Ludwig Maximilians Universität München
Institut für Informatik
Lehr- und Forschungseinheit für Datenbanksysteme

Skript zur Vorlesung

Managing and Mining Multiplayer Online Games

im Sommersemester 2015

Kapitel 8: Zeitliche Verhaltensmodellierung

http://www.dbs.ifi.lmu.de/cms/VO_Managing_Massive_Multiplayer_Online_Games

Kapitelüberblick

- Verhalten und Sequenzen
- Vergleiche zwischen Sequenzen
- Finden häufiger Teilsequenzen
- Markow-Ketten
- Hidden Markow-Ketten
- Zeitreihen und Feature-Transformationen
- Vergleich zwischen Zeitreihen
- Poisson-Prozesse

Spielerverhalten

Beispiele für Spielerverhalten

- Abfolge der Züge im Schach
- Folge von Bewegungen, Aktionen und Interaktionen in einem MMORPG
- Folge von Aufträgen an die eigenen Einheiten in RTS Games

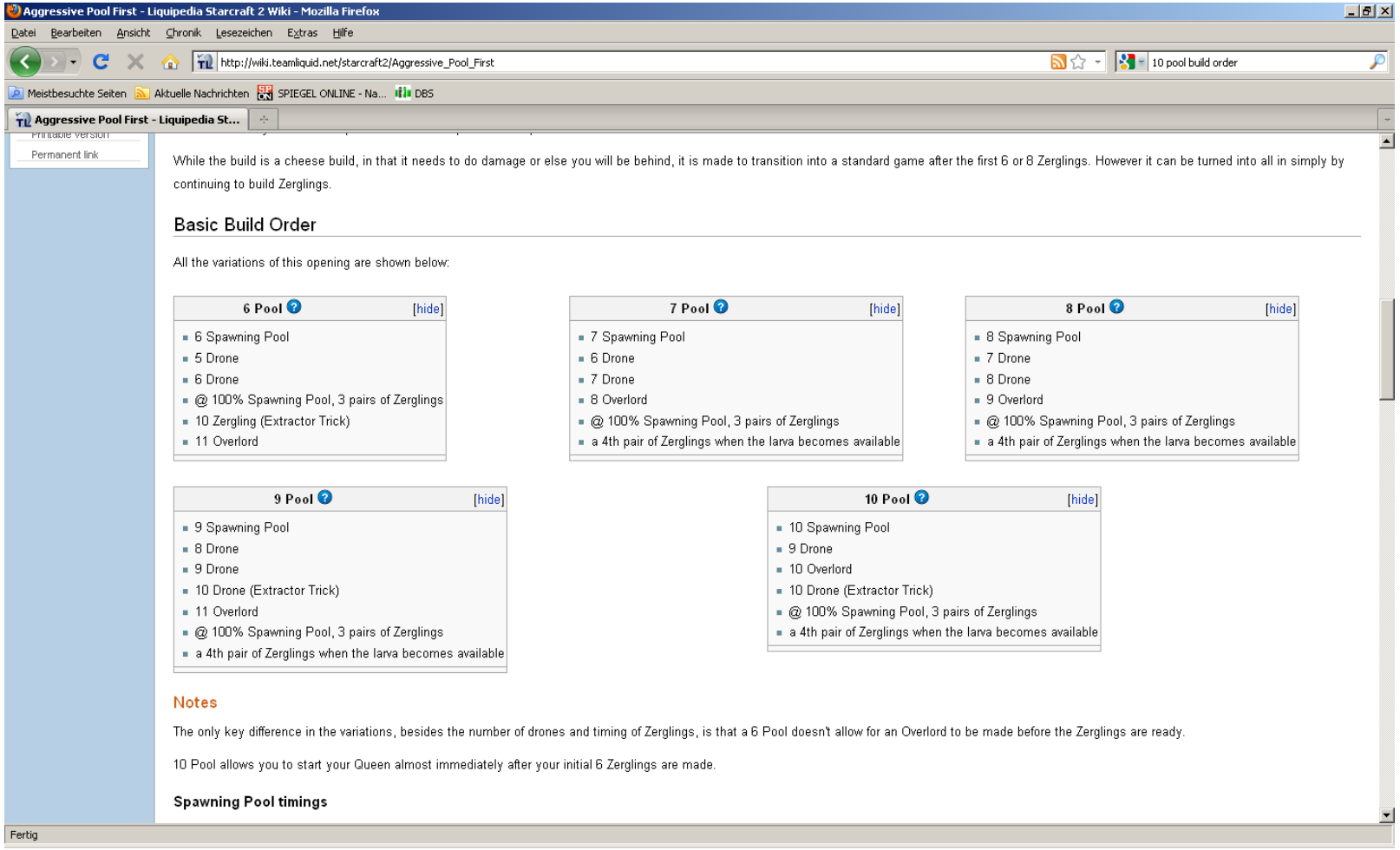
- Abstrakt besteht Verhalten aus einer Folge von möglichen Aktionen
- Einfachstes Modell für Verhalten sind Strings oder Sequenzen.

Definition: Sei $A=\{A_1, \ldots, A_n\}$ ein endliches Alphabet von n möglichen Spieleraktionen. Dann heißt der l-Tupel $(a_1, \ldots, a_l) \in A \times \ldots \times A$ Sequenz der Länge l über **A**.

Bemerkung:

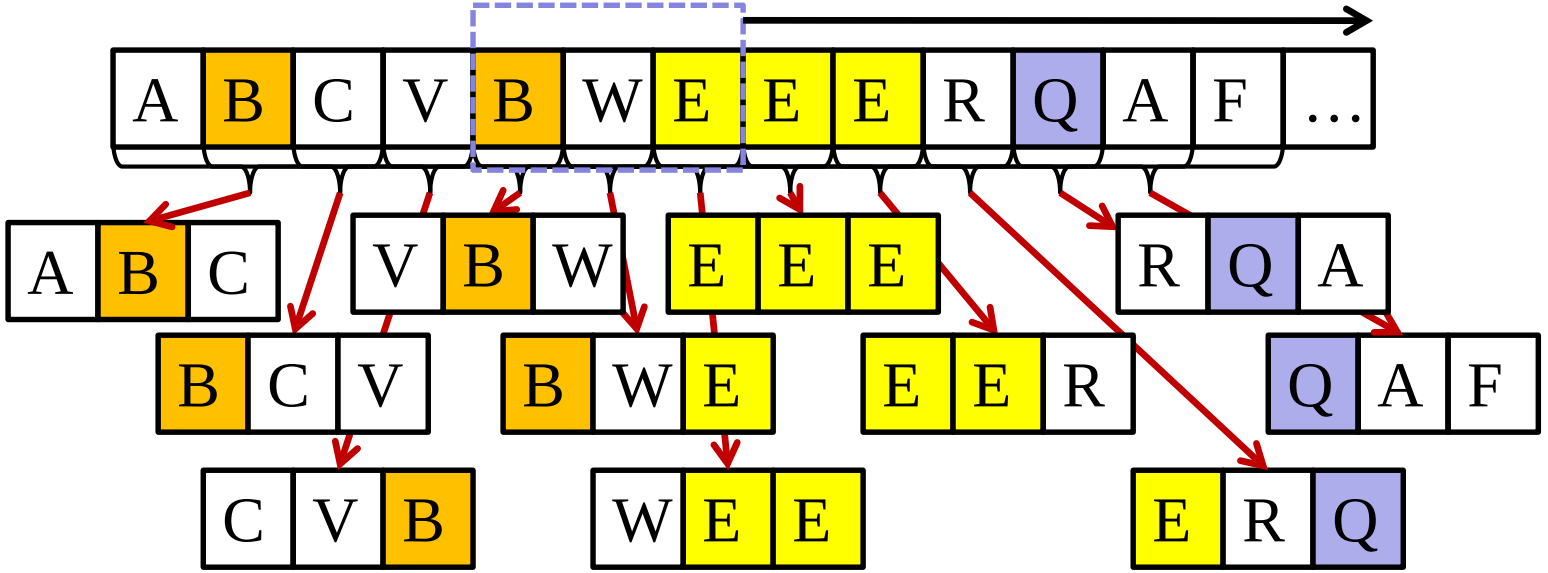
- Modell beschreibt erstmal nur Beobachtung und unterscheidet nicht zwischen möglichen und unmöglichen Sequenzen.
- Modell vernachlässigt die Zeit zwischen den Aktionen.

Beispiel: SC II Zerg Rushes



Subsequenzen und Aufteilung der Beobachtung

- Wann wird welcher Spieler wie lange beobachtet?
- Je länger ein Spieler beobachtet wird, desto unwahrscheinlicher wird es, das ein anderer Spieler sich sehr ähnlich verhält.
- Zum Finden typischer Verhaltensmuster wird eine Sequenz in der Regel in Subsequenzen aufgeteilt.
- Aufteilen durch Windowing.
  Schiebe ein Fenster der Länge k über die Sequenz und betrachte alle Teilsequenzen: im Beispiel ist k = 3



Subsequenzen und Aufteilung der Beobachtung

Problem: Eine Sequenz der Länge *l* hat *l – (k-1)* k-elementige Subsequenzen und viele davon sind irrelevant.

