi = (I, O, cost, \min)$. Eine Eingabe $I = (x_1, ..., x_n) \in \mathcal{I}$ ist eine Folge von *Anfragen*, jeweils für *Zeitschritt* i. Eine akzeptierte Lösung $O = (y_1, ..., y_n)$ ist eine Folge von *Antworten*.

Beim analogen Maximierungsproblem spricht man statt von $cost(I, O)$ oft vom *Gewinn* $gain(I, O)$.

Online-Algorithmus Sei Π ein Online-Optimierungsproblem. Ein *Online-Algorithmus* $\mathcal{A}$ berechnet die Ausgabe $\mathcal{A}(I) = (y_1, ..., y_n)$ wobei y_i nur von $(x_1, ..., x_i)$ abhängt. $\mathcal{A}(I)$ ist eine zulässig Lösung für I.

Kompetitiver Faktor (aka. competitive ratio, Wettbewerbsgüte, kompetitive Güte)
Ein Online-Algorithmus $\mathcal{A}$ ist *c-kompetitiv* falls gilt:

$$\exists \alpha \geq 0 \quad \forall I : \quad cost(\mathcal{A}(I)) \leq c \cdot cost(Opt(I)) + \alpha$$
$$\frac{cost(\mathcal{A}(I))}{cost(Opt(I))} + \alpha' \leq c$$

für ein Minimierungsproblem und α konstant. *Opt* ist ein optimaler Offline-Algorithmus, d.h. mit vollständiger Information.

Das kleinste c für das dies gilt heisst *kompetitiver Faktor*.
$\mathcal{A}$ heisst *strikt c-kompetitiv* falls $\alpha = 0$.
$\mathcal{A}$ heisst *optimal* falls er strikt 1-kompetitiv ist ($\alpha = 0, c = 1$).

Wir sprechen hierbei von *kompetitiver Analyse*. Der kompetitiver Faktor ist vergleichbar mit der Approximationsgüte von Approximationsalgorithmen.

Ein Online-Algorithmus heisst *kompetitiv* wenn sein kompetitiver Faktor nicht von der Länge der Eingabe abhängt (d.h. es keine Startkosten gibt die amortisiert werden müssen). Die Konstante α ist wichtig da sie erlaubt auf kurze Eingaben schlecht zu sein (und erst auf lange besser zu werden). [1]

Untere Schranken beweisen Für einen strikt kompetitiven Algorithmus: Finde eine Instanz I mit $\frac{\mathcal{A}(I)}{Opt(I)} > c \implies$ <u>nicht</u> strikt-kompetitiv.

Für einen nicht-strikt kompetitiven Algorithmus: Finde eine unendliche Folge $I_1, I_2, ...$ von Instanzen so dass $\frac{\mathcal{A}(I_i)}{Opt(I_i)} > c$ und $Opt(I_i) \xrightarrow{i \to \infty} \infty$.

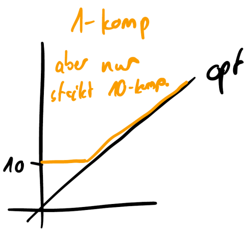

Figure 2: *Opt* in schwarz. $\mathcal{A}$ in orange, 1-kompetitiv und strikt-10-kompetitiv.

2.1. Das Paging-Problem

Paging

- Eingabe: $I = (x_1, ..., x_n)$ mit Speicher-Indizes $x_i \in \mathbb{N}$
- Hauptspeicher mit m Seiten: $(s_1, ..., s_m)$
- Cache-Speicher mit k Seiten: $B = (s_{j_1}, ..., s_{j_k})$, initialisiert mit $(s_1, ..., s_k)$ [2]
- Zeitschritt i:

[1] Warum brauchen wir bei der Approximationsgüte keine vergleichbare Konstante?
[2] Der Vorsprung eines selbstgewählten Startinhalts kann in α versteckt werden.

- Index x_i wird angefragt
- Falls x_i im Cache (d.h. $s_{x_i} \in B$): return $y_i = 0$
- Andernfalls: return $y_i = j$, und setze $B = B \backslash \{s_j\} \cup \{s_{x_i}\}$, d.h. lösche Seite s_j aus dem Cache und ersetze sie durch s_{x_i}. [3]

- $cost(\mathcal{A}(I)) := |\{i \mid y_i > 0\}|$
- goal := min

Strategien bei *Seitenfehlern (page faults)* zum *Verdrängen* von Seiten: First-in-First-Out (FIFO, wie eine Queue), Last-in-First-Out (LIFO, wie ein Stack), Least-Recently-Used (LRU), Longest-Forward-Distance (LFD, offline-only!).

Satz (FIFO) Ein Online-Algorithmus für Paging der FIFO nutzt ist strikt-k-kompetitiv.

