Wiederholung:

Ein <u>metrischer Raum</u> M ist ein Paar (S, d), wobei S eine Menge von Punkten und $d: S \times S \to \mathbb{R}^+$ eine <u>Distanzfunktion</u> sind. Dabei gilt:

i)	$d(i,j) > 0$	$\forall i \in S, j \in S \setminus \{i\}$	(Positivität)
ii)	$d(i,i) = 0$	$\forall i \in S$	(Reflexivität)
iii)	$d(i,j) + d(j,k) \geq d(i,k)$	$\forall i,j,k \in S$	(Δ-Ungleichung)
iv)	$d(i,j) = d(j,i)$	$\forall i,j \in S$	(Symmetrie)

3. Das k-Server-Problem

Seien $k > 1$ ganzzahlig, $M = (S, d)$ ein metrischer Raum mit $|S| > k$.

<u>Problemstellung:</u>

- k mobile Server befinden sich auf Punkten in S. Anfragefolge $\sigma := r_1 r_2 \dots r_n$, wobei $r_i \in S$, $1 \leq i \leq n$.
- Um die Anfrage r_i zu bedienen muss sich ein Server auf r_i befinden
- Ein Algorithmus arbeitet σ ab, indem er die Server bewegt um nacheinander alle Anfragen zu bedienen.

$ALG(\sigma)$:= Gesamtdistanz, die die Server, bewegt durch ALG, zurücklegen.

Das k-Server-Problem ist die Frage nach c-kompetitiven Online-Algorithmen für beliebige oder spezielle metrische Räume.

Beispiel:

Paging, modelliert als k-Server-Problem:

$M := (S, d)$, wobei

S Menge aller Seiten

$d: S \times S \to \mathbb{R}^+$ ist definiert durch

$$d(i,j) := \begin{cases} 1 & \text{falls } i \neq j \\ 0 & \text{falls } i = j \end{cases}$$

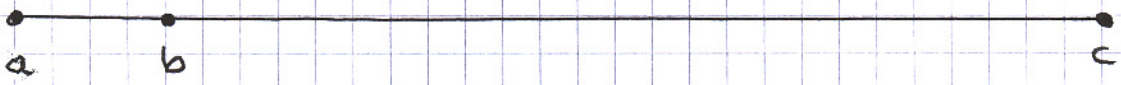
k Server repräsentieren den Hauptspeicher der Größe k.

Übung:
Gegeben sei eine Anfragefolge $\sigma := r_1 r_2 \dots r_n$ für das k-Server-Problem. Entwickeln Sie unter Verwendung von dynamischer Programmierung einen Algorithmus, der einen optimalen Schedule für diese Anfragefolge berechnet. Analysieren Sie die Komplexität Ihres Algorithmus.

Folgende Heuristik sieht plausibel aus:

- Bewege stets einen Server, der sich am nächsten zum angefragten Punkt sich befindet.

Folgendes einfache Beispiel zeigt, dass diese Heuristik beliebig schlecht sein kann:



2 Server, zu Beginn auf a und b.

$\sigma := c\,b\,a\,b\,a\,b\,a\,\ldots$

Im obigen Beispiel führt die Minimierung lokaler Kosten zu unbegrenzten Kosten, während

$$OPT(\sigma) = 2 \cdot d(b,c).$$

16. Okt.

Ähnlich wie beim Paging können wir auch hier das <u>(k,k) - Server - Problem</u> definieren. D.h.,

Online k Server $\longleftrightarrow$ Offline k Server.

Demzufolge implizieren untere Schranken für das (k,k) - Pagingproblem untere Schranken für gewisse (k,k) - Serverprobleme. Folgender Satz gibt uns eine allgemeine untere Schranke:

<u>Satz 3.1</u>

Sei $M = (S, d)$ ein metrischer Raum mit $|S| \geq k+1$. Seien $1 \leq k \leq k$ und ALG ein beliebiger Online-Algorithmus für das (k,k) - Server - Problem auf M. Dann ist das competitive Verhältnis von ALG $\geq \frac{k}{k-k+1}$.

Beweis:

Idee:
Konstruktion einer beliebig langen Anfragefolge σ, so dass

$$ALG(\sigma) \geq \frac{k}{k-k+1} \, OPT(\sigma).$$

O.B.d.A. können wir annehmen, dass ALG stets die k Server auf k verschiedene Punkte in S plaziert.

Wir wählen beliebige $k+1$ Punkte in S und nummerieren diese mit 0 beginnend durch.

Sei $[1, 2, \ldots, k]$ die Startkonfiguration von ALG. D.h., die k Server befinden sich zu Beginn auf den Punkten $1, 2, \ldots, k$.

Bezeichne $\sigma := r_1 r_2 \ldots r_n$ die zu konstruierende Anfragefolge. Dann definieren wir für $1 \leq i \leq n$

$r_i := x \in \{0, 1, 2, \ldots, k\}$ mit kein Server befindet sich auf Punkt x.

$\Rightarrow$

ALG bedient r_i mit dem Server auf Punkt r_{i+1}.

$\Rightarrow$

Die Kosten hierfür sind $d(r_{i+1}, r_i)$.

Sei $n \geq 2$. Dann gilt:

$$ALG(\sigma) \geq \sum_{i=1}^{n-1} d(r_{i+1}, r_i)$$

$$= \sum_{i=1}^{n-1} d(r_i, r_{i+1}).$$

Ziel:
Beweis der Existenz eines Offline-k-Server-Algorithmus, der für σ Kosten

$$\leq \frac{k-k+1}{k} \; ALG(\sigma)$$

hat.

Idee:
Konstruktion einer Menge $\mathcal{B}$ von Offline-Algorithmen nebst Abschätzung der Gesamtkosten von $\mathcal{B}$ für σ. Dann existiert in $\mathcal{B}$ ein Algorithmus, dessen Kosten nicht höher, als die durchschnittlichen Kosten sind.

Durchführung:

Sei $S' := \{0, 1, 2, \ldots, k\}$. Wir betrachten $R \subseteq S'$ mit $|R| = k$ und $r_1 \in R$ beliebig, aber fest. Wir definieren bzgl. R folgenden Algorithmus A_R:

Algorithmus A_R:

- Startkonfiguration ist R
- bedient nicht besetzte Position r_i mit Server auf r_{i-1} $(i = 2, 3, \ldots, k)$

Gemäß Konstruktion gilt $r_1 \in R$. D.h., r_1 befindet sich in der Startkonfiguration.

$\Rightarrow$

A_R ist wohldefiniert.

Sei

$$\mathcal{B} := \{ A_R \mid R \subseteq S,\ |R| = k \text{ und } r_1 \in R \}.$$

Lemma 3.1

Während der Bearbeitung von σ erreichen niemals zwei Algorithmen A_{R_1} und A_{R_2} mit $A_{R_1}, A_{R_2} \in \mathcal{B}$ und $R_1 \neq R_2$ dieselbe Konfiguration.

Bevor wir das Lemma beweisen, führen wir den Beweis von Satz 3.1 zu Ende.

Lemma 3.1 $\Rightarrow$

Alle $\binom{k}{k-1}$ Algorithmen in $\mathcal{B}$ sind stets in paarweise unterschiedlichen Konfigurationen.

Also gilt für $i > 1$:

Exakt $\binom{k-1}{n-1}$ der Algorithmen haben bei der Anfrage r_i keinen Server auf r_i und bewegen den Server von r_{i-1} nach r_i.

$\Rightarrow$

Für die Gesamtkosten $B(\sigma)$ gilt:

$$B(\sigma) = \binom{k-1}{n-1} \sum_{i=2}^{n} d(r_{i-1}, r_i)$$

$$= \binom{k-1}{n-1} \sum_{i=1}^{n-1} d(r_i, r_{i+1}).$$

Also betragen die mittleren Kosten der Algorithmen in $\mathcal{B}$:

$$\frac{\binom{k-1}{n-1}}{\binom{k}{n-1}} \cdot \sum_{i=1}^{n-1} d(r_i, r_{i+1})$$

$$\leq \frac{k-n+1}{k} \; ALG(\sigma).$$

Somit verbleibt nur noch der Beweis von Lemma 2.1.

<u>Bew. v. Lem. 3.1:</u> (indirekt)

Betrachten wir die erste Situation, in der das Lemma verletzt wird.

Sei dies bedingt durch Abarbeiten von r_i, $i > 1$.

D.h., $\exists\, A_{R_1}, A_{R_2} \in \mathcal{B}$, $R_1 \neq R_2$ mit:

- Vor Abarbeitung von r_i befinden sich A_{R_1} und A_{R_2} in unterschiedlichen Konfigurationen.
- Danach sind A_{R_1} und A_{R_2} in derselben Konfiguration.

Seien $\tilde{R}_1$ und $\tilde{R}_2$ die Konfigurationen von A_{R_1} und A_{R_2} unmittelbar vor der Bearbeitung von r_i.

Wir unterscheiden drei Fälle:

1. Fall: $r_i \in \tilde{R}_1 \cap \tilde{R}_2$

Dann ändern beide Algorithmen ihre Konfiguration nicht. ✓

2. Fall: r_i ist in genau einer der Konfigurationen $\tilde{R}_1$ und $\tilde{R}_2$ enthalten.

O.B.d.A. sei $r_i \in \tilde{R}_1$.

$\Rightarrow$

A_{R_1} ändert seine Konfiguration nicht. D.h. insbesondere, dass r_{i-1} seinen Server behält.

Aber:

A_{R_2} bewegt den Server von r_{i-1} nach r_i. D.h., auf r_{i-1} befindet sich anschließend kein Server mehr. ✓

3. Fall: $r_i \notin \tilde{R}_1$ und $r_i \notin \tilde{R}_2$.

Beide Algorithmen bewegen den Server von r_{i-1} nach r_i. ✓ ■

3.1 Spezielle metrische Räume

Zunächst betrachten wir die reelle Linie mit der euklidischen Metrik.

Algorithmus DC

- Anfrage jenseits vom äußersten Server
  ⇝
  Bewege einen nächsten äußersten Server auf den angefragten Punkt.
- Anfrage zwischen zwei benachbarten Server
  ⇝
  Bewege jeweils einen der benachbarten Server mit derselben Geschwindigkeit zum angefragten Punkt bis mindestens einer dieser erreicht hat. HALT.

Beobachtung:
DC stellt nicht sicher, dass sich alle k Server stets auf unterschiedlichen Punkten befinden.