Idee: Nur Sequenzen, die in einer bestimmten Häufigkeit auftreten sind interessant.

Frequent Subsequenz Mining

Finde alle Teilsequenzen einer Sequenz-Datenbank, die häufiger als *minsup* vorkommen. (vgl. Frequent Itemset Mining)

$\Rightarrow$ Länge der Sequenz ist beliebig.

$\Rightarrow$ Suchraum ist größer als beim Itemset Mining.
(mehrfaches Auftreten eines Elements und Reihenfolge)

Frequent Subsequence Mining

- Häufigkeit *fr(S,G)* von S in der Sequence G:
  Zähle Vorkommen von *S* in *G*.
- relative Häufigkeit von S:

$$\varphi(S,G) = \frac{fr(S,G)}{|G| - |S| - 1}$$

- Sequenzbeschreibung von G:

$$\delta(\mathrm{G}) = \{(\mathrm{S}, \varphi(\mathrm{S},\mathrm{G})) \mid \mathrm{S} \in \mathrm{G}\}$$

- Mining Sequential Patterns ist gut erforscht
  => viele Lösungsansätze und Algorithmen

Suffix Bäume

Eigenschaften eines Suffix Baums *ST* über dem Alphabet *A* für Sequenz *G* mit $|G| = n$:

- Um Ambivalenzen auszuschließen, werden Worte mit einem Terminalzeichen ($\notin A$) gepadded, üblicherweise *$*.
- *ST* hat genau $n+1$ Blattknoten, die von *0* bis *n* nummeriert werden, auf dem Weg von der Wurzel zum Blatt mit der Nummer *i* ist das Suffix der Länge $n-i$ abgelegt.
- Kanten repräsentieren Elemente von A∪{$} (unkompromierte Form) bzw. nicht-leere Teilsequenzen von A∪{$}
- Kanten, die vom gleichen Startknoten asugehen, müssen mit unterschiedlichen Elementen von *A* beginnen.

Aufbau in *O*(Stringlänge), Suche in *O*(Querystringlänge)

Suffix Trees

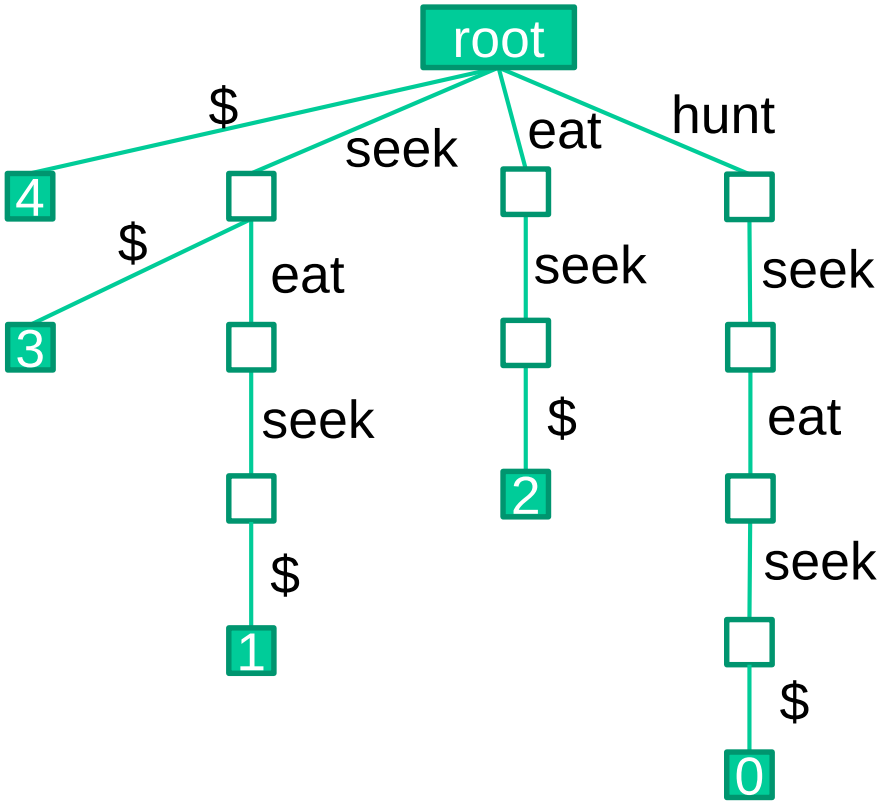
- Beispiel: Alphabet A ={eat, hunt, seek, flee, defend}
- Einfügen:

  S_1 = (seek, hunt, eat, seek)
  S_2 = (seek, flee,hunt)

Suffix Trees

- Beispiel: Alphabet A ={eat, hunt, seek, flee, defend}
- Einfügen:

  S_1 = (hunt, seek, eat, seek) → (hunt, seek, eat, seek, $)
  S_2 = (seek, flee, hunt) → (seek, flee, hunt, $)



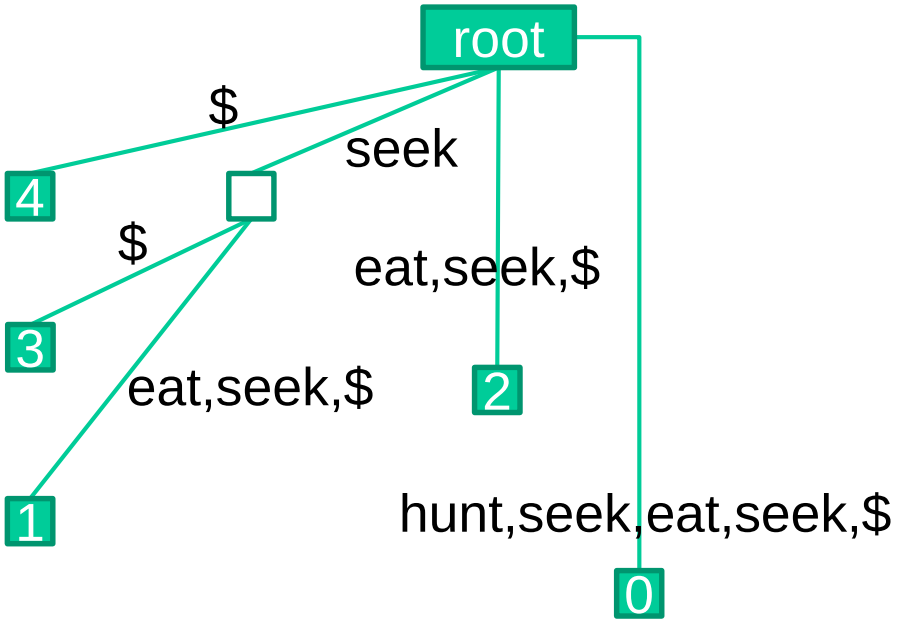
unkomprimierte Variante:
jede Kante ist mit einem Element von A∪{$} beschriftet

komprimierte Variante:
fasse Subpfade ohne Abzweigung zu einer Kante zusammen

Suffix Trees

- Beispiel: Alphabet A ={eat, hunt, seek, flee, defend}
- Einfügen:

  S_1 = (hunt, seek, eat, seek) → (hunt, seek, eat, seek, $)
  S_2 = (seek, flee, hunt) → (seek, flee, hunt, $)



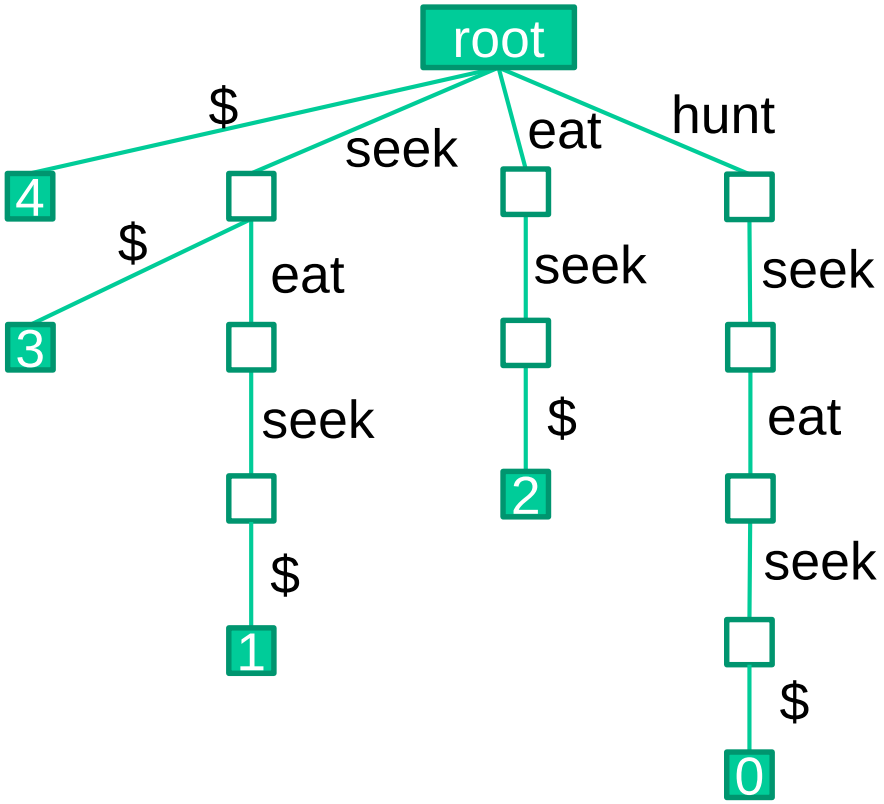
unkomprimierte Variante:
jede Kante ist mit einem Element von A∪{$} beschriftet