Beweis: Gruppiere Zeitschritte in *Phasen*. Phase 1 endet nach dem ersten Seitenfehler. Phase $P \geq 2$ endet nach $1 + (P-1)k$ Seitenfehlern, d.h. alle k Fehler endet eine Phase und beginnt eine neue.

In Phase 1 machen *Opt* und $Fifo$ je genau einen Fehler (warum?).

Sei s die Seite die den letzten Seitenfehler von Phase $P-1$ verursacht (d.h. sie kommt neu in den Cache, und wird dank FIFO als letztes in Phase P verdrängt werden).
$\Longrightarrow$ Zu Beginn von Phase P ist s im Cache von *Opt* und von $Fifo$.
$\Longrightarrow$ Es gibt $\leq k-1$ Seiten die im Cache von *Opt* sind, aber nicht in dem von $Fifo$.
Während Phase P macht $Fifo$ genau k Fehler.
$\Longrightarrow$ Während P muss *Opt* mindestens einen Seitenfehler machen.
$\Longrightarrow$ $Fifo$ ist k-kompetitiv.

LRU ist in der Theorie ebenfalls k-kompetitiv, in der Praxis allerdings tendenziell besser als FIFO.

Satz (untere Schranke) Kein Online-Algorithmus für Paging kann eine besseren kompetitiven Faktor als k erreichen.

Beweis: Sei k die Grösse vom Cache und $k+1$ die Grösse vom Hauptspeicher. [4] Betrachte die "worst case" Eingabe $I = (k+1, s_{y_1}, s_{y_2}, , ..., s_{y_{n-1}})$, d.h. in Zeitschritt i wird die Seite angefragt die $\mathcal{A}$ zuvor erst verdrängt hat. $\mathcal{A}$ verursacht also exakt k Seitenfehler, und *Opt* nur einen in Zeitschritt 1.

Für alle Strategien von $\mathcal{A}$ lässt sich eine worst-case Eingabe konstruieren (siehe Idee eines *Gegenspielers* der die Strategie/den Quellcode kennt). Durch Wiederholen solcher k-langen Phasen lässt sich ausserdem eine unendlich lange Eingabe konstruieren. Eingabelänge n, $\mathcal{A}$ mit n Fehlern, *Opt* mit n/k Fehlern $\Longrightarrow$ k-kompetitiv. [5]

2.2. Randomisierte Online-Algorithmen

Motivation Randomisierung verunmöglicht es dem Gegenspieler die genaue Strategie von $\mathcal{A}$ zu kennen, d.h. es verunmöglicht ihm eine worst case Instanz zu konstruieren.

[3] Zusätzliches, proaktives Entfernen bringt keinen Vorteil.
[4] $k+1$ macht die Aussage nur stärker. Warum?
[5] Mit etwas Glück (abhängig davon was $\mathcal{A}$ in zukünftigen Phasen verdrängt) macht *Opt* sogar nur den Fehler in Zeitschritt 1, und macht danach nie wieder einen Fehler.

Randomisierter Online-Algorithmus Bekommt als Eingabe zusätzlich ein unendliche langes Zufallsband ϕ mit Zufallsbits (die u.a.r. 0 oder 1 sind). Jede Antwort y_i darf nur von $\phi, x_1, ..., x_i, y_1, ..., y_{i-1}$ abhängen.

Beobachtung: Jeder randomisierte Algorithmus *Rand* der $b(n)$ Zufallsbits für Eingaben der Länge n liest kann als eine Menge $strat(Rand) = \{A_1, ..., A_{2^{b(n)}}\}$ von $2^{b(n)}$ deterministischen Online-Algorithmen angesehen werden, von denen einer mit Wahrscheinlichkeit jeweils $\frac{1}{2^{b(n)}}$ ausgewählt wird.

Erwarteter kompetitiver Faktor Ein Online-Algorithmus *Rand* ist *c-kompetitiv im Erwartungswert* falls

$$\exists \alpha \geq 0 \quad \forall I \ : \quad \mathbb{E}[cost(Rand(I))] \leq c \cdot cost(Opt(I)) + \alpha$$

Das kleinste c für das dies gilt heisst *erwarteter kompetitiver Faktor*.
Rand heisst *strikt c-kompetitiv im Erwartungswert* falls $\alpha = 0$.

Wahrscheinlichkeitsverstärkung Einen randomisierten Offline-Algorithmus der mit Wahrscheinlichkeit $\frac{1}{2}$ korrekt ist, kann man k Mal wiederholen um $\frac{1}{2^k}$ zu erreichen. Online ist dies nicht möglich, da wir direkt eine Antwort auf jede Anfrage geben müssen.

