Algorithmendesign

Gruppe 2

Lösungen zu Übungsblatt 10

Max Bannach

WS 13/14

Markus Richter (614027)

Aufgabe 10.1 Online-Interval-Scheduling

Teilaufgabe 1)

Nicht stark ρ-kompetitiv bedeutet

$$\forall \mathcal{A} \ \forall \rho \ \exists X : \mathcal{A}(X) < \frac{\mathrm{OPT}(X)}{\rho}$$

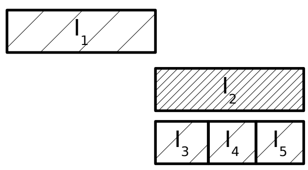
Das lässt sich relativ einfach anhand einer Strategie für einen Gegenspieler zeigen. Siehe hierfür auch das Tantau Skript Seite 36. Dem Online-Algorithmus wird im Grunde ein Intervall nach dem anderen vorgesetzt und dieser entscheidet, ob er es schedult oder nicht. Der Gegenspieler

1. produziert die Intervalle $[1,2], [2,3), [3,4)$ usw. bis der Algorithmus sich das erste Mal bei einem Intervall für *schedule* entscheidet.
2. Nun produziert er Intervalle der Größe $\frac{1}{\rho+1}$ und positioniert sie alle innerhalb des Intervalls, für welches sich der Algorithmus vorher entschieden hat.

Der Gegenspieler produziert auf diese Weise mindestens $\rho + 2$ Intervalle. Der Algorithmus entscheidet sich also nur für ein Intervall, obwohl $\rho + 1$ Intervalle möglich wären. Damit ist die kompetitive Rate schlechter als ρ.

∎

Gegeben sei folgendes Beispiel:



Sei $\rho = 2$. Entscheidet sich der Algorithmus für I_2, so platziert der Gegenspieler I_3 bis I_5 unterhalb von I_2. Damit gilt: $1 < \frac{4}{2}$

Teilaufgabe 2)

Um zu zeigen, dass auch für die Summe der Ausführungszeiten eines Schedules für kein festes $\rho \in \mathbb{N}$ einen stark ρ-kompetitiven Algorithmus gibt, verfährt man analog zu oben und entwirft eine Gegenspieler-Strategie:

1. Der Gegenspieler generiert ein Intervall nach dem anderen mit gleicher Länge mit Hilfe von f_i und s_i.
2. Entscheidet sich der Algorithmus das erste mal für ein Intervall I_j, so produziert der Gegenspieler ein Intervall der Länge $(\rho+1)(f_j - s_j)$ und platziert ihn so, dass der Algorithmus ihn nicht mehr schedulen kann. Erst nach Ende dieses Intervalls produziert der Gegenspieler wieder gleich lange Intervalle.

Es gilt also:

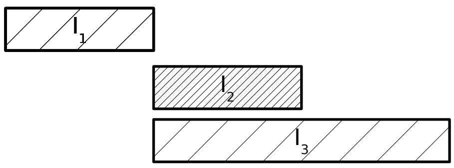
$$\frac{\mathcal{A}(\mathrm{X})}{\mathrm{OPT}(X)} \geq \frac{\sum_{x_i \in S} |f_i - s_i|}{\sum_{x_i \in S} (\rho+1)|f_i - s_i|} \Leftrightarrow \frac{\mathcal{A}(\mathrm{X})}{\mathrm{OPT}(X)} \geq \frac{1}{\rho+1}$$

Gelten muss jedoch

$$\frac{\mathcal{A}(\mathrm{X})}{\mathrm{OPT}(X)} \geq \frac{1}{\rho}$$

Daraus folgt, dass die kompetitive Rate schlechter ist als ρ.

∎



Aufgabe 10.2 Feuerwehr

Teilaufgabe 1

Im Folgenden nehme ich an, dass die Linienmenschen einen begrenzten Bereich der unendlich langen Straße (wie soll das gehen?) bebaut haben, andernfalls hätten sie nicht wenig natürliche Resourcen. Mit wenig natürlichen Resourcen können sie auch nur endlich viele Wagen bauen und damit nur einen endlich langen Bereich der Straße abdecken.

Angenommen der Algorithmus iteriert von einem Schritt oder Zeitpunkt zum nächsten. Zu jedem solchen Zeitpunkt i bricht *ein* Brand aus. Dann beschreibt $X = x_1, x_2, \dots, x_n$ mit $x_i \in [A, E]$ eine Stelle im Linienland, an der zum Zeitpunkt i ein Brand ausbricht. A und E beschreiben die Grenzen, A ist der Anfang und E das Ende.

$Y = y_1, y_2, \dots, y_n$ mit $y_i \in \{1, \dots, k\}$ ist dann die Aktion zum Zeitpunkt i. Die Aktion besagt welcher der k Wagen den Brand löschen soll. Damit die Wagen w wissen, wohin sie fahren müssen, müssen sie wissen wo sie sich zum Zeitpunkt i befinden. Sei dies ihre Position p_{w_i}.