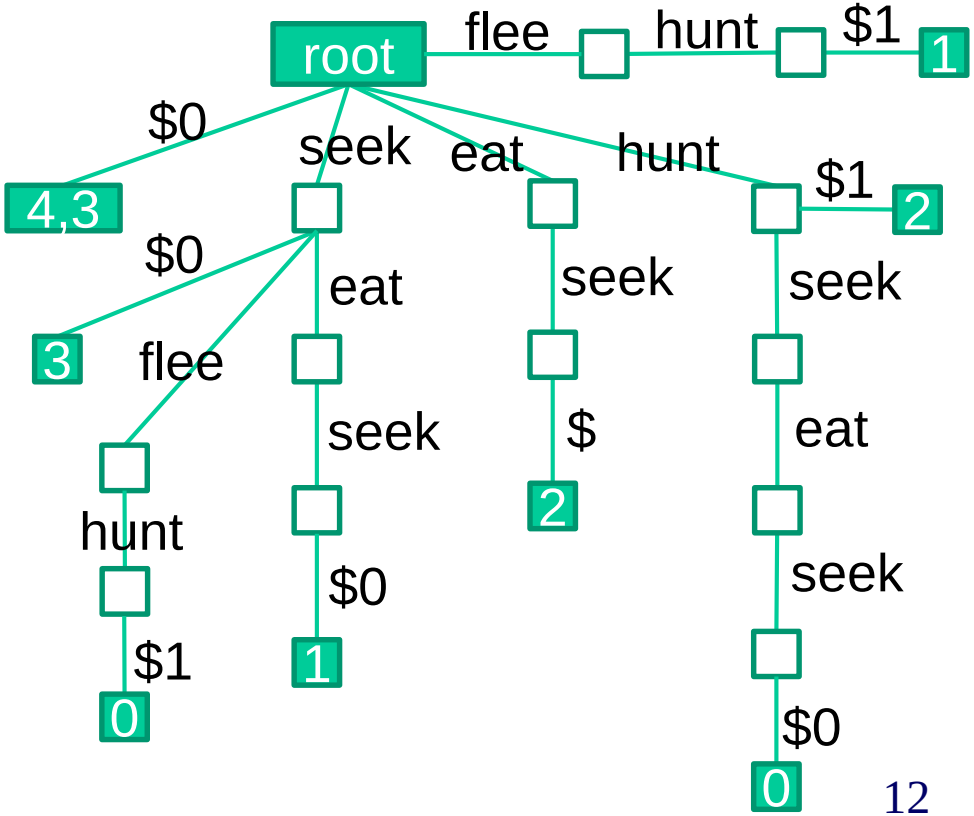
komprimierte Variante:
fasse Subpfade ohne Abzweigung zu einer Kante zusammen

Suffix Trees

- Beispiel: Alphabet A ={eat, hunt, seek, flee, defend}
- Einfügen:

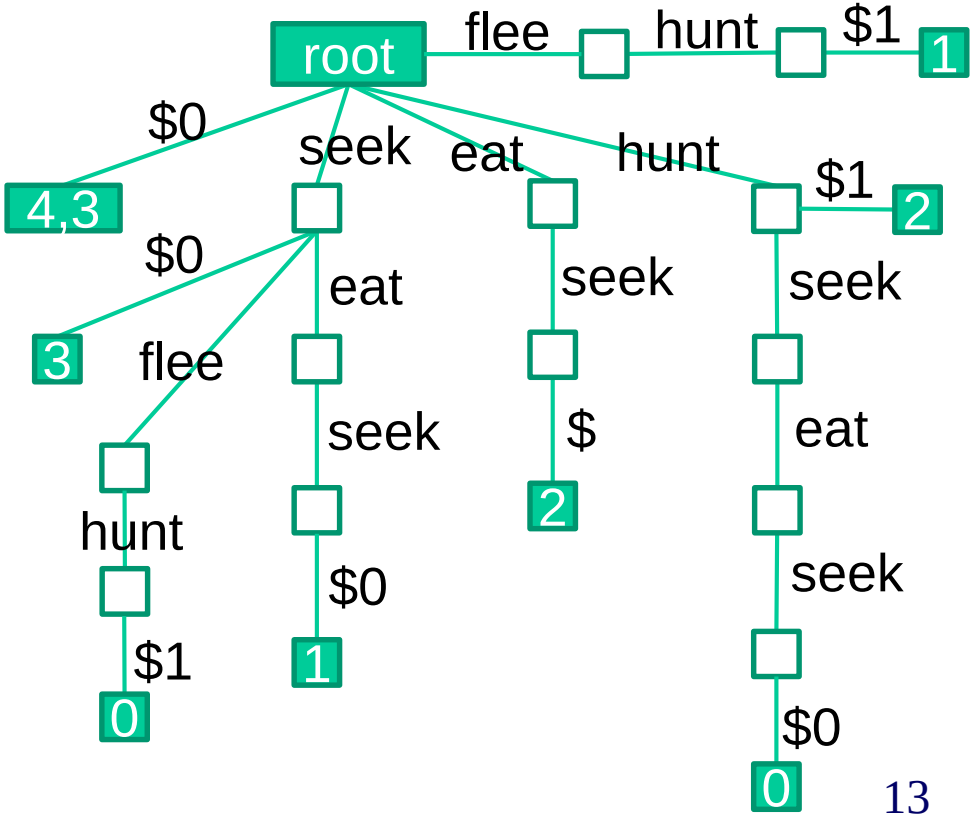
  S_1 = (hunt, seek, eat, seek) → (hunt, seek, eat, seek, $)
  S_2 = (seek, flee, hunt) → (seek, flee, hunt, $)





Suffix Trees

- Beispiel: Alphabet A ={eat, hunt, seek, flee, defend}
- Beispielanfragen:
  - Ist *q* ein Suffix?
  - Ist *q* ein Substring?
  - Wie oft kommt *q* vor?



Suffix Trees

- Beispiel: Alphabet A ={eat, hunt, seek, flee, defend}
- Beispielnfragen:

– Ist *q* ein Suffix?

=> folge Pfad (*q$*) von Wurzel aus,

falls Ankunft bei Blatt, dann Suffix

– Ist *q* ein Substring?

=> folge Pfad (*q*) von Wurzel aus,

falls Abarbeiten möglich,

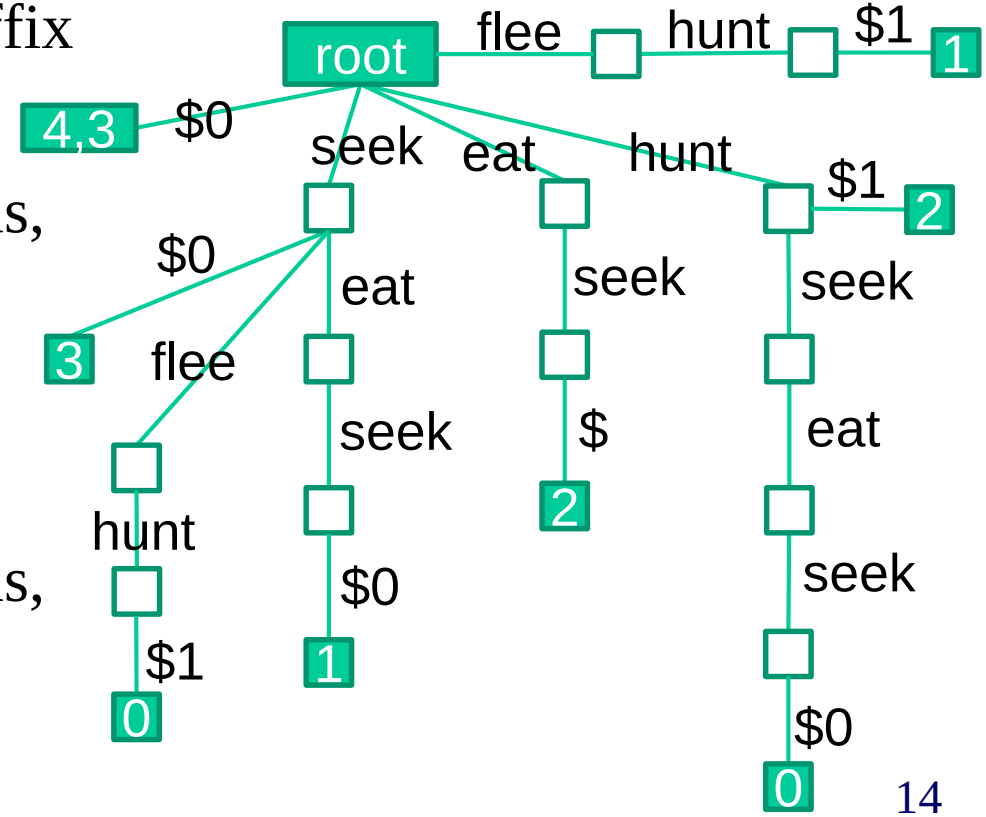
dann Substring

– Wie oft kommt *q* vor?