Randomisierter Paging-Algorithmus RMark Eine Phase endet/beginnt wenn nach einem Seitenfehler alle Seiten unmarkiert werden.

Algorithm 1 RMark

```
mark alle Seiten im Cache
while Eingabe ist noch nicht beendet do
    s ← Seite mit Index x_i
    if s ist im Cache then
        if s ist unmarkiert then
            mark s
        end if
        output "0"
    else
        if es existiert keine unmarkierte Seite mehr im Cache then
            unmark alle Seiten im Cache
        end if
        s' ← zufällig gewählte unmarkierte Seite
        verdränge s' und füge s an der alten Stelle von s' ein
        mark s
        output "Index von s' "
    end if
    i ← i + 1
end while
```

Satz $RMark$ hat einen erwarteten kompetitiven Faktor von $2H_k$. [6] D.h. $RMark$ ist im Erwartungswert $\mathcal{O}(\log k)$-kompetitiv.

Beweis: Siehe auch Skript S.14ff.

[6] Für jedes $l \in \mathbb{N}^+$ heisst H_l die *l-te Harmonische Zahl* und $H_l := 1 + \frac{1}{2} + \frac{1}{3} + \cdots + \frac{1}{l} = \sum_{i=1}^{l} \frac{1}{i}$.

Betrachte eine einzelne Phase P. O.B.d.A werden k veschiedene Seiten angefragt (eventuell auch mehrmals). Im worst case werden zuerst l "neue" Seiten angefragt, und danach $k-l$ "alte". Mit Wahrscheinlichkeit $(k-l)/k$ ist die erste alte Seite noch im Cache, dann mit Wahrscheinlichkeit $(k-l-1)/(k-1)$ die zweite alte, usw. Umgekehrt ist die i-te alte Seite mit Wahrscheinlichkeit $1-\frac{k-l-(i-1)}{k-(i-1)} = \frac{l}{k-(i-1)}$ <u>nicht</u> mehr im Cache. Die erwarteten Kosten während P sind also

$$l + \sum_{i=1}^{k-l} \frac{l}{k-(i-1)} = \dots = l(H_k - H_l + 1) \leq lH_k$$

Ausserdem gilt $l \geq 1$ da jede Phase per Definition mit einer neuen Seite beginnt.

Betrachte die Kosten von *Opt*. Betrachte zwei aufeinanderfolgende Phasen P_{j-1}, P_j. In diesen wurden $\geq k + l_j$ verschiedene Seiten angefragt. $\implies$ *Opt* macht $\geq l_j$ Seitenfehler. *RMark* und *Opt* machen in P_1 beide l_1 Fehler (da sie mit demselben Cache starten).

Durch unterschiedliches Gruppieren $((P_1, P_2), (P_3, P_4), ...$ vs $P_1, (P_2, P_3), (P_4, P_5), ...)$ erhalten wir:

$$cost(Opt(I)) \geq \max \left\{ \sum_{i=1}^{\lfloor N/2 \rfloor} l_{2i}, \sum_{i=1}^{\lceil N/2 \rceil} l_{2i-1} \right\} \geq \frac{1}{2} \left(\sum_{i=1}^{\lfloor N/2 \rfloor} l_{2i} + \sum_{i=1}^{\lceil N/2 \rceil} l_{2i-1} \right) = \sum_{i=1}^{N} \frac{1}{2} l_i$$

Der kompetitive Faktor ist also

$$c \geq \frac{\sum_{i=1}^{N} H_k l_i}{\sum_{i=1}^{N} \frac{1}{2} l_i} = 2H_k$$

Verbesserung Für Paging existiert kein deterministischer Online-Algorithmus mit kompetitivem Faktor k (s.o.). Mit Randomisierung können wir im Erwartungswert aber $\mathcal{O}(\log k)$ erreichen! D.h. asymptotisch expontentieller Speedup! Dies ist asymptotisch optimal für randomisierte Algorithmen (s.u.).

2.3. Yaos Prinzip

Motivation Untere Schranke für kompetitiven Faktor von deterministischen OAs $\implies$ Untere Schranke für erwarteten kompetitiven Faktor von randomisierten OAs

Wahrscheinlichkeitsverteilung über Instanzen $\Pr_{Adv} \implies$ W'keitsverteilung über Algorithmen $\Pr_{Rand}$.

Limitierung: konstante Anzahl von Instanzen $\mathcal{I} = \{I_1, ..., I_m\}$ und Algorithmen $strat(Rand) = \{A_1, ..., A_l\}$ (d.h. Eingabelänge n begrenzt).