Satz 3.2
DC ist k- competitive.

Beweis:

Bezeichne M_{min} die Kosten eines perfekten Matchings minimaler Kosten zwischen den

Servern von OPT und von DC. Seien

- s_i, $1 \le i \le k$ Positionen der Server von DC
- $\Sigma_{DC} := \sum_{i<j} d(s_i, s_j)$
- Potentialfunktion $\Phi := k \cdot M_{min} + \Sigma_{DC}$.

Beobachtung: $\Phi \geq 0$.

Falls die Startkonfigurationen für OPT und DC gleich sind, dann gilt zu Beginn

$$\Phi_{start} = \Sigma_{DC},$$

was eine Konstante ist.

Für die Durchführung der Potentialfunktionsmethode nehmen wir an, dass OPT stets vor DC seine Serverbewegung durchführt.

Da auch nach der Terminierung $\Phi \geq 0$ gilt, genügt es folgendes zu zeigen:

i) Wenn OPT seine Server eine Distanz d bewegt, dann wächst der Wert der Potentialfunktion höchstens um $k \cdot d$.

ii) Wenn DC seine Server eine Distanz d bewegt, dann vermindert sich der Wert der Potentialfunktion mindestens um d.

i) und ii) implizieren dann

$$DC(\sigma) \leq k \cdot OPT(\sigma) + \Phi_{start}.$$

i) folgt aus folgenden Beobachtungen:

a) Bewegungen von Servern durch OPT verändern Σ_{DC} nicht.

b) Eine Bewegung eines Servers durch OPT um d erhöht M_{min} höchstens um d.

Bew. von ii):

Wir unterscheiden zwei Fälle:

1. Fall: DC bewegt einen einzelnen Server um die Distanz d.

Definition von DC $\Rightarrow$

Äußerster Server wird bewegt. D.h., dieser Server wird von jeden anderen Server um die Distanz d wegbewegt.

$\Rightarrow$

Σ_{DC} wächst um $(k-1) \cdot d$.

Seien s der Server, den DC auf den Anfragepunkt bewegt und t der Server von OPT, der sich auf dem Anfragepunkt befindet.

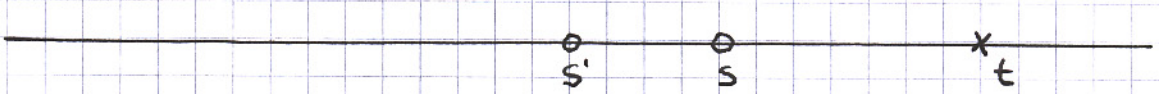
Sei M ein Matching minimaler Kosten unmittelbar bevor DC den Server s bewegt. Zunächst werden wir uns davon überzeugen, dass ein Matching minimaler Kosten existiert, in dem s und t miteinander gematcht sind. Falls M bereits diese Eigenschaft besitzt, dann ist diesbezüglich nichts mehr zu zeigen.

Annahme:
s und t sind in M nicht miteinander gematcht.

Seien s' derjenige Server von DC, der in M mit t und t' derjenige Server von OPT, der in M mit s gematcht ist.

Falls s und s' sich auf demselben Punkt der Linie befinden, dann können wir beide im Matching vertauschen, ohne die Kosten des Matchings zu verändern. Im resultierenden Matching minimaler Kosten sind s und t miteinander gematcht.

Falls sich s und s' auf verschiedenen Punkten befinden, dann liegt, da s äußerster Server ist, folgende Situation vor:



Egal, wo t' auf der Linie sich befindet, führt das Vertauschen von s und s' zu einem Matching, dessen Kosten nicht größer als die Kosten von M sind. (überzeugen sie sich.)

Insgesamt haben wir gezeigt, dass ein Matching minimaler Kosten, in dem s und t miteinander gematcht sind, existiert.

$\Rightarrow$

M_{min} vermindert sich mindestens um d.

$\Rightarrow$

Der Wert der Potentialfunktion vermindert sich mindestens um

$$k \cdot d - (k-1)d = d.$$

2. Fall: DC bewegt zwei Server s_1 und s_2 von zwei gegenüberliegenden Punkten auf den Anfragepunkt zu.

Es existiert ein Matching M minimaler Kosten, in dem einer der Server, die bewegt werden, mit dem Server von OPT, der sich auf dem Anfragepunkt befindet, gematcht ist.

Übung:
Beweisen Sie obige Aussage.

$\Rightarrow$

- Die Kosten der betreffenden Zuordnung vermindert sich in M um $\frac{d}{2}$.

- Der andere Server kann sich von seinem zugeordneten Server maximal um $\frac{d}{2}$ entfernen.

$\Rightarrow$

Gesamtveränderung von $\Phi_{min} \leq 0$.

Änderung von Σ_{DC}:

- Die Summe der Änderungen von s_1 und s_2 zu den anderen Servern ist 0.
- $d(s_1, s_2)$ vermindert sich um $2 \cdot \frac{d}{2} = d$.

$\Rightarrow$

Σ_{DC} vermindert sich um d.

$\Rightarrow$

Der Wert der Potentialfunktion vermindert sich mindestens um d. ∎

Übung: (Verallgemeinerung von DC auf reele Bäume)

Algorithmus DC-Baum

- Bewege alle zum Anfragepunkt benachbarten Server mit gleicher Geschwindigkeit in Richtung Anfragepunkt, bis mindestens einer dieser

erreicht hat. HALT

Beachten Sie, dass zwischendurch ein Server die Eigenschaft 'benachbart' verlieren kann, da ein anderer Server sich auf den Pfad zwischen diesem und dem Anfragepunkt platziert hat. In dem Moment bleibt dieser Server stehen. Falls mehrere Server sich in derselben Position befinden und zum Anfragepunkt benachbart sind, wird nur einer von diesen ausgewählt und bewegt.

Zeigen Sie, dass DC-Baum k-competitive ist.

3.2 Balancierungsalgorithmen

Idee:
Verteile die Last auf den k Servern gleichmäßig.

Algorithmus BAL

- Verwalte für jeden Server s_i, $1 \leq i \leq k$ stets die bisher zurückgelegte Gesamtdistanz D_i.
- Anfrage r
  - Falls sich ein Server auf r befindet, dann tue nichts.
  - Andernfalls wähle Server s_i, der

$$D_i + d(s_i, r)$$

minimiert und platziere gewählten Server

auf r.

Ziel:
Beweis, dass BAL für metrische Räume $M=(S,d)$ mit $|S| = k+1$ k-competitive ist.

Satz 3.3
Sei $M=(S,d)$ ein beliebiger metrischer Raum mit $|S| = k+1$. Dann ist BAL k-competitive bzgl. des k-Server-Problems auf M.

Beweis:

- aus
  S. Irani, A.R. Karlin, Online Computation, Chapter 13 in D. Hochbaum (ed.) "Approximation Algorithms for NP-hard Problems", PWS Publishing Company (1997).

anderer Beweis in Borodin/El Yaniv.

Zu jedem Zeitpunkt existiert genau ein Punkt in S, auf dem sich keiner von BAL's Servern befindet. Diesen Punkt nennen wir Lücke.

Annahme:
Alle Anfragen beziehen sich stets auf die Lücke von BAL.

Eine Anfrage bzgl. eines anderen Punktes kostet BAL nichts und kann lediglich die Kosten von OPT erhöhen. Also können wir obiges o.B.d.A. annehmen.

Sei

$$S := \{1, 2, \ldots, k+1\}.$$

Bezeichne

- k_i diejenige Konfiguration, in der jeder Punkt bis auf den Punkt i einen Server besitzt.
- $opt_t(k_i)$ optimale Kosten, die ersten t Anfragen zu bedienen und dabei in Konfiguration k_i zu enden.

Beobachtung:
Für alle $i, j \in S$ betragen die Kosten um von i nach j zu gelangen $d(i,j)$

$\Rightarrow$

$$opt_t(k_i) \leq opt_t(k_j) + d(i,j) \quad \forall\, i,j \in S.$$

Bezeichne

- σ^t die Folge der ersten t Anfragen.
- h^t die Lücke von BAL nach σ^t

$$D_i^t := \begin{cases} \text{Distanz, die BAL's Server auf } i \text{ nach } \sigma^t \text{ bisher insgesamt zurückgelegt hat} & \text{falls } i \neq h^t \\ \text{undefiniert} & \text{falls } i = h^t \end{cases}$$

Annahme $\Rightarrow$

In der $(t+1)$-ten Anfrage wird der Punkt h^t angefragt.

Beh.: Für $i \neq h^t$ gilt $D_i^t \leq opt_t(k_i)$

Bevor wir die Behauptung beweisen, führen wir den Beweis des Satzes zu Ende.

Es gilt:

$$\begin{aligned} BAL(\sigma^t) &= \sum_{i \neq h^t} D_i^t \\ &\underset{\text{Beh.}}{\leq} \sum_{i \neq h_t} opt_t(k_i) \\ &\leq k \cdot \left[\min_{1 \leq j \leq k+1} \{ opt_t(k_j) \} + \max_{i \neq j} \{ d(i,j) \} \right] \\ &\leq k \cdot OPT(\sigma^t) + b, \end{aligned}$$

wobei $b := k \cdot \max_{i \neq j} \{ d(i,j) \}$ eine Konstante ist.

Da dies für jedes t gilt, beweist dies den Satz. Somit verbleibt nur noch der Beweis der Behauptung.

Bew. d. Beh.: (vollständige Induktion)

Da zu Beginn keiner der Server irgendeine Distanz zurückgelegt hat, ist die Behauptung für $t=0$ trivialerweise erfüllt.

Annahme: Beh. ist erfüllt für alle $\ell \leq t$.

$t \rightsquigarrow t+1$:

Es gilt: $r_{t+1} = h^t$.

Sei min derjenige Punkt, der

$$\min_{i \neq h^t} \{ D_i^t + d(i, h^t) \}$$

minimiert und von BAL gewählt wird.

Dann schickt BAL den Server von min nach h^t.

$\Rightarrow$

- $h^{t+1} = \min$ und
- $D_{h^t}^{t+1} = D_{\min}^t + d(\min, h^t)$.

Ferner gilt:

$$opt_{t+1}(k_{h^t}) = \min_{i \neq h^t} \{ opt_t(k, i) + d(h^t, i) \}$$

Beachten Sie, dass zur Bedienung von r_{t+1} sich ein Server auf h^t befinden muss. Dieser muss dann noch weggeschickt werden, damit die Konfiguration k_{h^t} erreicht wird.