=> folge Pfad (*q*) von Wurzel aus,

#Blätter unter Endknoten

= #Vorkommnisse



Das Interessante an Teilsequenzen

- interessant ≠ häufig
- **Beispiel**: select drones, collect crystals, train drone, …
  Aber: Die ersten Aktionen in SC II sind fast immer identisch.
- die Anzahl häufiger Teilsequenzen kann sehr groß sein.
- die meisten davon beschreiben Standard-Game-Plays.
- Interessantheit sollte immer im Bezug auf ein weiteres Merkmal untersucht werden:
  - Karte (Bezug auf eine Örtlichkeit)
  - Spieler (Bezug auf das Individuum)
  - Strategie (Bezug auf Situation)
  - Kombination mehrerer Bezüge (Karte und Strategie …)

Maß für die Interessantheit

- Verwendung von Maßen für die Korrelation:
- Finde eine Zielvariable: z.B player_id
- Finde interessante Ereignisse: z.B. Boss-Kämpfe, Flaggenträger, …
- Finde Orte, die ähnliches Verhalten auslösen: Spawn-Punkte für Monster, Flaggenabgabepunkte, Ort einer Bossbegegnung …
- Berechnung durch zum Beispiel:
  - **Mutual Information**

$$\mathrm{MI}(\mathrm{S}, \mathrm{Player\_ID}) = \sum_{\mathrm{P} \in Players} \sum_{S \in \{\mathrm{S}_1, \overline{\mathrm{S}_1}\}} \Pr[S, P] \cdot \log \frac{Pr[\mathrm{S}, \mathrm{P}]}{\Pr[\mathrm{S}] \cdot \Pr[P]}$$

  - **Odds Ratio**

$$\mathrm{oddsR}_{\mathrm{S}}(\mathrm{G}_1, \mathrm{G}_2) = \frac{\varphi(\mathrm{S}, \mathrm{G}_1)}{\varphi(\mathrm{S}, \mathrm{G}_2)}$$

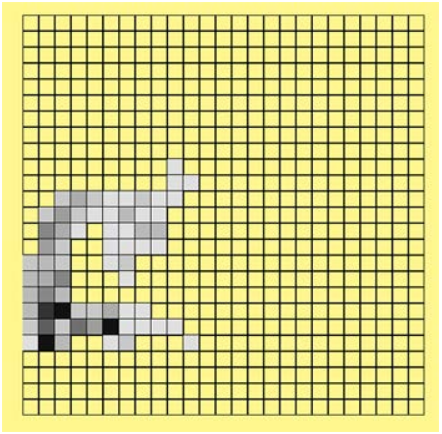
Verwendung von Frequent Subsequences

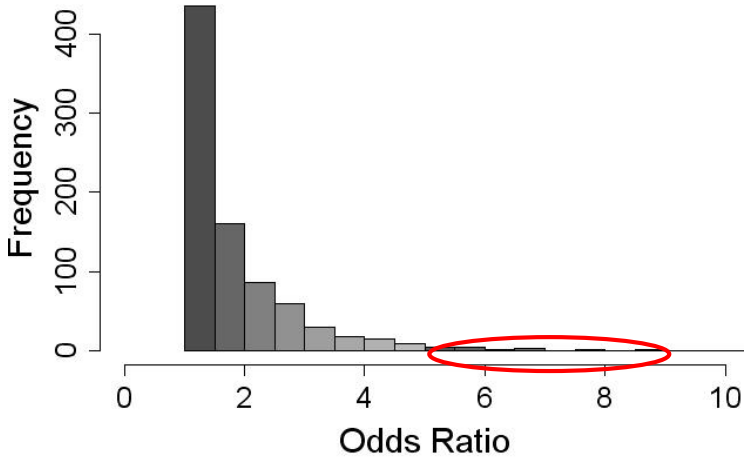
- **Identifikation von Spielern**: Verwende das Vorkommen der k-„interessantesten“ Teilsequenzen als Dimensionen eines Vektorraums.

  (hier Interessantheit = höchste MI mit der Player_id)

  => Beschreibe Spieler als Vektor der beobachteten Teilsequenzen.

- **Suche ortsspezifisches Verhalten**: Vergleiche die Häufigkeit von Handlungen auf der gesamten Karte mit der Anzahl der Handlungen in einem bestimmten Gebiet. (Odds-Ratio)





Vergleiche zwischen 2 Sequenzen

Gegeben: Alphabet A und eine Sequenzdatenbank
$DB \subseteq \{(x_1, \ldots, x_k) \mid k \in IN \wedge x_i \in A \text{ für } 1 \leq i \leq k\}$.

Gesucht: Die Ähnlichkeit von $S1, S2 \in DB$.

- **Hamming Distanz**: Anzahl der unterschiedlichen Einträge aller Positionen.

  Für 2 Sequenzen mit |S1|=|S2|=k:

$$Sim_{Ham}(S1,S2) = \sum_{i=0}^{k} \begin{cases} 0 & if \quad s_{1,i} = s_{2,i} \\ 1 & else \end{cases}$$

  Anmerkung: Bei unterschiedlich langen Sequenzen auffüllen der kürzeren Sequenz mit dem Gap Symbol „-“.

  Beispiel: S1 = (A,B,B,A,B) und S2 = (A,A,A,A,A)

$$\left.\begin{matrix}(A,B,B,A,B)\\(A,A,A,A,A)\end{matrix}\right\} Sim_{Ham}(S1,S2)=2$$

Levenshtein Distanz

- **Hamming Distanz**: Berechnet minimale Kosten, um S1 in S2 zu verwandeln. Dabei sind nur Substitutionen der einzelnen Elemente erlaubt. (Aus B mache A.)

- **Idee**: Erweitere die erlaubten Änderungsoperationen durch Löschen und Einfügen von Symbolen.
- **Levenshtein Distanz:** Minimale Kosten, um *S1* in *S2* mit den 3 Operationen ***Löschen, Einfügen*** und ***Substitution*** zu verwandeln.

$$\left.\begin{matrix}(A,B,B,A,B)\\(A\ ,A,B)\end{matrix}\right\} \left.\begin{matrix}(A,B,B,A,B)\\(A\ ,-\ ,-\ ,A,B)\end{matrix}\right\} \quad Sim_{Lev}(S1,S2)=3$$

Berechnung der Levenshtein Distanz

Gegeben: Zwei Sequenzen S1, S2 über dem Alphabet A mit |S1| = n und |S2| = m.

Gesucht: $Dist_{Lev}(S1,S2)$

Berechnung der Levenshtein Distanz mit dynamischer Programmierung:

Sei D eine $n \times m$-Matrix über IN *mit:*

$$D_{0,0} = 0$$

$$D_{0,i} = i, \quad 0 \le i \le n$$

$$D_{j,0} = j, \quad 0 \le j \le m$$

$$D_{i,j} = \min \begin{cases} D_{i-1,j-1} + 0, \; falls \quad s_{1i} = s_{2,j} \\ D_{i-1,j-1} + 1, (Ersetzung) \\ D_{i,j-1} + 1, (Einfügung) \\ D_{i-1,j} + 1, (Löschung) \end{cases} \quad für \; 1 \le i \le n, \quad 1 \le j \le m$$

Nach Aufbau der Matrix enthält der Eintrag $D_{n,m}$ die Levenshtein-Distanz.

Beispiel Levenshtein Distanz

S1 = auto, S2 = ute

	-	a	u	t	o
-	0	1	2	3	4
u	1				
t	2				
e	3				

	-	a	u	t	o
-	0	1	2	3	4
u	1	1→	1		
t					
e					

	-	a	u	t	o
-	0	1	2	3	4
u	1	1	1	2	3
t	2	2	2	1	2
e	3	3	3	2	2

	-	a	u	t	o
-	0	1	2	3	4
u	1	1	1	2	3
t	2	2	2	1	2
e	3	3	3	2	2

(a,u,t,o)
(-, u,t,e)

$Dist_{Lev}(S1,S2)=2$

Edit Distanzen

- Verallgemeinerung der Levenshtein-Distanzen:
  - andere Kosten Matrix: Substitution kostet 4, Löschen 1, Einfügen 2..
  - weitere Operationen:
    - Reihenfolge vertauschen

(A,B,B,A,B)
(A,B,A,B,B)
} *1 Vertauschung*

    - vervielfältigen, …

(A,B,B,B,B)
(A,B,)
} *3 Vervielfältigungen von B*

- Kosten können für Werte unterschiedlich sein:

  Subst.(A,B) ≠ Subst.(A,Z)

- Funktioniert auch bei Sequenzen über reellwertige Alphabeten, zum Beispiel: für *A = IR: Subst(5,1) = |5-1|*

Markow-Ketten und Sequenzen

- Sequenzen aus Aktionen unterliegen bestimmten Regeln
- Modellierung über endliche Automaten
  (Test auf Gültigkeit der Sequenz)
- Markow-Ketten sind probabilistische Automaten:

  Modellieren nicht nur erlaubte Zustandsübergänge, sondern auch Wahrscheinlichkeitsverteilungen über Zustandsübergänge.
- **Markow-Annahme 1. Ordnung**: Der Zustand zum Zeitpunkt $t+1$ ist nur vom Zustand zum Zeitpunkt t abhängig.
- Ordnung einer Markow-Kette ist die Anzahl der Vorgängerzustände, von denen die Auswahl des nächsten Zustands abhängig ist.

