Notizen zu Algorithmen II

Jens Ochsenmeier

12. Februar 2018

Inhaltsverzeichnis

1

Online-Algorithmen

Inhalt dieses Kapitels:

- Einführung in Online-Algorithmen
- Beispiel: Job-Scheduling
- Beispiel: Skiausleihe
- Beispiel: Speicherverwaltung
- Beispiel: Auswahl von Experten

Online-Algorithmen werden verwendet, wenn Eingabegrößen nicht im Vornerein bekannt sind. Viele Algorithmen, die wir bisher diskutiert haben, benötigen Vorarbeitszeit, um optimale Ergebnisse liefern zu können, und sind daher nicht auf Probleme anwendbar, wo die Eingabe in serieller Natur vorliegt. Die bisher diskutierten Algorithmen werden daher auch **Offline-Algorithmen** genannt.

1.1 Übersicht

Wir werden uns im Folgenden Konzepte von Online-Algorithmen anhand von Beispielen anschauen. Zuerst definieren wir, was ein Online-Algorithmus formal ist.

- **Eingabe**: Folge von Anforderungen (*requests*)

  $$\sigma = (r_1, \ldots, r_n) \in R^n$$

- r_i muss auf Anforderung bearbeitet werden, es entsteht eine Antwort

  $$a_i = g_i(r_1, \ldots, r_i) \in A.$$

  Es sind keine Informationen über die Zukunft verfügbar ($r_{i+1}, \ldots$). Antworten sind unwiderruflich.
  Der konkrete Algorithmus ist also durch $g_1, \ldots$ eindeutig festgelegt.
- **Kosten** $\text{cost}_n : R^n \times A^n \to \mathbb{R}_{>0}$
- **Ausgabe** für $\sigma \in R^n$ ist also

  $$\text{alg}[\sigma] = (g_1(r_1), g_2(r_1, r_2), \ldots, g_n(r_1, \ldots, r_n)) \in A^n$$

  und die Kosten

  $$\text{alg}(\sigma) = \text{cost}_n(\sigma, \text{alg}[\sigma])$$

Kompetitive Analyse

Um einschätzen zu können, wie gut ein Online-Algorithmus ist, vergleichen wir ihn mit einem optimalen Offline-Algorithmus — das ist die **Kompetitive Analyse** dieses Online-Algorithmus.

Definition 1.1.1 (c-kompetitiv). Ein Online-Algorithmus ist für ein Optimierungsproblem auf Input σ **c-kompetitiv**, falls es einen Offline-Algorithmus OPT gibt, der nach einem Kostenmaß $\text{OPT}(\sigma)$ eine Optimallösung berechnen kann, sodass für den betrachteten Online-Algorithmus ALG und alle Eingabesequenzen σ gilt:

$$\text{ALG}(\sigma) \leq c \cdot \text{OPT}(\sigma) + \alpha.$$

Ist die zusätzliche Konstante $\alpha \leq 0$, dann heißt ALG **strikt c-kompetitiv**.
Unterschied ist, dass man bei nicht-strikter kompetitivität Ausnahmen erlaubt.

Wir können nun den **Wettbewerbsfaktor** (*competitive ratio*) für den Algorithmus ALG definieren als

$$c_{\text{alg}} = \sup\left\{\frac{\text{alg}(\sigma)}{\text{opt}(\sigma)} : \sigma \in R^+\right\}.$$

Ist alg ein strikt c-kompetitiver Online-Algorithmus und

$$C = \{c : \text{alg ist strikt } c\text{-kompetitiv}\},$$

so ist

$$c_{\text{alg}} = \inf C \quad \text{und} \quad c_{\text{alg}} \in C.$$

Im Folgenden geben wir eine kurze Übersicht über die Beispiele, die wir im Folgenden behandeln werden.

Job-Scheduling

Dieses Beispiel ist dem Beispiel im Kapitel "Approximationsalgorithmen" sehr ähnlich. Wir haben

- **Maschinen** $M_1, \ldots, M_m$
- **Anfrage**: Job J_i, benötigt Zeit $t_i \geq 0$
- **Antwort**: Zuordnung von J_i zu M_j

Wieder stellt sich die Frage, wie sich der Makespan (also das Intervall zwischen Start und Fertigstellen der letzten Maschine) minimieren lässt. Diese Entscheidung muss hier für jedes J_i ohne Kenntnis über die zukünftigen Jobs passieren.

Skiausleihe

Die Situation ist hier folgende: Man befinde sich im Skiurlaub und solange das Wetter gut ist, lautet jeden Morgen die **Anforderung** "Ski ausleihen!". Sobald das Wetter allerdings schlecht ist, lautet die **Anforderung** "Heimfahren!".
Es sind zwei **Antworten** möglich:

- Ski für einen Tag ausleihen: Kosten k Euro
- Ski kaufen: Kosten K Euro ($K \gg k$)

Was muss nun getan werden, um die Gesamtkosten klein zu halten? Diese Entscheidung muss ohne Kenntnis des zukünftigen Wetters gemacht werden!

Speicherverwaltung

"Kosten minimieren" bedeutet im Speicherverwaltungskontext meist, die Anzahl an Cache Misses zu minimieren. Welche Verdränungsstrategie minimiert die Kosten hier am besten? Und wie kann man am besten zu verdrändende Seiten auswählen, ohne Kenntnis über zukünftige Anforderungen zu haben?

Auswahl von Experten

Hier gibt es mehrere Runden, jede läuft wie folgt ab:

1. Jeder von n Experten gibt zu einer Frage eine Ja/Nein-Empfehlung ab. Diese sind im Allgemeinen nicht richtig.
2. Man trifft seine eigene Ja/Nein-Entscheidung zu derselben Fragestellung.
3. Es wird mitgeteilt, welche Entscheidung richtig gewesen wäre.

Wie lässt sich hier die Anzahl an Fehlentscheidungen minimieren?

Selbstorganisierende Datenstrukturen

Hier haben wir eine einfach verkettete Liste und erhalten als **Anforderung** das Element x in der Liste. Die dabei entstehenden Kosten sind x. Als **Reaktion** kann entweder

- ohne weitere Kosten das angefragte Element weitrer nach vorne gerückt werden, oder
- mit Kosten von 1 zwei aufeinanderfolgende Elemente vertauscht werden.

Welche Listen-Verwaltung minimiert hier die Kosten? Wie kann ohne Kenntnis über zukünftige Anforderungen sinnvoll umgeordnet werden?

1.2 Job-Scheduling

Wir nehmen den listScheduling-Approximationsalgorithmus und bauen ihn in einen Online-Algorithmus um:

```
LISTSCHEDULING(n, m, t_{1...n})
each L_i := 0   // load of machine 1 ≤ i ≤ m
each S_j := 0   // machine for job 1 ≤ j ≤ n
for each j in range(1, n) do
  pick k from {i : L_i is currently minimal}
  S_j := k
  L_k := L_k + t_j
return S
```

Frühere Analysen ergeben, dass der Wettbewerbsfaktor höchstens 2 ist. Der Online-Algorithmus sieht nun so aus:

```
LISTSCHEDULING(  m      )
each L_i := 0   // load of machine 1 ≤ i ≤ m

for each t_j in σ do
  pick k from {i : L_i is currently minimal}
  a_j := k
  L_k := L_k + t_j
  assign job to machine a_j
```