Wegen

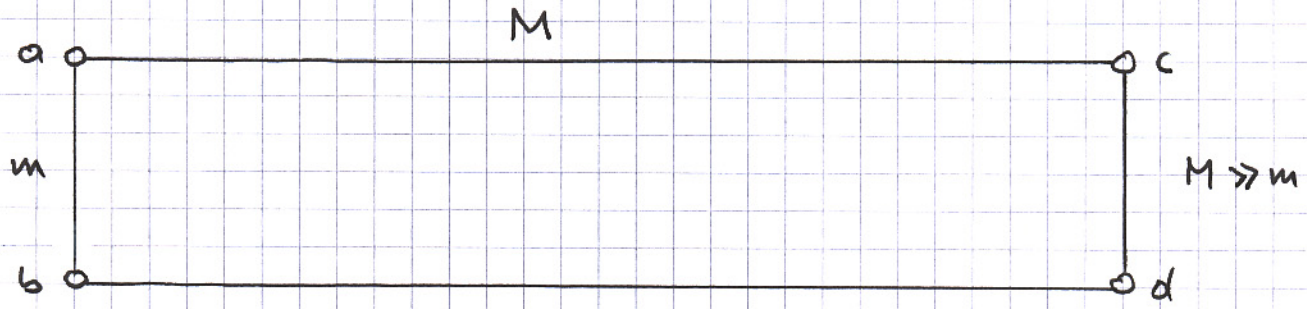
$$opt_t(k_i) \geq D_i^t \quad \text{für } i \neq k^t$$

gilt gemäß der Auswahlregel von BAL

$$opt_{t+1}(k_{k^t}) \geq D_{k^t}^{t+1}$$

Wegen $D_i^{t+1} = D_i^t$ und $opt_{t+1}(k_i) \geq opt_t(k_i)$ für $i \neq k^t$ und $i \neq \min$ folgt die Behauptung für diese i direkt aus der Induktionsannahme. ∎

Falls $|S| > k+1$, dann ist BAL nicht mehr competitive. Betrachten wir hierzu folgendes Beispiel:



$|S| = 4$, $k = 2$, Startkonfiguration $X_0 := \{c, d\}$
Anfragefolge $\sigma = a\,b\,c\,d\,a\,b\,c\,d\,a\,b\,c\,d \ldots$

BAL bewegt die Server stets entlang den horizontalen Linien. Also sind die Kosten für BAL für jede Anfrage M. OPT schickt zur Bearbeitung der ersten Anfrage den auf c befindlichen Server nach a. Diese Anfrage kostet somit M. Jede weitere Anfrage wird

mittels Bewegung entlang einer vertikalen Linie bedient. Somit zahlt OPT einmal M und ansonsten m.

Bemerkung:
Obiges Beispiel würde durch unsere erste Online-Heuristik, die stets die lokalen Kosten optimiert, optimal gelöst werden.

Bisher kennen wir keinen competitiven Online-Algorithmus für allgemeine metrische Räume. Unser Ziel ist es, solchen zu finden.

3.3 Der Arbeitsfunktionenalgorithmus

Zunächst werden wir einige Bezeichnungen einführen und einige grundlegende Eigenschaften beweisen. Seien

- $M := (S, d)$ ein metrischer Raum,
- $S^k := \{ X \subseteq S \mid |X| = k \}$ die Menge der möglichen Konfigurationen,
- $D(X,Y)$, $X, Y \in S^k$ die minimalen Kosten, um von X nach Y zu gelangen. D.h., $D(X,Y)$ sind die Kosten eines minimalen Matchings zwischen den Punkten in X und den Punkten in Y.
- $\sigma := r_1 r_2 \dots r_t \dots r_n$ Anfragefolge. D.h., $r_i \in S$ für $1 \leq i \leq n$.

- $\sigma^t := r_1 r_2 \dots r_t$ die Folge der ersten t Anfragen von σ.
- $X_0 \in S^k$ Startkonfiguration des Online- und des optimalen Offline-Algorithmus.
- $opt_t(\sigma, X)$, $X \in S^k$ minimale Kosten die Anfragefolge σ^t mit Startkonfiguration X_0 und Endkonfiguration X zu bedienen.

Bemerkung:

- $opt_t(\sigma, X)$ kann mit Hilfe dynamischer Programmierung wie folgt berechnet werden:

$$- opt_0(\sigma, X) := D(X_0, X)$$

$$- opt_t(\sigma, X) := \begin{cases} opt_{t-1}(\sigma, X) & \text{falls } r_t \in X \\ \min\limits_{Y : r_t \in Y} \{ opt_{t-1}(\sigma, Y) + D(Y, X) \} & \text{falls } r_t \notin X \end{cases}$$

Wegen $opt_{t-1}(\sigma, X) = opt_{t-1}(\sigma, X) + D(X, X)$ können beide Fälle wie folgt zusammengefasst werden:

$$opt_t(\sigma, X) := \min_{Y : r_t \in Y} \{ opt_{t-1}(\sigma, Y) + D(Y, X) \}.$$

- Die Funktion $opt_t(\sigma, \cdot)$ wird üblicherweis Arbeitsfunktion genannt und häufig mit w_t bezeichnet.

Als nächstes werden wir einige Eigenschaften der Arbeitsfunktion herausarbeiten.

Lemma 3.2

Für alle $0 \leq t \leq n$, $X, Y \in S^k$ gilt:

a) $opt_t(\sigma, X) \leq opt_t(\sigma, Y) + D(Y, X)$.

b) $opt_t(\sigma, X) = \min_{x \in X} \{opt_{t-1}(\sigma, X - x + r_t) + d(r_t, x)\}$.

c) $opt_t(\sigma, X) \geq opt_{t-1}(\sigma, X)$.

d) $r_t \in X \Rightarrow opt_t(\sigma, X) = opt_{t-1}(\sigma, X)$.

e) $opt_t(\sigma, X) = \min_{x \in X} \{opt_t(\sigma, X - x + r_t) + d(r_t, x)\}$.

Beweis:

a) Es gilt

$$\begin{aligned} & opt_t(\sigma, Y) + D(Y, X) \\ &= \min_{Z : r_t \in Z} \{ opt_{t-1}(\sigma, Z) + D(Z, Y) + D(Y, X) \} \\ &\geq \min_{Z : r_t \in Z} \{ opt_{t-1}(\sigma, Z) + D(Z, X) \} \\ &= opt_t(\sigma, X). \end{aligned}$$

b) Es gilt:

(+) $opt_t(\sigma, X) = \min_{Y : r_t \in Y} \{ opt_{t-1}(\sigma, Y) + D(Y, X) \}$.

"$\leq$"

Setze in (+) $Y := X - x + r_t$. Dann gilt
$D(Y, X) = d(r_t, x)$.

"$\geq$"

Sei Y_0 eine Konfiguration, die die rechte Seite von (+) minimiert.

Betrachten wir ein Matching M minimaler Kosten zwischen X und Y_0. Sei $x \in X$ derjenige Punkt, der von M dem Punkt $r_t \in Y_0$ zugeteilt wird. D.h., $(x, r_t) \in M$.

Wir definieren dann

$$Y' := X - x + r_t .$$

Dann gilt da $M = (S, d)$ metrischer Raum und wegen a)

$$\begin{aligned} & opt_{t-1}(\sigma, Y_0) + D(Y_0, X) \\ &= opt_{t-1}(\sigma, Y_0) + D(Y_0, Y') + d(x, r_t) \\ &\geq opt_{t-1}(\sigma, Y') + d(x, r_t) \\ &\geq \min_{x \in X} \{ opt_{t-1}(\sigma, X - x + r_t) + d(x, r_t) \}. \end{aligned}$$

c) Es gilt:

$$\begin{aligned} opt_t(\sigma, x) &= opt_{t-1}(\sigma, Y) + D(Y, X) \text{ für ein } Y \in S^k \\ &\underset{a)}{\geq} opt_{t-1}(\sigma, x). \end{aligned}$$

d)

Teil b) mit $x = r_t$ impliziert

$$opt_t(\sigma, X) \leq opt_{t-1}(\sigma, X)$$

Wegen Teil c) gilt auch

$$opt_t(\sigma, X) \geq opt_{t-1}(\sigma, X)$$

Also gilt

$$opt_t(\sigma, X) = opt_{t-1}(\sigma, X).$$

e) Die Behauptung folgt direkt aus b) und d). ■

Idee des Arbeitsfunktionenalgorithmus:

Annahme:
Nach der Bearbeitung von σ^{t-1} befindet sich der Online-Algorithmus in der Konfiguration $X_{t-1} \in S^k$.

Um die nächste Anfrage r_t zu bearbeiten, muss der Online-Algorithmus in eine Konfiguration $X \in S^k$ mit $r_t \in X$ übergehen.

- Falls $r_t \in X_{t-1}$, dann ist der Online-Algorithmus bereits in solcher Konfiguration und diese Anfrage kostet nichts.
- Falls $r_t \notin X_{t-1}$, dann stellt sich folgende Frage:

Wie wählt der Online-Algorithmus $X \in S^k$ mit $r_t \in X$ aus?

Eine naheliegende Idee ist es, in diejenige Konfiguration X_{opt} überzugehen, in die OPT bei der Bearbeitung von σ^t gelangen würde. Der Übergang von X_{t-1} nach X_{opt} kann sehr teuer sein. Nach Hinzukommen von wenigen weiteren Anfragen könnte OPT die Konfiguration X_{opt} wieder verwerfen, während die Kosten dieses teuren Überganges die Kosten des Online-Algorithmus in die Höhe treiben würde. Demzufolge sind bei der Wahl von $X \in S^k$ mit $r_t \in X$ auch die Übergangskosten von X_{t-1} nach X zu berücksichtigen. Diese Überlegungen führen zu folgendem Algorithmus:

<u>Arbeitsfunktionenalgorithmus (WFA):</u>

- Gehe in eine Konfiguration $X \in S^k$ mit

  a) $r_t \in X$ und

  b) X minimiert

  $$(*) \quad opt_t(\sigma, X) + D(X_{t-1}, X)$$

  über.

Ziel:

Gegenüberstellung der Algorithmen WFA und OPT.

Beide Algorithmen haben dieselbe Startkonfiguration X_0 und erhalten die Anfragefolge σ online.