Markow-Ketten 1. Ordnung

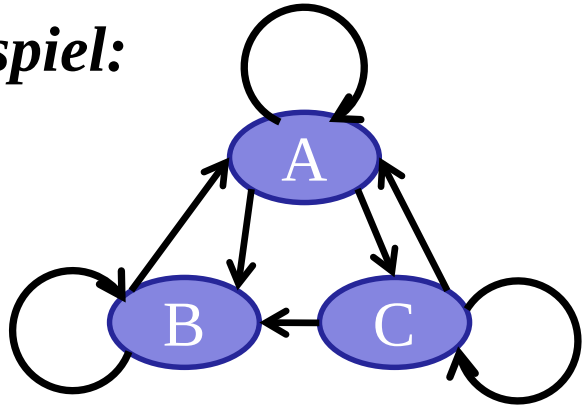
Definition: Eine Markow-Kette ***M*** ist definiert über eine Zustandsmenge ***A*** und eine stochastische $|A| \times |A|$ Übergangsmatrix ***D***.

Erläuterungen:

- *A* kann einen Start- und einen Absorbtionszustand enthalten (Modellierung von Start und Ende)
- Stochastische Matrix: Zeilen addieren sich auf 1 auf.

  (Zeile i enthält die Verteilung der Nachfolger vom Zustand i)

Beispiel:



	-	A	B	C
-	0.0	0.3	0.3	0.4
A	0.1	0.25	0.5	0.15
B	0.1	0.5	0.4	0.0
C	0.1	0.1	0.7	0.1

$$p(ACBB) = P(A|-) \cdot P(C|A) \cdot P(B|C) \cdot P(B|B) \cdot P(-|B)$$

$$= 0.3 \cdot 0.15 \cdot 0.4 \cdot 0.7 \cdot 0.4 \cdot 0.1$$

Hidden Markow Modelle

Training einer Markow Kette:

- Zerlegen der Trainingssequenzen in 2-Gramme und bestimme die relative Häufigkeit.
  (Wie häufig folgt auf A ein B?) $P(B \mid A) = \frac{fr(AB)}{fr(A)}$

Problem:

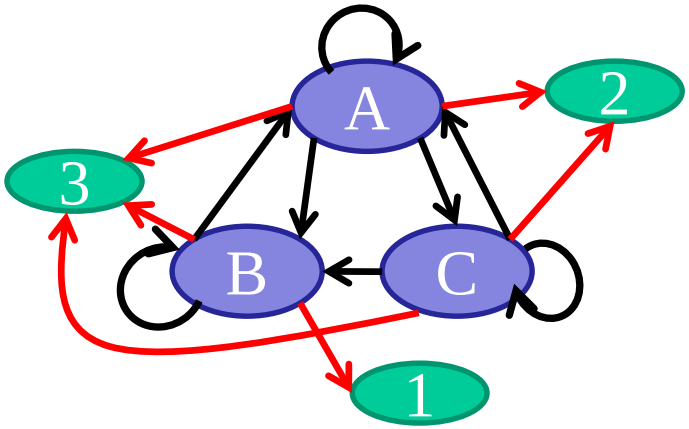
- Beobachtungen entsprechen häufig nicht direkt dem beobachteten Verhalten:
  - Action-Log ist verfügbar, aber Spielgeschehen soll analysiert werden
  - Fehlerhafte Ausführung verschleiert die tatsächliche Intention
  - Analyse der Zustandswechsel einer KI

    (beobachtete Aktionen können in verschiedenen Zuständen angewendet werden)

Hidden Markow Modelle

Definition: Ein Hidden Markow Model ***M*** ist definiert über eine Zustandsmenge ***A,*** eine stochastische $|A|\times|A|$ Übergangsmatrix *D*, eine Beobachtungsmenge B und eine stochastische $|A|\times|B|$ Output-Matrix F.

Beispiel: A={A,B,C}, B={1,2,3}



D	**-**	**A**	**B**	**C**
-	0.0	0.3	0.3	0.4
A	0.1	0.25	0.5	0.15
B	0.1	0.5	0.4	0.0
C	0.1	0.1	0.7	0.1

F	**1**	**2**	**3**
A	0.0	0.2	0.8
B	0.5	0.0	0.5
C	0.0	0.5	0.5

P(122): Bestimme alle Zustandstripel, die 122 erzeugen können:

BAA, BAC

$$P(122) = \quad P(BAA)\cdot P(122\,|\,BAA) + P(BAC)\cdot P(122\,|\,BAC)$$

Verwendung von HMMs

- **Evaluierung**: Wie wahrscheinlich ist eine Beobachtung $O=(o_1,..,o_k)$ mit $o_i \in B$ für das HMM (A,B,D,F)?

  (Forward Estimation)

- **Erkennung**: Gegeben ist die Beobachtung $O=(o_1, \ldots, o_k)$ und das HMM (A,B,D,F). Welche Sequenz $(s_1, \ldots, s_k)$ mit $s_i \in A$ erklärt O am besten? *(Viterbi-Algorithmus)*

- **Training**: Gegeben ist die Beobachtung $O=(o_1, \ldots, o_k)$, wie kann man D und F anpassen, um $P(O|(A,B,D,F))$ zu maximieren?

  (Baum-Welch Estimation)

Evaluierung: Forward Variablen

Gegeben: *O=(o_1, …, o_k) und (A,B,D,F)*

Gesucht: P(O|*(A,B,D,F)*)

Naive Lösung: Berechne P(O|S) für alle k-elementigen Sequenzen S über A.
(Anzahl steigt exponentiell mit k)

Bessere Lösung: Ausnutzen der Markow-Annahme

Definiere Forward-Variable $\alpha_j(t)$ als

$$\alpha_j(t) = P(o_1, o_2, ..., o_t, s_t = a_j \mid (ABDF))$$

Berechnung durch Induktion:

$$\alpha_j(1) = d_{-,j} \cdot f_{j,o_1} \quad ,1 \le j \le |A|$$

$$\alpha_j(t+1) = \left(\sum_{i=1}^{|A|} \alpha_i(t) \cdot d_{i,j} \right) \cdot f_{j,ot+1} \quad ,1 \le t \le k-1$$

Berechnung mit $|A|^2 \cdot k$ Operationen:

$$P(O \mid (A,B,D,F)) = \sum_{i=1}^{|A|} P(O, s_t = a_i \mid (A,B,D,F)) = \sum_{i=1}^{|A|} \alpha_i(k)$$

Erkennung: Viterbi Algorithmus

Gegeben: $O=(o_1, \ldots, o_k)$, und Model (A, B, D, F).

Gesucht: $S=(s_1, \ldots, s_k)$, die $P(O|S,(A, B, D, F))$ maximiert.

- Definiere δ(t) als höchste Wahrscheinlichkeit einer einzelnen Sequenz über A der Länge t für die Beobachtung O.

$$\delta_j(t) = \max_{s_1,..,s_{t-1}} P(s_1, ..., s_{t-1}, O \,|\, (A, B, D, F))$$

- Berechnung durch Induktion

$$\delta_j(1) = d_{-,j} \cdot f_{j,o_1} \qquad ,1 \le j \le |A|$$

$$\delta_j(t+1) = \left(\max_{1 \le i \le |A|} \left(\delta_i(t) d_{i,j} \right) \right) \cdot f_{j,o_{t+1}} \qquad ,1 \le j \le k-1$$

$$\psi_j(1) = 0 \qquad ,1 \le j \le |A|$$

$$\psi_j(t+1) = \arg\max_{1 \le i \le |A|} \left(\delta_i(t) d_{i,j} \right) \qquad ,1 \le j \le k-1$$

- Ähnlich dem Forward Algorithmus, aber effizienter, da immer nur die beste Lösung verfolgt wird.

Backward Variablen

Analog zur Forward-Variable lässt sich auch eine Backward-Variable definieren, die für das Training des HMM verwendet wird.