Das Optimierungsproblem Π besteht also nun darin die Entfernungen $p_{w_{i-1}}$ und p_{w_i} aller Wagen w zwischen zwei aufeinander folgenden Zeitpunkten $i-1$ und i zu minimieren.

Teilaufgabe 2

Die Idee ist nun den bebauten Bereich der Straße in gleichlange Partitionen aufzuteilen. Für die Länge einer Partition gilt $l = \frac{E}{k}$. Jeder Wagen hat also seine Partition. Zu Beginn wird jeder Wagen in der Mitte seiner Partition positioniert. Initial müssen alle Wagen an ihre Position gebracht werden, was in einer Wegstrecke $\frac{k}{2} \cdot E$ resultiert.

Damit man weiß welcher Wagen für einen Brand x_i zuständig ist, wird die zuständige Partition wie folgt ermittelt: $(x_i \mod l) + 1$.

Im schlimmsten Fall kann es passieren, dass ein Feuer immer an den Grenzen einer Partition ausbricht, z. B. bei 0 und l, sodass ein Wagen ständig hin und her fahren muss. Ein Offline-Algorithmus weiß das bereits im Vorfeld und kann gleich zu Beginn die Wagen direkt an die Grenzen stellen, sodass es nur initial einer Bewegung bedarf und danach keiner weiteren.

Für die kompetitive Rate – wobei n für die Anzahl der Zeitpunkte steht – bedeutet das

$$\mathrm{CR}(\mathcal{A}(\mathrm{X})) = \frac{\frac{k^2}{2} \cdot l + l \cdot n}{l + 0} = \frac{k^2}{2} + n$$

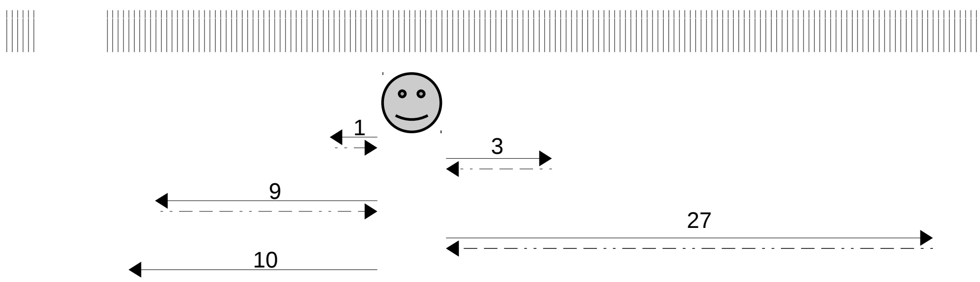
Man sieht also, dass der Ausdruck beliebig wächst für n, außerdem gilt $k^2 > k$. Daraus resultiert, dass der Algorithmus nicht k-kompetitiv ist.

Aufgabe 10.3 Flucht der Kuh

Teilaufgabe 1

Die denkbar schlechteste Position für das Verhältnis ist die, bei der die Kuh das Loch knapp verfehlt, dann eine möglichst lange Strecke in die entgegengesetzte Richtig hin und zurück laufen muss, zuzüglich die direkte Entfernung Ausgangspunkt zum Loch. Man will also den Ausdruck $\frac{\mathcal{A}(X)}{\mathrm{OPT}(X)}$ maximieren, indem man den Zähler möglichst maximiert und den Nenner minimiert.

Sie das Loch an Position -10. Die Kuh läuft dann eine Strecke von 90 Metern, bis sie das Loch gefunden hat. Die Offline-Version des Algorithmus bräuchte aber nur 10 Meter, was der optimalen Strecke entspricht. Das Verhältnis in diesem Beispiel ist also $\frac{90}{10} = 9$.



Teilaufgabe 2

Für den schlimmsten Fall, also einem Loch bei $3^k + 1$, wobei k der Anzahl der Richtungswechsel entspricht, beträgt die optimale Strecke eben $3^k + 1$. Der Algorithmus jedoch läuft
$2 \cdot \sum_{i=0}^{k+1} 3^i + (3^k + 1) = 3^{k+2} - 1 + (3^k + 1) = 3^{k+2} + 3^k < 10 \cdot (3^k + 1) < 10 \cdot OPT$
Daraus folgt, dass die Strategie der Kuh maximal das 10-fache des optimalen Weges zurücklegt.

Teilaufgabe 3

Ja, indem man verdoppelt statt zu verdreifachen. Dann ist der Worst-Case nämlich $2^k + 1$. Daraus ergibt sich für den Algorithmus:
$2 \cdot \sum_{i=0}^{k+1} 2^i + (2^k + 1) = 2 \cdot (2^{k+2} - 1) + (2^k + 1) < 8 \cdot 2^k + 2^k < 9 \cdot OPT$
Die kompetitive Rate sinkt dann also von 10 auf 9.