Annahme:

OPT und WFA haben gerade die Anfrage r_t erhalten.

Verhalten der Algorithmen:

- WFA darf seine Lösung für σ^{t-1} nicht modifizieren und ergänzt diese gemäß obiger Regel zu einer Lösung für σ^t.
- OPT erstellt für σ^t eine optimale Lösung. Dabei kann seine bisherige Lösung für σ^{t-1} modifiziert werden.

  Annahme

  OPT berechnet stets eine optimale Lösung für σ^t, die den längstmöglichen Präfix für σ^{t-1} enthält.

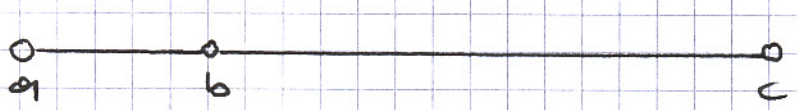
Wir bezeichnen mit OPT_{on} die oben beschriebene Online-Version von OPT.

Frage:

Wann wirkt sich eine Änderung der bisherigen Lösung für σ^{t-1} durch OPT_{on} auf das Verhalten von WFA aus?

Zur Beantwortung dieser Frage betrachten wir zunächst ein Beispiel.

Beispiel:



$d(b,c) = 3 \cdot d(a,b)$

2 Server, Startkonfiguration $X_0 = \{a,b\}$

Wir betrachten zunächst die Anfragefolge

$\sigma := c\,b\,a\,b\,a\,b\,a\,b\,a \ldots$

Wir diskutieren das Verhalten von OPT_{on} und von WFA für wachsende Präfixe der Anfragefolge σ. Dabei symbolisiert $x \to y$ die Servverbewegung vom Punkt x zum Punkt y.

$\sigma^1 = c$:

OPT_{on}:

$b \to c \rightsquigarrow$ Konfiguration $\{a,c\}$

WFA:

$b \to c \rightsquigarrow \{a,c\}$

$\sigma^2 = cb$:

OPT_{on}:

$a \to b \rightsquigarrow \{b,c\}$

WFA:

$a \to b \rightsquigarrow \{b,c\}$

$\sigma^3 = cba$

OPT_{ori}

$b \to a \rightsquigarrow \{a, c\}$

WFA:

$b \to a \rightsquigarrow \{a, c\}$

$\sigma^4 = cbab$

OPT_{ori}:

$a \to b \rightsquigarrow \{b, c\}$

Bemerkung:

Zu einer Lösung mit denselben Kosten würde folgende Modifikation der bisherigen Lösung $b \to c, a \to b, b \to a$ führen:

$b \to c, c \to b \rightsquigarrow \{a, b\}$.

Alle nachfolgenden Anfragen $r_3 r_4 \ldots$ bedient OPT_{on} ohne Serverbewegung. Gemäß unserer „Präfixannahme" wählt OPT_{on} jedoch obige Lösung.

WFA:

$a \to b \rightsquigarrow \{b, c\}$

$\sigma^5 = cbaba$

OPT_{ori}:

Würde OPT_{on} seine bisherige Lösung

$b \to c$, $a \to b$, $b \to a$, $a \to b$ mit $b \to a$

fortsetzen, dann hätte die konstruierte Lösung die Kosten $7 \cdot d(a,b)$. Dahingegen hat die in obiger Bemerkung beschriebene Lösung die Kosten $6 \cdot d(a,b)$.

$\Rightarrow$

OPT_{on} modifiziert seine Lösung wie in obiger Bemerkung beschrieben.

<u>WFA</u>:

Um das Verhalten von WFA zu bestimmen, benötigen wir für beide möglichen Konfigurationen $\{a,b\}$ und $\{a,c\}$ für X_t die optimalen Kosten bei festgelegter Zielkonfiguration zuzüglich den Übergangskosten $D(X_{t-1}, X_t)$.

- $opt_5(r, \{a,b\}) + D(\{b,c\}, \{a,b\})$

  $= 6 \cdot d(a,b) + d(c,a)$

  $= 10 \cdot d(a,b)$

- $opt_5(r, \{a,c\}) + D(\{b,c\}, \{a,c\})$

  $= 7 \cdot d(a,b) + d(b,a)$

  $= 8 \cdot d(a,b)$

$\Rightarrow$

$b \to a \quad \rightsquigarrow \{a,c\}$.

$\sigma^6 = cbababab$

Da OPT_{on} ab jetzt seine Server nicht mehr bewegt, diskutieren wir ab jetzt nur noch WFA.

WFA:

- $opt_6(\sigma, \{a,b\}) + D(\{a,c\}, \{a,b\})$
  $= 6 \cdot d(a,b) + d(c,b)$
  $= 9 \cdot d(a,b)$

- $opt_6(\sigma, \{b,c\}) + D(\{a,c\}, \{b,c\})$
  $= 8 \cdot d(a,b) + d(a,b)$
  $= 9 \cdot d(a,b)$

Heuristik:
Im Fall von Kostengleichheit wählt WFA unter den in Frage kommenden Lösungen eine mit minimalen Übergangskosten aus.

$\Rightarrow$

$a \to b \rightsquigarrow \{b,c\}$

$\sigma^7 = cbababa$

- $opt_7(\sigma, \{a,b\}) + D(\{b,c\}, \{a,b\})$
  $= 6 d(a,b) + d(c,a)$
  $= 10 \cdot d(a,b)$

- $opt_7(\sigma, \{a,c\}) + D(\{b,c\}, \{a,c\})$

  $= 9\, d(a,b) + d(b,a)$

  $= 10\, d(a,b)$

$\Rightarrow$

$b \to a \rightsquigarrow \{a,c\}$

$\underline{\sigma^8 = cbabababab}$:

WFA:

- $opt_8(\sigma, \{a,b\}) + D(\{a,c\}, \{a,b\})$

  $= 6 \cdot d(a,b) + d(c,b)$

  $= 9 \cdot d(a,b)$

- $opt_8(\sigma, \{b,c\}) + D(\{a,c\}, \{b,c\})$

  $= 10\, d(a,b) + d(a,b)$

  $= 11\, d(a,b)$

$\Rightarrow$

$c \to b \rightsquigarrow \{a,b\}$

Alle nachfolgenden Anfragen $r_9 r_{10} \ldots$ bedient WFA ohne Serverbewegung.

Als nächstes modifizieren wir die Anfragefolge σ derart, dass OPT_{on} seine beim Präfix σ^5 vorgenommene Modifikation zurücknimmt und

analysieren den daraus resultierenden Effekt auf WFA.

Betrachten wir hierzu die Anfragefolge $\bar{\sigma}$, die wir aus σ durch Ersetzung von $r_7 = a$ durch $\bar{r}_7 = c$ erhalten.

Beobachtung:
Auf dem Präfix $\bar{\sigma}^6$ verhalten sich OPT_{on} und WFA genauso, wie auf σ^6, da beide Anfragefolgen identisch sind. D.h., die konstruierten Lösungen sind:

OPT_{on}:

$b \to c, c \to b \rightsquigarrow \{a, b\}$

WFA:

$b \to c, a \to b, b \to a, a \to b, b \to a, a \to b$

$\rightsquigarrow$

$\{b, c\}$

Nach Erhalt von $\bar{r}_7 = c$ verhalten sich OPT und WFA wie folgt:

OPT_{on}:
Das Zurückschicken des Servers von b nach c ergäbe die Lösung $b \to c, c \to b, b \to c$, welche die Kosten 9 hat. Modifiziert OPT_{on} stattdessen seine Lösung für $\bar{\sigma}^6$ und verhält sich wie WFA, dann hat die konstruierte

Lösung kosten 8.

$\Rightarrow$

$b \to c,\ a \to b,\ b \to a,\ a \to b,\ b \to a,\ a \to b$

$\rightsquigarrow$ $\{b, c\}$

WFA:

WFA hat alles richtig gemacht und $\bar{\sigma}^7$ genauso wie OPT_{on} gelöst.

Bemerkung:

Wenn die Anfrage bzgl. Punkt c früher erfolgt, dann ist der Effekt derselbe.

Wenn die Anfrage bzgl. Punkt c spät genug erfolgt, dann modifiziert OPT_{on} seine bisherige Lösung nicht. Auch dies möchten wir anhand des Beispiels illustrieren.

Betrachte hierzu die Anfragefolge $\bar{\sigma}$, die wir aus σ durch Ersetzung von $r_9 = b$ durch $\bar{r}_9 = c$ erhalten.

Auch hier verhalten sich OPT_{on} und WFA auf dem Präfix $\bar{\sigma}^8$ genauso wie auf σ^8. Die konstruierten Lösungen sind:

OPT_{on}:

$b \to c,\ c \to b \rightsquigarrow \{a, b\}$

WFA:

$b \to c, a \to b, b \to a, a \to b, b \to a, a \to b, b \to a,$
$c \to b$

$\rightsquigarrow \{a,b\}$

Kosten: $12 \cdot d(a,b)$

Nach Erhalt der Anfrage $r_9 = c$ verhalten sich OPT_{on} und WFA wie folgt:

OPT_{on}:

OPT_{on} schickt den Server auf b nach c und erhält somit die Lösung

$b \to c, c \to b, b \to c \quad \rightsquigarrow \{a,c\}$

Die Kosten dieser Lösung betragen 9.

WFA:

- $opt_9(\sigma, \{a,c\}) + D(\{a,b\}, \{a,c\})$
  $= 9\,d(a,b) + d(b,c)$
  $= 12\,d(a,b)$

- $opt_9(\sigma, \{b,c\}) + D(\{a,b\}, \{b,c\})$
  $= 10\,d(a,b) + d(a,c)$
  $= 14\,d(a,b)$

$\Rightarrow$

$b \to c \quad \rightsquigarrow \{a,c\}$

Kosten: $15\,d(a,b)$

Obiges Beispiel beschreibt Situationen, in denen sich OPT und WFA unterschiedlich verhalten. Wir werden diese nachfolgend kurz charakterisieren.

1) Der Server auf c war derart lange auf dem Punkt c, ohne dass dieser angefragt worden ist, dass es, trotz größeren Bewegungskosten, rückblickend günstiger ist, diesen zu den angefragten Punkten zu bewegen um aufgrund des weiteren Servers dort Serverbewegungen einzusparen. Da die Bewegungskosten von c nach $\{a,b\}$ relativ groß sind, kann WFA der Modifikation von OPT_{on} nicht unmittelbar folgen. Erst dann, wenn die durch die Modifikation bedingte Ersparnis für OPT_{on} so groß wie die Bewegungskosten ist, bewegt WFA den Server von c nach $\{a,b\}$.