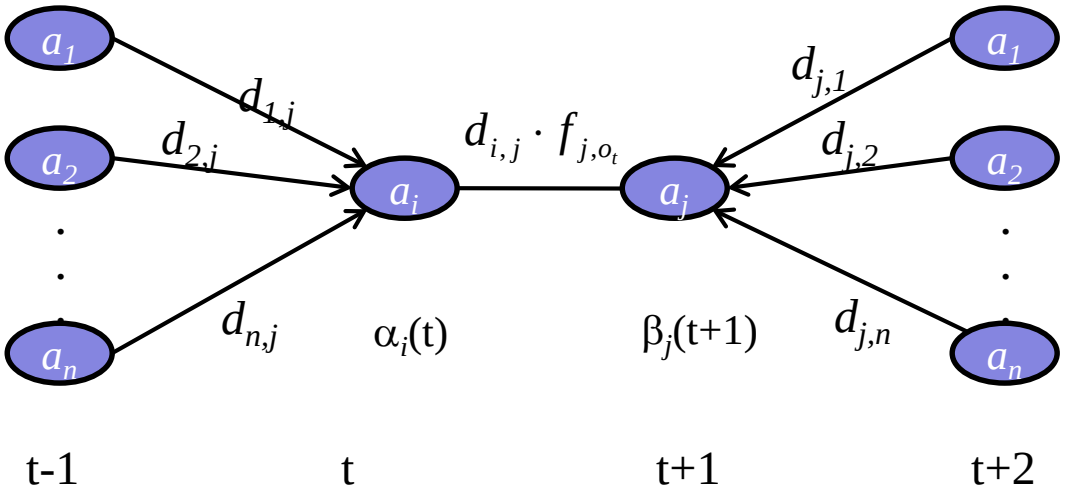
Definiere Backward-Variable $\beta_j(t)$ als

$$\beta_j(t) = P(o_{t+1}, ..., o_k \mid s_t = a_j, (ABDF))$$

Berechnung durch Induktion:

$$\beta_i(k) = 1 \quad ,1 \le i \le |A|$$

$$\beta_i(t-1) = \sum_{j=1}^{|A|} d_{i,j} \cdot f_{j,o_t} \cdot \beta_j(t) \quad ,2 \le t \le k$$



Training: Baum-Welch Estimation

Gegeben: $O=(o_1, \ldots, o_k)$, A und B.

Gesucht: D, F, die P(O|(A,B,D,F)) maximieren.

- Lokal optimierende Lösung durch *Expectation Maximization* **(EM)**
- Definiere $\xi_{i,j}(t)$ als Wahrscheinlichkeit, dass man zum Zeitpunkt t in Zustand a_i ist und zum Zeitpunkt $t+1$ in Zustand a_j:

$$\xi_{i,j}(t) = P(s_t = a_i, s_{t+1} = a_j \mid O,(A,B,D,F))$$

$$= \frac{\alpha_i(t) \cdot d_{i,j} \cdot f_{j,o_{t+1}} \beta_j(t+1)}{P(O \mid (A,B,D,F))}$$

$$= \frac{\alpha_i(t) \cdot d_{i,j} \cdot f_{j,o_{t+1}} \beta_j(t+1)}{\sum_{k=1}^{|A|} \sum_{l=1}^{|A|} \alpha_k(t) \cdot d_{k,l} \cdot f_{l,o_{t+1}} \beta_j(t+1)}$$

- Definiere γ_i (t) als die Wahrscheinlichkeit, dass man zum Zeitpunkt t in State a_i *ist:*

$$\gamma_i(t) = \sum_{j=0}^{|A|} \xi_{i,j}(t)$$

Training: Baum-Welch Estimation

- $\sum_{t=1}^{k-1} \xi_{i,j}(t)$ entspricht der erwarteten Anzahl von Zustandsübergängen von a_i auf a_j.
- $\sum_{t=1}^{k-1} \gamma_i(t)$ entspricht der erwarteten Anzahl an Zustandsübergängen von a_i auf andere Zustände.
- Parameter werden jetzt wie folgt neu abgeschätzt:

$$d_{-,a_i} = \gamma_i(1) \quad , d_{i,j} = \frac{\sum_{t=1}^{k-1} \xi_{i,j}(t)}{\sum_{t=1}^{k-1} \gamma_i(t)} \quad , f_{j,b_l} = \frac{\sum_{t \in \{t | o_t = b_l\}} \gamma_i(t)}{\sum_{t=1}^{k-1} \gamma_i(t)}$$

- Training erfolgt durch abwechselndes
  - Berechnen von $\gamma_i(t)$, $\xi_{i,,j}(t)$ und $P(O|(A,B,D,F))$
  - Updates von D und F *(Neuabschätzung wie oben)*
- Algorithmus terminiert, wenn $P(O|(A,B,D,F))$ weniger als ε wächst.

Reellwertige Sequenzen

- **Bisher**: Alphabet ist eine diskrete Domäne
- Sequenzen können aber auch über reellwertige Domänen, zum Beispiel IR^d, gebildet werden.
- Frequent Pattern Mining auf reellwertigen Domänen ist i.d.R. nicht möglich.
- Vergleich von 2 reellwertigen Sequenzen über der Domäne D mittels einer Distanzfunktion $dist: D \times D \rightarrow IR_0^+$.
  - Analog zur Hamming Distanz kann man die Summe der Distanzen an jeder Position der Sequenz bestimmen.

$$dist_{sequ}(S_1, S_2) = \sum_{i=1}^{|S_1|} dist(s_{1,i}, s_{2,i}) + (|S_2| - |S_1|) \cdot \varphi, \quad \text{für} \quad |S_2| \geq |S_1|, \varphi \in IR^+$$

  - Erweitern der Edit-Distanz ist ebenfalls möglich: Kosten für Substitution von v und u entsprechen dann $dist(v,u)$.

    (Genaueres kommt später beim Dynamic Time Warping)

Zeitreihen

- **Bisher**: Sequenzen modellieren nur die Reihenfolge, nicht aber die Zeitpunkte der Handlungen.

  Aber: In Echtzeitspielen ist Timing entscheidend.

  $\Rightarrow$ RTS Spiele: Build-Order sind nur dann effektiv, wenn sie in minimaler Zeit realisiert werden.

  $\Rightarrow$ In MMORPGs hängt der ausgeteilte Schaden von der Anzahl der Aktionen pro Zeiteinheit ab.

  $\Rightarrow$ Schach mit Schachuhr: Ein Zug wird auch daran gemessen, wie viel Zeit man zum Nachdenken gebraucht hat.

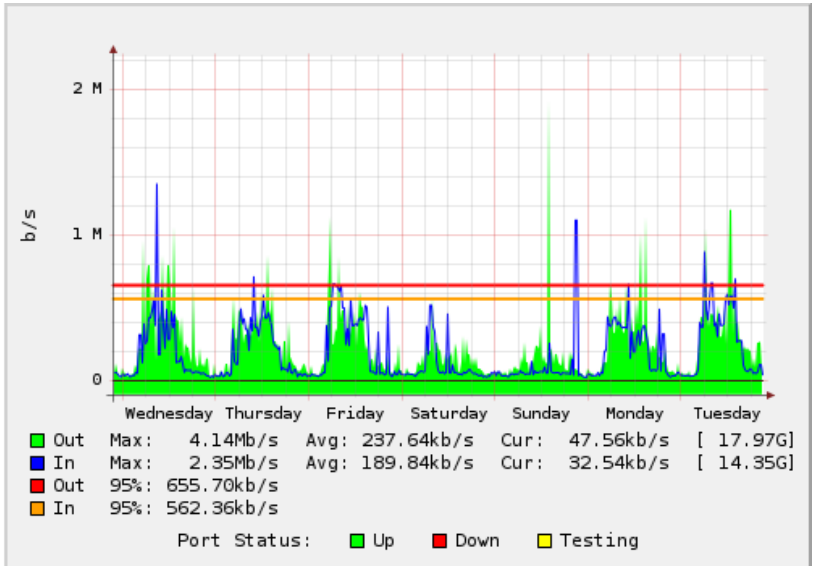
- **Zeitreihe**: Sei T eine Domäne zur Darstellung der Zeit und sei F eine Objektdarstellung, dann heißt:
  $Z=((x_1,t_1),..,(x_l,t_l)) \in (F \times T) \times .. \times (F \times T)$ Zeitreihe der Länge l über F.