Lemma 1 (1.13) Sei Π ein Optimierungsproblem, sei $\mathcal{I}$ eine Klasse von Instanzen. Sei $\Pr_{Adv}$ eine W'keitsverteilung so dass gilt:

$$\forall A \in strat(Rand) \ : \ \mathbb{E}_{Adv}[cost(A(\mathsf{I}))] \geq c \cdot \mathbb{E}_{Adv}[cost(Opt(\mathsf{I}))]$$

Dann gilt:

$$\forall Rand \ \exists I \in \mathcal{I} \ : \ \mathbb{E}_{Rand}[cost(\mathsf{A}(I)))] \geq c \cdot cost(Opt(I))$$

wobei A, I Zufallsvariablen sind aus den Wahrscheinlichkeitsräumen $\Pr_{Rand}, \Pr_{Adv}$.

<u>Beweis:</u> Siehe Skript S.18f.

Lemma 2 (1.14) Seien $\Pi, \mathcal{I}, \Pr_{Adv}$ wie oben. Sei $\forall$ det. OAs A_j der erwartete kompetitive Faktor $\geq c$, d.h.

$$\mathbb{E}_{Adv}\left[\frac{cost(A_j(\mathsf{I}))}{cost(Opt(\mathsf{I}))}\right] \geq c$$

Dann gilt: $\forall$ rand. OAs ist der erwartete kompetitive Faktor $\geq c$, d.h. $\exists I \in \mathcal{I}$ so dass

$$\frac{\mathbb{E}_{Rand}[cost(\mathsf{A}(I))]}{cost(Opt(I))} \geq c$$

Beweis: Siehe Skript S.20f.

Satz (Yaos Prinzip) Folgt aus Lemma 1 und 2. Seien $\Pi, \mathcal{I}, \Pr_{Adv}$ wie oben. Für jeden randomisierten Online-Algorithmus existiert dann eine Eingabe I so dass

$$\frac{\mathbb{E}_{Rand}[cost(\mathsf{A}(I))]}{cost(Opt(I))} \geq \max\left\{\left\{\min_j \frac{\mathbb{E}_{Adv}[cost(A_j(\mathsf{I}))]}{\mathbb{E}_{Adv}[cost(Opt(\mathsf{I}))]}\right\}, \left\{\min_j \mathbb{E}_{Adv}\left[\frac{cost(A_j(\mathsf{I}))}{cost(Opt(\mathsf{I}))}\right]\right\}\right\}$$

Anders formuliert (laut Wikipedia):

$$\max_{I \in \mathcal{I}} \mathbb{E}_{Rand}[cost(\mathsf{A}(I))] \geq \min_{A \in strat(Rand)} \mathbb{E}_{Adv}[cost(A(\mathsf{I}))]$$

wobei A, I Zufallsvariablen sind.

Spieltheoretische Interpretation Yaos Minimax Prinzip. Spezialfall von Von Neumanns Minimax Theorem (in Nullsummenspielen mit 2 Spielern und gemischten Strategien gibt es ein Gleichgewicht). Zero-sum game, Spieler A wählt den det. Algorithmus, Spieler B wählt die Instanz, der payoff ist $cost(A_j(I))$.

Für jeden Spieler ist "zufällig wählen" eine Strategie. Aus Yao folgt: für eine fixe Eingabe zufällig eine Algo wählen ist nicht schlechter als für einen fixen Algo zufällig eine Eingabe wählen.

Satz (Untere Schranke für randomisiertes Paging) Kein randomisierter Online-Algorithmus für Paging kann einen besseren (= kleineren) erwarteten kompetitiven Faktor als H_k erreichen.

Beweis: Siehe Skript S.27ff.

Analog zu Paging: k Cache, $k+1$ Hauptspeicher, frage zuerst s_{k+1} an, danach (neu!) jede der nicht gerade angefragten Seiten mit Wahrscheinlichkeit $\frac{1}{k}$. Eine Phase endet nach k Fehlern, d.h. nachdem alle $k+1$ Seiten mind. einmal angefragt wurden.

Betrachte einzelne Phase. Zeige dass für alle deterministischen A_j die erwarteten Kosten circa H_k-mal höher sind als die von Opt. Es gilt für die Eingabe während Phase P:

$$\frac{\mathbb{E}_{Adv}[cost(A_j(P))]}{\mathbb{E}_{Adv}[cost(Opt(P))]} \geq \frac{|P| \cdot \frac{1}{k}}{1} = \frac{|P|}{k}$$

Schätze ab (siehe Skript, siehe Coupon Collector): $\mathbb{E}_{Adv}[\,|P|\,] = 1 + k \cdot H_k$.

Wende Yaos Prinzip an:

$$\frac{\mathbb{E}_{Rand}[cost(\mathsf{A}(I))]}{cost(Opt(I))} \geq H_k$$