2) Falls der Punkt c, nachdem OPT_{on} den Server von c nach $\{a,b\}$ bewegt hat, derart früh angefragt wird, dass es günstiger gewesen wäre, wenn OPT_{on} den Server auf c belassen hätte, dann macht OPT_{on} seine Modifikation wieder rückgängig. Da die durch diese Modifikation bedingte Ersparnis für OPT_{on} noch nicht so groß wie die Bewegungskosten war, ist WFA noch nicht dieser Modifikation gefolgt.

3) Die Bewegung des Servers von c nach b war für OPT_{on} bereits soviel nützlich, dass sich dieser Transfer auch bei einer Anfrage bzgl. c gelohnt hat und auch WFA dieser Modifikation gefolgt ist.

Zum Verständnis der Arbeitsweise von WFA in Relation zur Arbeitsweise von OPT_{on} ist es sinnvoll, weitere etwas größere Beispiele zu analysieren. Um WFA und somit auch die Analyse der Beispiele zu vereinfachen, liegt es nahe, zunächst folgende Frage zu beantworten:

Muss WFA zur Bestimmung einer Konfiguration $\bar{X} \in S^k$ mit $r_t \in \bar{X}$ und $\bar{X}$ minimiert bzgl. allen Konfigurationen $X \in S^k$ mit $r_t \in X$

$$(*) \quad opt_t(\sigma, X) + D(X_{t-1}, X)$$

alle mögliche Konfigurationen betrachten?

Folgendes Lemma beantwortet diese Frage:

Lemma 3.3

Es existiert eine Konfiguration $\bar{X} \in S^k$, $r_t \in \bar{X}$, die (*) minimiert und sich von der aktuellen Konfiguration X_{t-1} höchstens in der Position eines Servers unterscheidet.

Beweis:

Sei Y eine beliebige Konfiguration mit $r_t \in Y$, die $(*)$ minimiert. Sei M ein perfektes Matching minimaler Kosten zwischen X_{t-1} und Y. Sei $x \in X_{t-1}$ derjenige Punkt, der von M dem Punkt $r_t \in Y$ zugeordnet wird. D.h., $(x, r_t) \in M$. Wir definieren dann:

$$\bar{X} := X_{t-1} - x + r_t.$$

Dann gilt:

$$\begin{aligned} & opt_t(\sigma, Y) + D(X_{t-1}, Y) \\ &= opt_t(\sigma, Y) + D(\bar{X}, Y) + d(x, r_t) \\ &\underset{\text{Le 3.2.a)}}{\geq} opt_t(\sigma, \bar{X}) + d(x, r_t) \\ &\underset{\text{Def. v. } \bar{X}}{=} opt_t(\sigma, \bar{X}) + D(X_{t-1}, \bar{X}), \end{aligned}$$

womit das Lemma bewiesen ist. ∎

Der Beweis des obigen Lemmas kann wie folgt interpretiert werden:

Mit Kosten $D(Y, \bar{X}) = D(\bar{X}, Y)$ kann man von der Konfiguration Y in die Konfiguration $\bar{X}$ gelangen. Wegen $opt_t(\sigma, Y) + D(\bar{X}, Y) \geq opt_t(\sigma, \bar{X})$ bildet die Summe aus

- optimale Kosten, um σ^t zu bedienen und in der Konfiguration Y zu enden und

- die Kosten, um von der Konfiguration X_{t-1} in die Konfiguration Y zu gelangen

eine obere Schranke für die Summe aus

- optimale Kosten, um σ^t zu bedienen und in der Konfiguration $\bar{x}$ zu enden und
- die Kosten, um von der Konfiguration X_{t-1} in die Konfiguration $\bar{x}$ zu gelangen.

Also kann WFA die Anfrage r_t durch Bewegen eines Servers von x nach r_t, wobei x folgenden Ausdruck (+) minimiert, bedienen:

$$(+) \quad \min_{x \in X_{t-1}} \left\{ opt_t(\sigma, X_{t-1} - x + r_t) + d(r_t, x) \right\}$$

$$= \quad opt_t(\sigma, X_{t-1})$$

(Lemma 3.2.e)

Danach gilt: $X_t = X_{t-1} - x + r_t$.

Um WFA in Relation zu OPT_{on} besser zu verstehen, werden wir nun ein weiteres Beispiel betrachten.

Beispiel:

Seien

$$M := (\{1, 2, \ldots, k+1\}, d),$$

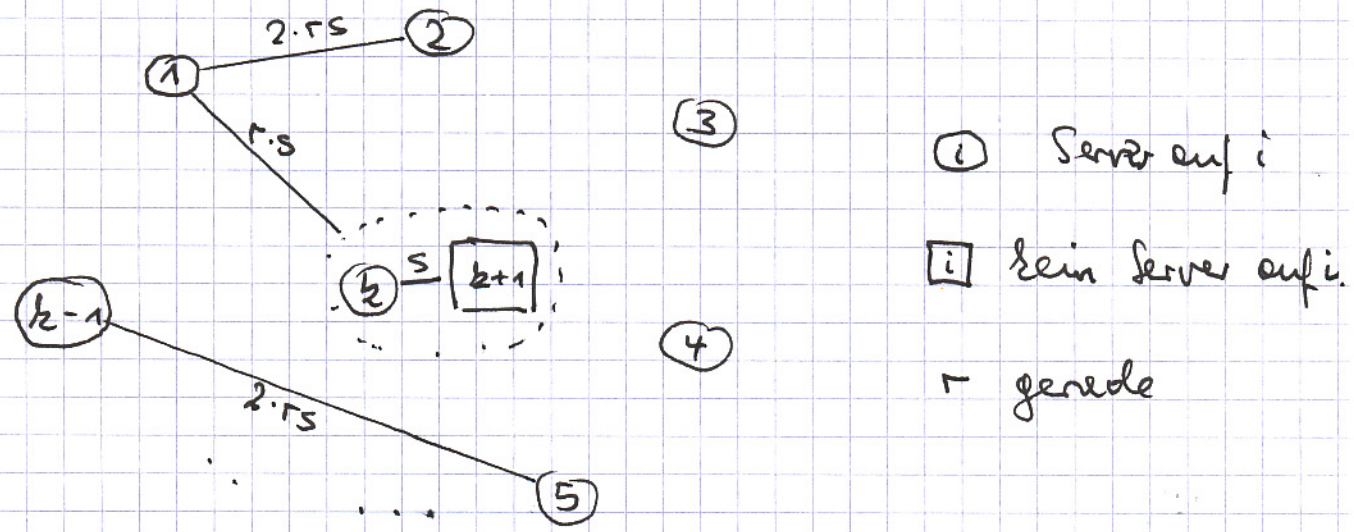
wobei

$$d(i,j) := \begin{cases} s & \text{falls } i = k \text{ und } j = k+1 \\ r \cdot s & \text{falls } i \in \{1,2,\dots,k-1\},\ j \in \{k, k+1\} \\ 2 \cdot r \cdot s & \text{falls } i,j \in \{1,2,\dots,k-1\} \end{cases}$$

Startkonfiguration von WFA und OPT_{on}:

$$X_0 := \{1,2,\dots,k\}.$$

Folgendes Bild beschreibt die Startkonfiguration.



Wir betrachten nun folgende Anfragefolge σ:

$$\sigma := \underbrace{(k+1)\,k\,(k+1)\,k \dots (k+1)\,k}_{(r+1)\text{-mal}}\ 1\ 2\ 3 \dots k-2 \dots$$

Verhalten von OPT_{on}:

Die ersten r Anfragen bedient OPT_{on} mittels Bewegen des einen Servers im Cluster $k, k+1$. Bei der $(r+1)$-te Anfrage stellt OPT_{on} fest, dass es zur Bedienung der $r+1$ Anfragen günstiger ist, bereits bei der ersten Anfrage einen der

Server, die sich auf den Punkten $1, 2, \ldots, k-1$ befinden, nach $k+1$ zu bewegen.

Annahme:

Stets wenn OPT_{on} die Auswahl unter mehreren Servern hat, bewegt OPT_{on} von diesen Servern denjenigen, der sich auf dem Punkt mit der kleinsten Nummer befindet.

$\rightsquigarrow$

$1 \rightarrow k+1 \quad \rightsquigarrow \quad \{2, 3, \ldots, k+1\}$

Die nächsten Anfragen bzgl. k und $k+1$ kann OPT_{on} ohne Serverbewegung bedienen.

Bei der Anfrage bzgl. dem Punkt 1 stellt OPT_{on} fest, dass es günstiger gewesen wäre $2 \rightarrow k+1$ anstatt $1 \rightarrow k+1$ durchzuführen und modifiziert seine Serverbewegung entsprechend. Bei der nächsten Anfrage stellt OPT_{on} fest, dass es günstiger gewesen wäre, $3 \rightarrow k+1$ anstatt $2 \rightarrow k+1$ durchzuführen und modifiziert seine Serverbewegung entsprechend. u.s.w

$\rightsquigarrow$

OPT_{on} bedient σ^{2r+k}, indem die erste Anfrage bzgl. $k+1$ mittels der Serverbewegung $k-1 \rightarrow k+1$ bedient wird. Danach sind keine Serverbewegungen mehr erforderlich.

$\rightsquigarrow$

$k-1 \to k+1 \quad \rightsquigarrow \quad \{1,2,\dots,k-2,k,k+1\}$

Kosten: $r \cdot s$

Verhalten von WFA:

Die ersten $2r$ Anfragen bedient WFA mittels Bewegen des einen Servers im Cluster $k, k+1$. Bei der $(2r+1)$-ten Anfrage gilt:

$$opt_t(\sigma, \{1,2,\dots,k-1,k+1\}) + d(k+1,k) = (2r+1)s$$

Wegen

$$opt_t(\sigma, (\{1,2,\dots,k-1\} \setminus \{i\}) \cup \{k,k+1\}) + d(k+1,i)$$

$$= 2rs \qquad \forall i \in \{1,2,\dots,k-1\}$$

wird WFA einen der Server in $\{1,2,\dots,k-1\}$ nach $k+1$ bewegen.

Annahme:

Stets wenn WFA die Auswahl unter mehreren Servern hat, bewegt WFA denjenigen Server, den auch OPT_{on} bewegt hat.