Beispiele für Zeitreihen

- SC2-Logs: Zeitreihe über diskrete Handlungen

```
0:00  TSLHyuN     Select Hatchery (10230)
0:00  TSLHyuN     Select Larva x3 (1027c,10280,10284), Deselect all
0:00  TSLHyuN     Train Drone
0:01  TSLHyuN     Train Drone
0:01  TSLHyuN     Select Drone x6 (10234,10238,1023c,10240,10244,10248), Deselect all
0:01  TSLHyuN     Right click; target: Mineral Field (10114)
0:01  TSLHyuN     Deselect 6 units
0:02  TSLHyuN     Right click; target: Mineral Field (10170)
….
```

- Netzwerk-Traffic:
  - Verwendung in der Bot-Erkennung
  - Abschätzung der Spielintensität

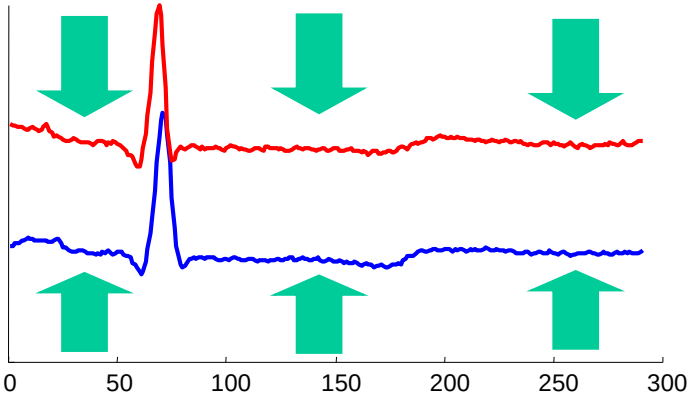


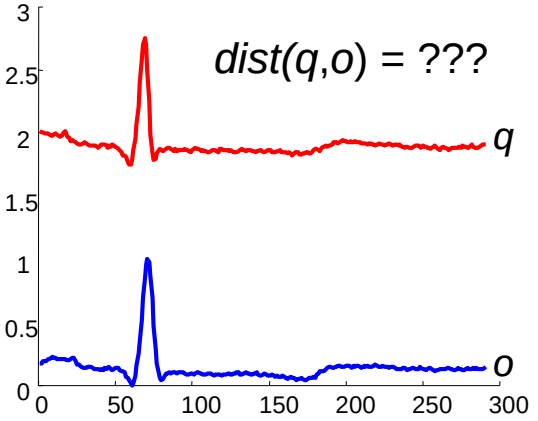
Vorverarbeitung von Zeitreihen (1)

Offset Translation

- Ähnliche Zeitreihen mit unterschiedlichen Offsets
- Verschiebung aller Zeitreihen um den Mittelwert *MW*:

$\forall 1 \leq i \leq |o|: o_i = o_i - \mathrm{MW}(o)$

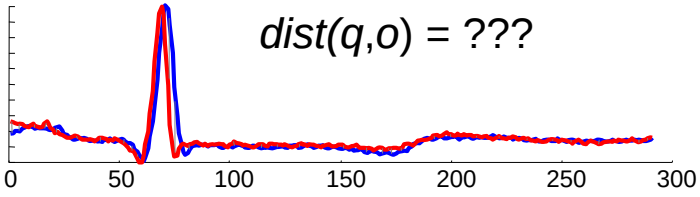




q = q - MW(q)

o = o - MW (o)

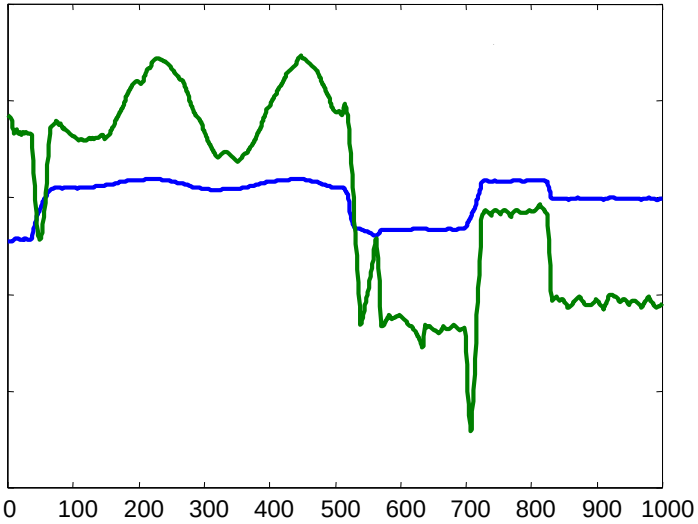
dist(q,o) = ???

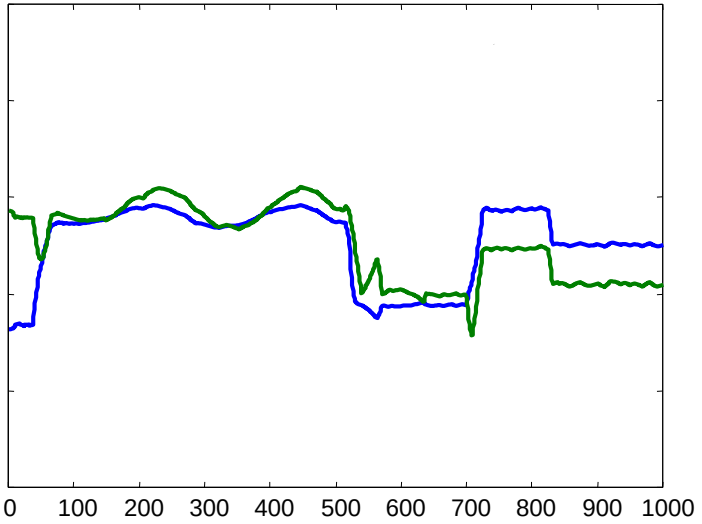


Vorverarbeitung von Zeitreihen(2)

Amplituden Skalierung

- Zeitreihen mit ähnlichem Verlauf, aber unterschiedlichen Amplituden
- Verschiebung der Zeitreihen um den Mittelwert (*MW*) und Normierung der Amplitude mittels der Standardabweichung (StD):
  $\forall 1 \leq i \leq |o|: o_i = (o_i - \mathrm{MW}(o)) / \mathrm{StD}(o)$





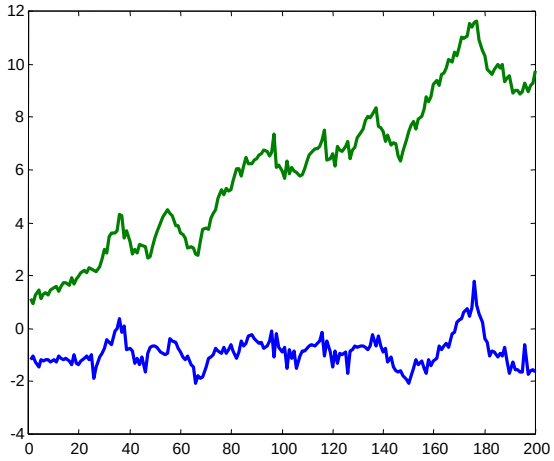
$q = (q - MW(q)) / \mathrm{StD}(q)$

$o = (o - MW(o)) / \mathrm{StD}(o)$

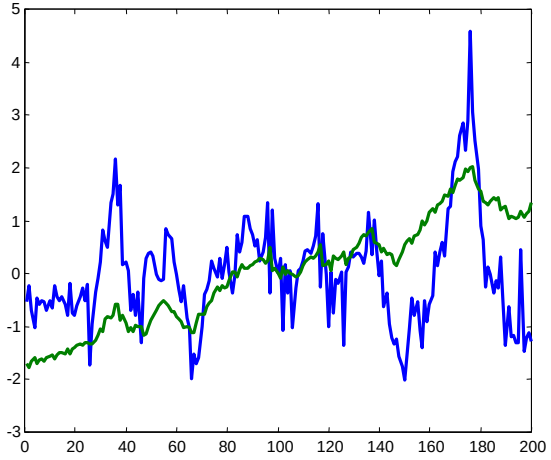
Vorverarbeitung von Zeitreihen (3)

Lineare Trends

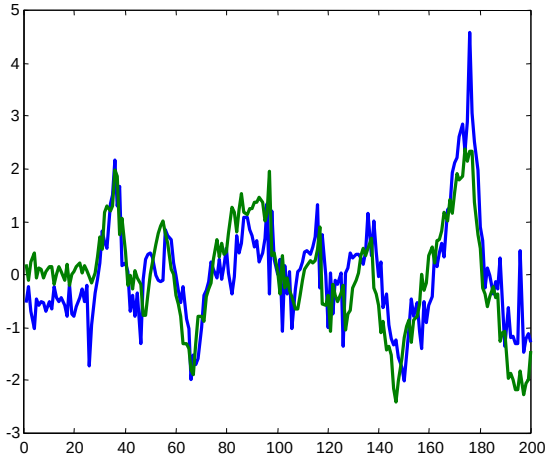
- Ähnliche Zeitreihen mit unterschiedlichen Trends
- Intuition:
  - Bestimme Regressionslinie
  - Verschiebe Zeitreihe anhand dieser Linie