$\rightsquigarrow$

$1 \to k+1 \quad \rightsquigarrow \quad \{2,3,\dots,k+1\}$

Die nächste Anfrage nach k bedient WFA ohne Serverbewegung. Betrachten wir die anschließende Anfrage bzgl. dem Punkt 1. Es gilt dann

$$opt_t(\sigma, \{1,2,\dots,k-1, k'\}) + d(1,k') = (3r+1)s$$

für alle $k' \in \{k, k+1\}$

und

$$opt_t(\sigma, (\{1,2,\dots,k-1\} \setminus \{i\}) \{k, k+1\}) + d(1,i) = 3rs \quad \forall\, i \in \{2,3,\dots,k-1\}$$

Also bewegt wegen obiger Annahme WFA den Server von 2 nach 1.

$\rightsquigarrow$

$2 \to 1 \quad \rightsquigarrow \quad \{1,3,4,\dots,k-1,k,k+1\}$

Genauso überlegt man sich, dass zur Bedienung der Anfrage 2 den Server 3 nach 2 bewegt.
u.s.w.

$\rightsquigarrow$

WFA bedient σ^{2r+k}, indem die ersten $2r$ Anfragen mittels Bewegen des einen Servers in Cluster $k, k+1$ bedient werden.

Kosten: $2rs$

Die $(2r+1)$-te Anfrage fragt den Punkt $k+1$ an.
WFA bewegt den Server auf 1 nach $k+1$

Kosten: rs

Die nächsten $k-2$ Anfragen $1\,2\,3 \dots k-2$ bedient WFA, indem bei der Anfrage i, $i \in \{1,\dots,k-2\}$

der Server auf $i+1$ nach i bewegt wird.

Kosten pro Anfrage: $2rs$
Gesamtkosten: $(k-2)2rs$

Demzufolge bedient WFA σ^{2r+k} mit Kosten

$(2k-1)\cdot rs$

Frage:
Dies sieht fast wie eine $2k-1$ untere Schranke für das competitive Verhältnis aus. Worin besteht der Denkfehler?

Antwort:
Obige Folge hat konstante Länge. Für eine untere Schranke benötigen wir eine Folge von beliebiger Länge. Zur Konstruktion einer beliebig langen Folge müsste obige Konstruktion wiederholt werden. Hierzu wäre eine Anfrage bezüglich Punkt $k-1$ notwendig, damit einer der Server auf k bzw. $k+1$ nach $k-1$ bewegt wird, so dass sowohl für OPT_{on} als auch für WFA die Ausgangssituation wieder gegeben ist. Demnach gilt aber:

Kosten für OPT_{on} : $2rs$
Kosten für WFA : $k\cdot 2rs$

Demnach wäre WFA eher k-competitive.

Ziel:

Beweis, dass eine Konstante c existiert, so dass für alle Anfragefolgen σ

$$WFA(\sigma) \leq (2k-1)\,OPT(\sigma) + c.$$

Annahme:

OPT und WFA starten in derselben Konfiguration und enden auch in derselben Konfiguration.

Falls WFA in einer anderen Konfiguration als OPT endet, dann könnten wir die Anfragefolge bezüglich Punkten in der Endkonfiguration von OPT verlängern, bis schließlich WFA auch in der Endkonfiguration von OPT gelangt ist.

Effekt:

WFA hat höchstens höhere Kosten während die Kosten von OPT unverändert bleiben.

Demzufolge bedeutet obige Annahme keine Einschränkung.

1. Ziel

Verteilung der optimalen Offline-Kosten auf die einzelnen Anfragen.

Hierzu definieren wir wie folgt die offline Pseudokosten $\widetilde{opt}_t$ der t-ten Anfrage:

$$\widetilde{opt}_t := opt_t(\sigma, X_t) - opt_{t-1}(\sigma, X_{t-1})$$

Es gilt:

$$\sum_{t=1}^{n} \widetilde{opt}_t = opt_n(\sigma, X_n) - opt_0(\sigma, X_0)$$
$$= OPT(\sigma).$$

Bezeichne WFA_t die Kosten von WFA für die t-te Anfrage.

2. Ziel:

Obere Schranke für $WFA_t + \widetilde{opt}_t$.

Annahme:

WFA bewegt Server von x nach r_t um r_t zu bedienen.

Dann gilt:

$$WFA_t + \widetilde{opt}_t = d(x, r_t) + opt_t(\sigma, X_t) - opt_{t-1}(\sigma, X_{t-1})$$
$$\overset{(+)}{=} opt_t(\sigma, X_{t-1}) - opt_{t-1}(\sigma, X_{t-1})$$
$$\leq \max_x \{ opt_t(\sigma, x) - opt_{t-1}(\sigma, x) \}$$

Bezeichne

$$EC_t := \max_x \{ opt_t(\sigma, x) - opt_{t-1}(\sigma, x) \}$$

die <u>erweiterten Kosten</u> der t-ten Anfrage.

Dann gilt:

$$WFA_t + \widetilde{opt}_t \leq EC_t,$$

womit das 2. Ziel erreicht ist.

3. Ziel:

Obere Schranke für EC_t.

Idee (Koutsoupias):

Konstruktion einer Potentialfunktion Φ mit:

a) $EC_t \leq \Phi_t - \Phi_{t-1}$ und

b) $\Phi_n - \Phi_0 \leq (k+1)\, OPT(\sigma) + c$.

Dann gilt:

$$WFA(\sigma) + OPT(\sigma) = \sum_t (WFA_t + \widetilde{opt}_t)$$

$$\leq \sum_t EC_t$$

$$\leq \sum_t (\Phi_t - \Phi_{t-1})$$

$$= \Phi_n - \Phi_0$$

$$\leq (k+1)\, OPT(\sigma) + c$$

$\Leftrightarrow$ $WFA(\sigma) \leq k\, OPT(\sigma) + c$.

$\Rightarrow$ WFA ist k-competitive.

Spezialfall: $|S| = k+1$.

Wähle $\Phi_t := \sum_x opt_t(\sigma, x)$.

Dann gilt:

$$\Phi_t - \Phi_{t-1} = \sum_X \left(opt_t(\sigma, X) - opt_{t-1}(\sigma, X) \right)$$

$$\geq EC_t$$

und

$$\Phi_n - \Phi_0 \leq \sum_X opt_n(\sigma, X)$$

$$\leq \sum_X \left[opt_n(\sigma, X_n) + D(X_n, X) \right]$$

$$\overset{|S| = k+1}{\leq} (k+1) \left[opt_n(\sigma, X_n) + \max_{i,j} d(i,j) \right]$$

$$= (k+1)\, OPT(\sigma) + \underbrace{(k+1) \max_{i,j} d(i,j)}_{:= C}$$

□

4. Ziel:

Charakterisierung der Konfigurationen, in denen die erweiterten Kosten angenommen werden.

Intuition:

Falls eine Konfiguration X die erweiterten Kosten annimmt, dann besitzt X folgende Eigenschaften:

1) Alle Punkte in X müssen genügend weit vom Anfragepunkt r_t entfernt sein.
(Ansonsten ist $opt_t(\sigma, X)$ nicht signifikant größer als $opt_{t-1}(\sigma, X)$.)

2) $opt_{t-1}(\sigma, X)$ darf nicht zu groß sein.

Begründung:
Sei Z eine Konfiguration mit $r_t \in Z$. Dann gilt:

$$opt_{t-1}(\sigma, X) \leq opt_{t-1}(\sigma, Z) + D(Z, X)$$

Beh.:
$$opt_{t-1}(\sigma, X) < opt_{t-1}(\sigma, Z) + D(Z, X)$$

Obige Behauptung besagt, dass für alle Konfigurationen Z mit $r_t \in Z$ $opt_{t-1}(\sigma, X)$ stikt kleiner ist, als $opt_{t-1}(\sigma, Z) + D(Z, X)$.

Bew. d. Beh.:

Annahme: $opt_{t-1}(\sigma, X) = opt_{t-1}(\sigma, Z) + D(Z, X)$

$\Rightarrow$

$$\begin{aligned} opt_t(\sigma, X) &= \min_{Y \mid r_t \in Y} \{ opt_{t-1}(\sigma, Y) + D(Y, X) \} \\ &\leq opt_{t-1}(\sigma, Z) + D(Z, X) \\ &= opt_{t-1}(\sigma, X). \end{aligned}$$

Da Arbeitsfunktionen monoton wachsend sind, folgt somit

$$opt_t(\sigma, X) = opt_{t-1}(\sigma, X).$$

Dies ist ein Widerspruch dazu, dass die erwarteten Kosten bezüglich der Konfiguration X angenommen werden.

□

Obige Intuition motiviert folgende Definition:

Eine Konfiguration A heißt <u>Minimierer für r_t bzgl. $opt_{t-1}(\sigma, \cdot)$</u>, falls A folgenden Ausdruck minimiert:

$$opt_{t-1}(\sigma, X) - \sum_{x \in X} d(r_t, x)$$

Eigenschaft 2 — Eigenschaft 1

<u>Frage:</u>

Warum nicht

$$opt_{t-1}(\sigma, X) - \min \{ d(r_t, x) \mid x \in X \} ?$$

Eine Antwort auf diese Frage erhalten wir, indem wir mit diesem statt obigen Ausdruck die nachfolgende Analyse durchführen.

<u>Lemma 3.4</u> (Dualitätslemma)
Sei die Konfiguration A ein Minimierer für r_t bzgl. $opt_{t-1}(\sigma, \cdot)$. Dann gilt:

a) $EC_t = opt_t(\sigma, A) - opt_{t-1}(\sigma, A)$

(D.h., A ist auch ein <u>Maximierer für r_t bzgl. EC_t</u>)

b) A ist auch ein Minimierer für r_t bzgl. $opt_t(\sigma, \cdot)$.

Bevor wir das Dualitätslemma beweisen, führen wir die Analyse des Arbeitsfunktionenalgorithmus zu Ende.

5. Ziel:

Definition der Potentialfunktion.

Idee:

Verwende das Dualitätslemma zur Definition einer Potentialfunktion Φ, so dass

a) $EC_t \leq \Phi_t - \Phi_{t-1}$ und

b) Ferner soll

$$\Phi_n - \Phi_0 \leq f(k)\, OPT(\sigma) + c$$

für eine Konstante c und möglichst kleinem $f(k)$ sein. (Wir bekommen $f(k) = 2 \cdot k$ hin).

Durchführung:

- Seien

  $U := (u_1, u_2, \ldots, u_k)$ und

  $B_i := (b_{i1}, b_{i2}, \ldots, b_{ik})$ $\quad 1 \leq i \leq k$

  $k+1$ Konfigurationen.

- Sei ψ eine Funktion, definiert durch

$$\psi(opt_t, U, B_1, B_2, \ldots, B_k) := k \cdot opt_t(\sigma, U) + \sum_{\ell=1}^{k} \left(opt_t(\sigma, B_\ell) - \sum_{j=1}^{k} d(u_\ell, b_{\ell j}) \right)$$

- Φ_t ist dann definiert als minimaler Wert von ψ über alle möglichen Konfigurationen $U, B_1, B_2, \ldots, B_k$.

  D.h.,

$$\phi_t := \min_{U, B_1, \dots, B_k} \psi(opt_t, U, B_1, B_2, \dots, B_k).$$

6. Ziel:
Charakterisierung der Konfigurationen $U, B_1, B_2, \dots, B_k$, die ϕ_t definieren.

Lemma 3.5
Sei ϕ_t durch $\tilde{U}, \tilde{B}_1, \tilde{B}_2, \dots, \tilde{B}_k$ definiert. Sei A ein Minimierer für r_t bzgl. $opt_{t-1}(\sigma, \cdot)$.
Dann existiert ein $i \in \{1, 2, \dots, k\}$, so dass

$$\phi_t \geq \psi_t := \psi(opt_t, \tilde{U} - u_i + r_t, \tilde{B}_1, \dots, \tilde{B}_{i-1}, A, \tilde{B}_{i+1}, \dots, \tilde{B}_k).$$

Interpretation:
Wir können annehmen, dass $\tilde{U}$ den letzten angefragten Punkt r_t enthält und eine der Konfigurationen $\tilde{B}_1, \tilde{B}_2, \dots, \tilde{B}_k$ ein Minimierer bzgl. $opt_{t-1}(\sigma, \cdot)$ ist.

Beweis:

Gemäß Definition gilt:

$$\phi_t = \psi(opt_t, \tilde{U}, \tilde{B}_1, \tilde{B}_2, \dots, \tilde{B}_k)$$

$$(*) \quad = k \cdot opt_t(\sigma, \tilde{U}) + \sum_{\ell=1}^{k} \left(opt_t(\sigma, \tilde{B}_\ell) - \sum_{j=1}^{k} d(\tilde{u}_\ell, \tilde{b}_{\ell j}) \right)$$

Lemma 3.2.e) $\Rightarrow$

$$opt_t(\sigma, \tilde{U}) = \min_{\tilde{u} \in \tilde{U}} \{ opt_t(\sigma, \tilde{U} - \tilde{u} + r_t) + d(r_t, \tilde{u}) \}.$$

Sei $\tilde{u}_i \in \tilde{U}$ ein Punkt, bzgl. dem das Minimum angenommen wird. D.h.,

$$opt_t(\sigma, \tilde{U}) = opt_t(\sigma, \tilde{U} - \tilde{u}_i + r_t) + d(r_t, \tilde{u}_i)$$

Eingesetzt in (*) ergibt dies:

$$\Phi_t = k\left[opt_t(\sigma, \tilde{U} - \tilde{u}_i + r_t) + d(r_t, \tilde{u}_i)\right] + \sum_{\ell=1}^{k}\left(opt_t(\sigma, \tilde{B}_\ell) - \sum_{j=1}^{k} d(\tilde{u}_\ell, \tilde{b}_{\ell j})\right).$$

$$= k \cdot opt_t(\sigma, \tilde{U} - \tilde{u}_i + r_t) + \sum_{\substack{\ell=1 \\ \ell \neq i}}^{k}\left(opt_t(\sigma, \tilde{B}_\ell) - \sum_{j=1}^{k} d(\tilde{u}_\ell, \tilde{b}_{\ell j})\right) + opt_t(\sigma, \tilde{B}_i) \underbrace{- \sum_{j=1}^{k}\left(d(\tilde{u}_i, \tilde{b}_{ij}) - d(r_t, \tilde{u}_i)\right)}_{\geq \text{ Δ-Ungl. + Symmetrie } - \sum_{j=1}^{k} d(r_t, \tilde{b}_{ij})}$$

(Beachten wir
Δ-Ungl. $\Rightarrow$ $d(\tilde{u}_i, r_t) + d(r_t, \tilde{b}_{ij}) \geq d(\tilde{u}_i, \tilde{b}_{ij})$)

$$\geq \psi(opt_t, \tilde{U} - \tilde{u}_i + r_t, \tilde{B}_1, \tilde{B}_2, \ldots, \tilde{B}_k).$$

Lemma 3.4.b) $\Rightarrow$

A ist Minimierer für r_t bzgl. $opt_t(\sigma, \cdot)$.

$$\Rightarrow \quad opt_t(\sigma, \tilde{B}_i) - \sum_{j=1}^{k} d(r_t, \tilde{b}_{ij})$$

ist minimal für $\tilde{B}_i = A$.

Also gilt

$$\Phi_t \geq \psi(opt_t, \tilde{u} - \tilde{u}_i + r_t, \tilde{B}_1, \ldots, \tilde{B}_{i-1}, A, \tilde{B}_{i+1}, \ldots, \tilde{B}_k)$$

Als nächstes zeigen wir, dass $\Phi_t - \Phi_{t-1}$ eine obere Schranke für die erweiterten Kosten EC_t der t-ten Anfrage ist.

Lemma 3.6

Für alle t gilt: $EC_t \leq \Phi_t - \Phi_{t-1}$.

Beweis:

Annahme:
Φ_t wird durch die Konfiguration $\tilde{u}, \tilde{B}_1, \tilde{B}_2, \ldots, \tilde{B}_k$ definiert.

Sei $A := \{\tilde{a}_1, \tilde{a}_2, \ldots, \tilde{a}_k\}$ ein Minimierer für r_t bzgl. $opt_{t-1}(\sigma, \cdot)$.

Lemma 3.5 $\Rightarrow \quad \Phi_t \geq \Psi_t$

$$\Rightarrow \quad \Phi_t - \Phi_{t-1} \geq \Psi_t - \overline{\Psi_{t-1}},$$

wobei

$$\bar{\Psi}_{t-1} := \Psi(opt_{t-1}, \tilde{u} - u_i + r_t, \tilde{B}_1, \dots, \tilde{B}_{i-1}, A, \tilde{B}_{i+1}, \dots, \tilde{B}_k)$$

Beachten wir, dass

$$\bar{\Psi}_{t-1} \geq \Phi_{t-1},$$

da Φ_{t-1} als Minimum über allen möglichen $k+1$ Konfigurationen definiert ist. In $\bar{\Psi}_{t-1}$ haben wir $k+1$ spezielle Konfigurationen gewählt, was den Wert höchstens vergrößert.

Es gilt:

$$\begin{aligned}
\Psi_t - \bar{\Psi}_{t-1} &= \Psi(opt_t, \tilde{u} - u_i + r_t, \tilde{B}_1, \dots, \tilde{B}_{i-1}, A, \tilde{B}_{i+1}, \dots, \tilde{B}_k) \\
&\quad - \Psi(opt_{t-1}, \tilde{u} - u_i + r_t, \tilde{B}_1, \dots, \tilde{B}_{i-1}, A, \tilde{B}_{i+1}, \dots, \tilde{B}_k) \\
&= k \cdot [opt_t(\sigma, \tilde{u} - u_i + r_t) - opt_{t-1}(\sigma, \tilde{u} - u_i + r_t)] \\
&\quad + \sum_{\substack{\ell=1 \\ \ell \neq i}}^{k} [opt_t(\sigma, \tilde{B}_\ell) - opt_{t-1}(\sigma, \tilde{B}_\ell)] \\
&\quad + opt_t(\sigma, A) - opt_{t-1}(\sigma, A) \\
&\geq_{\text{Le. 3.2.c)}} opt_t(\sigma, A) - opt_{t-1}(\sigma, A) \\
&=_{\text{Le. 3.4.a)}} EC_t
\end{aligned}$$

■

Nun können wir beweisen, dass WFA $(2k-1)$-competitive ist.

Satz 3.4

WFA ist $(2k-1)$ - competitive.

Beweis:

Es gilt:

$$\begin{aligned} WFA(\sigma) + OPT(\sigma) &= \sum_{t=1}^{n} (WFA_t + \widetilde{opt}_t) \\ &\leq \sum_{t=1}^{n} EC_t \\ &\underset{Le.3.6}{\leq} \sum_{t=1}^{n} (\Phi_t - \Phi_{t-1}) \\ &= \Phi_n - \Phi_0. \end{aligned}$$

Sei $X_n = \{x_1', x_2', \ldots, x_k'\}$. Dann gilt:

$$\begin{aligned} \Phi_n &\leq \psi(opt_n, X_n, X_n, \ldots, X_n) \\ &= k \cdot opt_n(\sigma, X_n) + \sum_{i=1}^{k} \left[opt_n(\sigma, X_n) - \sum_{j=1}^{k} d(x_i', x_j') \right] \\ &= 2k\, opt_n(\sigma, X_n) - \sum_{i=1}^{k} \sum_{j=1}^{k} d(x_i', x_j') \\ &\leq 2k\, opt_n(\sigma, X_n) \\ &= 2k\, OPT(\sigma). \end{aligned}$$

Also gilt

$$\Phi_n - \Phi_0 \leq 2k\, OPT(\sigma) - \Phi_0$$

$$\Rightarrow WFA(\sigma) \leq (2k-1)\, OPT(\sigma) - \Phi_0$$

Beh.: $\Phi_0 \geq -c$ für eine Konstante c.

Somit folgt aus obiger Behauptung direkt, dass WFA $(2k-1)$-competitive ist.

Bew. d. Beh.:

Annahme: $\Phi_0 = \Psi(opt_0, X_0, X_0, \ldots, X_0)$

Dann gilt:

$$\begin{aligned}\Phi_0 &= \Psi(opt_0, X_0, X_0, \ldots, X_0) \\ &= k \cdot opt_0(\sigma, X_0) + \sum_{\ell=1}^{k} \left(opt_0(\sigma, X_0) - \sum_{j=1}^{k} d(x_\ell, x_j) \right), \\ &\quad \text{wobei } X_0 := (x_1, x_2, \ldots, x_k). \\ &= - \sum_{\ell=1}^{k} \sum_{j=1}^{k} d(x_\ell, x_j),\end{aligned}$$

was eine Konstante ist, die nur von der Startkonfiguration und nicht von der Anfragefolge σ abhängt.

Es verbleibt noch der Beweis der obigen Annahme.

Übung. ■

Es fehlt noch der Beweis des Dualitätslemmas

Lemma 3.4 (Dualitätslemma)
Sei die Konfiguration A ein Minimierer für σ_t bzgl. $opt_{t-1}(\sigma, \cdot)$. Dann gilt

a) $EC_t = opt_t(\sigma, A) - opt_{t-1}(\sigma, A)$
b) A ist auch ein Minimierer für σ_t bzgl. $opt_t(\sigma, \cdot)$.

Beweis: (Michel X. Goemans course notes)
über Homepage beziehbar.

Folgendes Lemma wird für den Beweis benötigt:

Lemma 3.7

$\forall\, t\ \forall\, A,B\ \forall\, a \in A\ \exists\, b \in B$ mit

$$opt_t(\sigma, A) + opt_t(\sigma, B) \geq opt_t(\sigma, A-a+b) + opt_t(\sigma, B-b+a).$$

Bevor wir Lemma 3.7 beweisen, beweisen wir unter Verwendung dieses Lemmas das Dualitätslemma.

a) Zu zeigen: $\forall\, B$ gilt

$$opt_t(\sigma, A) - opt_{t-1}(\sigma, A) \geq opt_t(\sigma, B) - opt_{t-1}(\sigma, B)$$

$$\Leftrightarrow opt_t(\sigma, A) + opt_{t-1}(\sigma, B) \geq opt_t(\sigma, B) + opt_{t-1}(\sigma, A)$$

$\overset{\text{Le 3.2.b)}}{\Leftrightarrow}$ $\forall\, a \in A$ gilt:

$$opt_{t-1}(\sigma, A-a+r_t) + d(r_t, a) + opt_{t-1}(\sigma, B)$$

$$\geq \min_{b \in B} \{ opt_{t-1}(\sigma, B-b+r_t) + d(r_t, b) + opt_{t-1}(\sigma, A) \}$$

Da A ein Minimierer für r_t bzgl. $opt_{t-1}(\sigma, \cdot)$ ist, gilt:

$$opt_{t-1}(\sigma, A) - \sum_{x \in A} d(r_t, x) \leq opt_{t-1}(\sigma, A-a+b) - \sum_{y \in A-a+b} d(r_t, y).$$

$$\Leftrightarrow opt_{t-1}(\sigma, A) + d(r_t, b) \leq opt_{t-1}(\sigma, A-a+b) + d(r_t, a)$$

$$\Leftrightarrow d(r_t, a) \geq opt_{t-1}(\sigma, A) + d(r_t, b) - opt_{t-1}(\sigma, A-a+b)$$

Also genügt es folgendes zu beweisen:

$$opt_{t-1}(\sigma, A-a+r_t) + opt_{t-1}(\sigma, B)$$

$$\geq \min_{b \in B} \{ opt_{t-1}(\sigma, B-b+r_t) + opt_{t-1}(\sigma, A-a+b) \}$$

Wegen $r_t \in A - a + r_t$ und Lemma 3.7 gilt:

$\exists\, b \in B$ mit

$$opt_{t-1}(\sigma, A-a+r_t) + opt_{t-1}(\sigma, B)$$

$$\geq opt_{t-1}(\sigma, A-a+r_t-r_t+b) + opt_{t-1}(\sigma, B-b+r_t)$$

$$\Leftrightarrow\ opt_{t-1}(\sigma, A-a+r_t) + opt_{t-1}(\sigma, B)$$

$$\geq opt_{t-1}(\sigma, B-b+r_t) + opt_{t-1}(\sigma, A-a+b)$$

q.e.d.

b) zu zeigen: $\forall B$ gilt:

$$opt_t(\sigma, A) - \sum_{a \in A} d(r_t, a) \leq opt_t(\sigma, B) - \sum_{b \in B} d(r_t, b)$$

$\Leftrightarrow$ (Le 3.2.5) $\forall\, y \in B$ gilt:

$$\min_{x \in A} \{ opt_{t-1}(\sigma, A-x+r_t) + d(r_t, x) - \sum_{a \in A} d(r_t, a) \}$$

$$\leq opt_{t-1}(\sigma, B-y+r_t) + d(r_t, y) - \sum_{b \in B} d(r_t, b)$$

Betrachten wir $y \in B$ beliebig, aber fest.

Da A ein Minimierer für r_t bzgl. $opt_{t-1}(\sigma, \cdot)$ gilt:

$$opt_{t-1}(\sigma, A) - \sum_{a \in A} d(r_t, a) \leq opt_{t-1}(\sigma, B-y+x) - \sum_{b \in B-y+x} d(r_t, b).$$

Nach Einsetzung von

$$\sum_{b \in B} d(r_t, b) - d(r_t, y) + d(r_t, x) \text{ für } \sum_{b \in B-y+x} d(r_t, b)$$

erhalten wir:

$$opt_{t-1}(\sigma, A) - \sum_{a \in A} d(r_t, a)$$

$$\leq opt_{t-1}(\sigma, B-y+x) - \left(\sum_{b \in B} d(r_t, b) - d(r_t, y) + d(r_t, x) \right)$$

$\Leftrightarrow$

$$(*) \quad - opt_{t-1}(\sigma, B-y+x) + opt_{t-1}(\sigma, A) + d(r_t, x) - \sum_{a \in A} d(r_t, a)$$
$$\leq d(r_t, y) - \sum_{b \in B} d(r_t, b)$$

Wegen $r_t \in B - y + r_t$ impliziert Lemma 2.7:

$\exists\, x \in A$ mit

$$(**) \quad opt_{t-1}(\sigma, B-y+x) + opt_{t-1}(\sigma, A-x+r_t)$$
$$\leq opt_{t-1}(\sigma, B-y+r_t) + opt_{t-1}(\sigma, A)$$

Addition von $(*)$ und $(**)$ gefolgt von Subtraktion von $opt_{t-1}(\sigma, A)$ auf beiden Seiten ergibt:

$$opt_{t-1}(\sigma, A-x+r_t) + d(r_t, x) - \sum_{a \in A} d(r_t, a)$$

$$\leq opt_{t-1}(\sigma, B-y+r_t) + d(r_t, y) - \sum_{b \in B} d(r_t, b)$$

■

Beweis von Lemma 3.7: (durch Induktion über t)

t = 0:

Sei X_0 die Startkonfiguration. Dann gilt:

$$opt_0(\sigma, X) = D(X_0, X) \quad \forall \text{ Konfigurationen } X.$$

Seien M_A bzw. M_B zwei Matchings minimaler Kosten zwischen A und X_0 bzw. B und X_0

Für $a \in A$ beliebig, aber fest betrachten wir

$a_0 \in X_0$ mit $(a, a_0) \in M_A$ und
$b \in B$ mit $(b, a_0) \in M_B$.

Dann gilt:

$$opt_0(\sigma, A) + opt_0(\sigma, B)$$

$$= D(X_0, A) + D(X_0, B)$$

$$\overset{\text{wahl von } a_0 \text{ und } b}{=} D(X_0 - a_0, A - a) + D(X_0 - a_0, B - b) + d(a_0, a) + d(a_0, b)$$

$$\geq D(X_0, A - a + b) + D(X_0, B - b + a)$$

$$= opt_0(\sigma, A - a + b) + opt_0(\sigma, B - b + a)$$

t > 0:

Annahme: Die Behauptung gilt für t-1

t-1 ↝ t:

Sei $a \in A$ beliebig, aber fest.

Lemma 3.2.5 $\Rightarrow$

$$(+)\begin{cases} \exists x \in A : opt_t(\sigma, A) = opt_{t-1}(\sigma, A - x + r_t) + d(r_t, x) \\ \exists y \in B : opt_t(\sigma, B) = opt_{t-1}(\sigma, B - y + r_t) + d(r_t, y) \end{cases}$$

Wir unterscheiden zwei Fälle:

1. Fall: $x = a$

Setze $b := y$.

Definition von opt_t $\Rightarrow$

$opt_t(\sigma, A - a + b) \leq opt_{t-1}(\sigma, A - a + r_t) + d(r_t, b)$
und
$opt_t(\sigma, B - b + a) \leq opt_{t-1}(\sigma, B - b + r_t) + d(r_t, a)$.

Die Addition der beiden Ungleichungen unter Berücksichtigung von (+) ergibt:

$$opt_t(\sigma, A - a + b) + opt_t(\sigma, B - b + a) \leq opt_t(\sigma, A) + opt_t(\sigma, B)$$

q.e.d.

2. Fall: $x \neq a$

Dann gilt $a \in A - x$.

Die Anwendung der Induktionshypothese auf $A - x + r_t$ und $B - y + r_t$ ergibt:

$\exists\, b \in B - y + r_t$ mit

$$(++)\quad \begin{aligned} &opt_{t-1}(\sigma, A - x + r_t) + opt_{t-1}(\sigma, B - y + r_t) \\ &\geq opt_{t-1}(\sigma, A - x + r_t - a + b) + opt_{t-1}(\sigma, B - y + r_t - b + a) \end{aligned}$$

Ferner gilt wegen Lemma 3.2.6):

$$(+++)\quad \begin{cases} opt_t(\sigma, A - a + b) \leq opt_{t-1}(\sigma, A - a + b - x + r_t) + d(r_t, x) \\ opt_t(\sigma, B - b + a) \leq opt_{t-1}(\sigma, B - b + a - y + r_t) + d(r_t, y) \end{cases}$$

Insgesamt haben wir gezeigt:

$$\begin{aligned} &opt_t(\sigma, A) + opt_t(\sigma, B) \\ &\underset{(+)}{=} opt_{t-1}(\sigma, A - x + r_t) + d(r_t, x) + opt_{t-1}(\sigma, B - y + r_t) + d(r_t, y) \\ &\underset{(++)}{\geq} opt_{t-1}(\sigma, A - x + r_t - a + b) + d(r_t, x) \\ &\qquad + opt_{t-1}(\sigma, B - y + r_t - b + a) + d(r_t, y) \\ &\underset{(+++)}{\geq} opt_t(\sigma, A - a + b) + opt_t(\sigma, B - b + a) \end{aligned}$$

q.e.d.

■