Offset Translation + Amplituden Skalierung



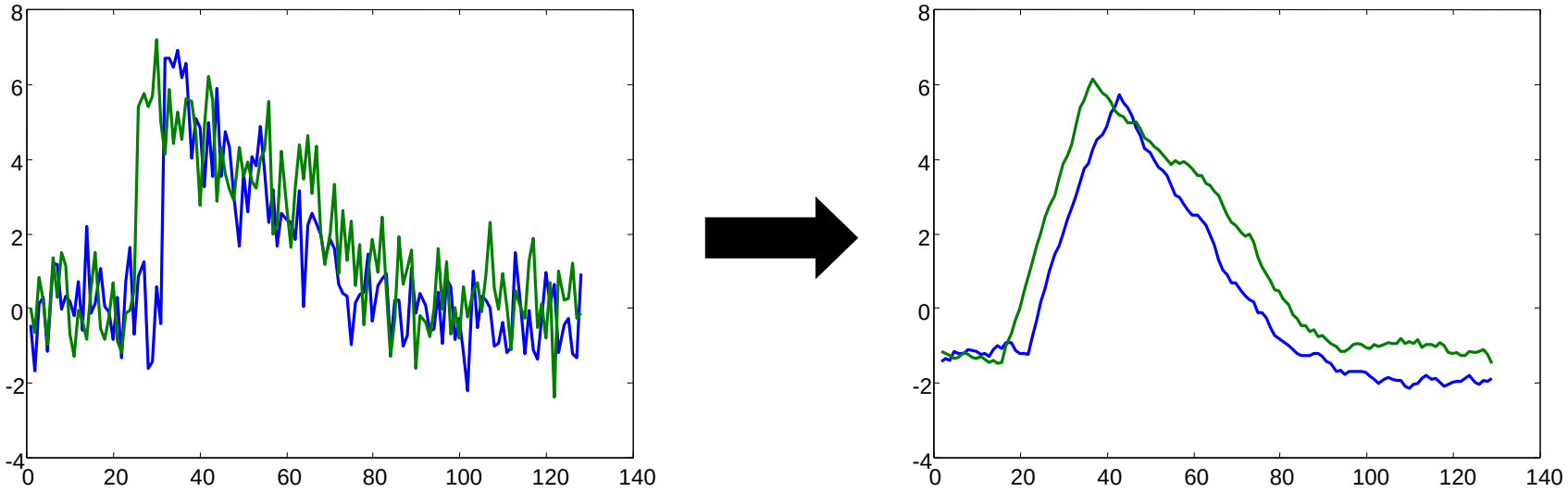
Offset Translation + Amplituden Skalierung
+ Lineare Trend-Beseitigung



Vorverarbeitung von Zeitreihen (4)

Bereinigung von Rauschen

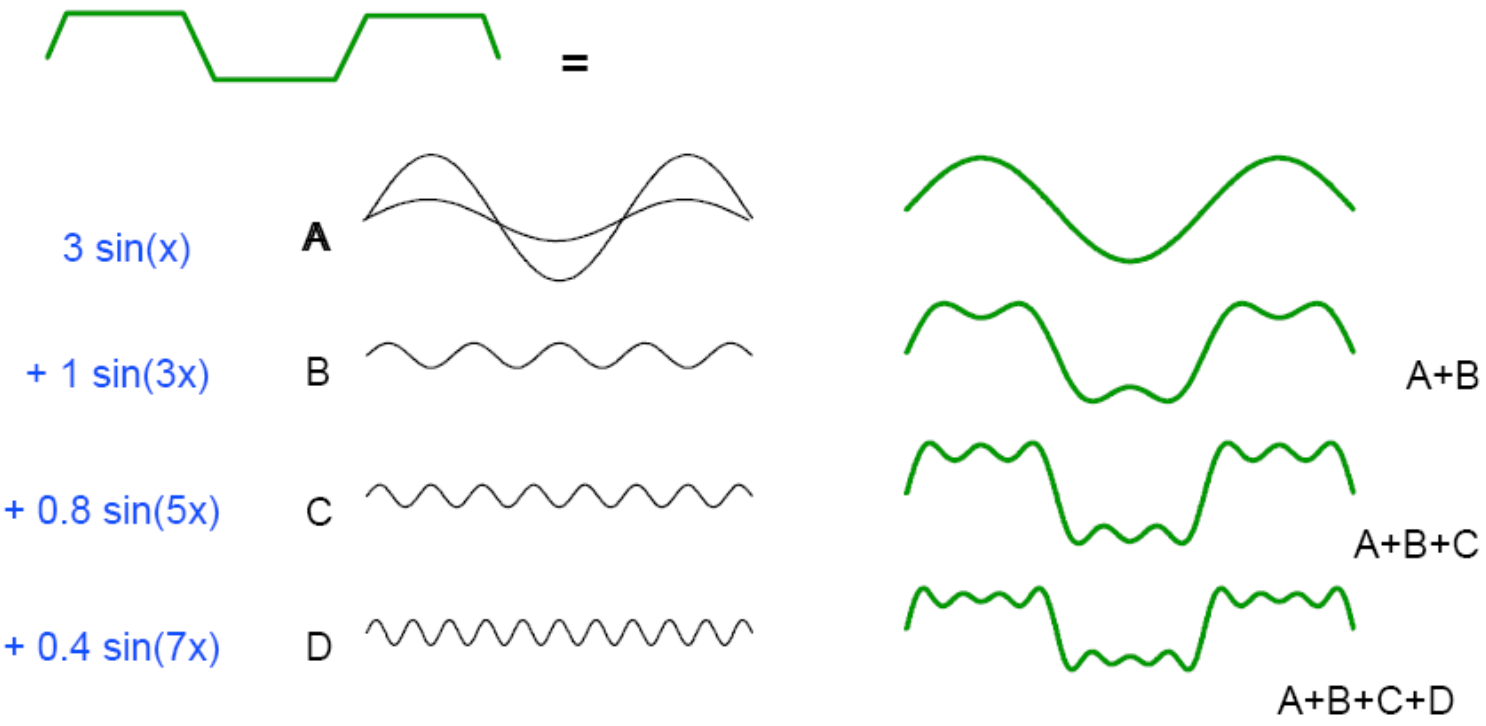
- Ähnliche Zeitreihen mit hohem Rauschanteil
- Glättung: Bilde für jeden Wert o_i den Mittelwert über alle Werte $[o_{i-k}, \ldots, o_i, \ldots, o_{i+k}]$ für ein gegebenes k.



Diskrete Fourier Transformation (DFT)

Idee:

- Beschreibe beliebige periodische Funktionen als gewichtete Summe periodischer Grundfunktionen (Basisfunktionen) mit unterschiedlicher Frequenz. Aus einer Zeitreihe wird ein Vektor mit fester Länge.
- Basisfunktionen: sin und cos

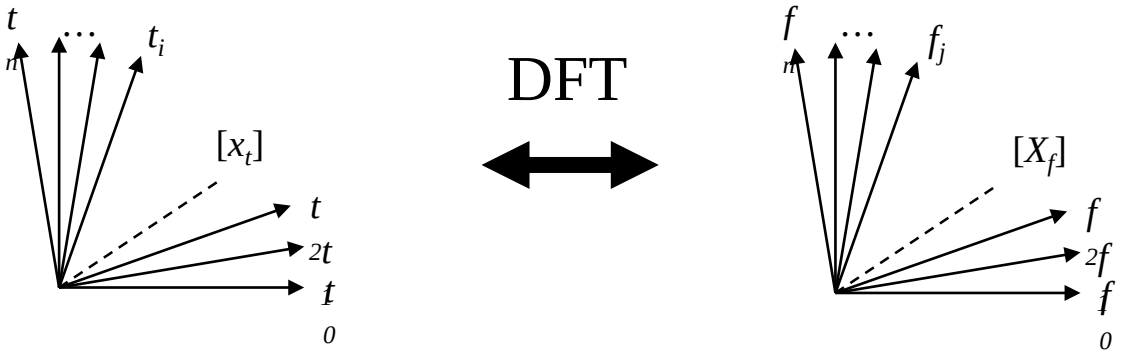


Diskrete Fourier Transformation (DFT)

Fouriers Theorem: Jede beliebige periodische Funktion lässt sich als Summe von Kosinus- und Sinus-Funktionen unterschiedlicher Frequenzen darstellen.

Eigenschaften:

- Transformation verändert eine Funktion nicht, sondern stellt sie nur anders dar
- Transformation ist umkehrbar => inverse DFT
- Analogie: Basiswechsel in der Vektorrechnung



Diskrete Fourier Transformation (DFT)

Formal:

- Gegeben sei eine Zeitreihe der Länge n: $x = [x_t]$, $t = 0, \ldots, n-1$
- Die DFT von x ist eine Sequenz $X = [X_f]$ von n komplexen Zahlen für die Frequenzen $f = 0, \ldots, n-1$ mit

$$X_f = \frac{1}{\sqrt{n}} \sum_{t=0}^{n-1} x_t \cdot e^{\frac{-i \cdot 2\pi \cdot f \cdot t}{n}} =$$

$$\underbrace{\frac{1}{\sqrt{n}} \sum_{t=0}^{n-1} x_t \cos\left(\frac{2 \cdot \pi \cdot f \cdot t}{n}\right)}_{\text{Realteil}} \underbrace{- i \cdot \frac{1}{\sqrt{n}} \sum_{t=0}^{n-1} x_t \sin\left(\frac{2 \cdot \pi \cdot f \cdot t}{n}\right)}_{\text{Imaginärteil}}$$

wobei i die imaginäre Einheit bezeichnet, d.h. $i^2 = -1$.

- Der Realteil gibt den Anteil der Kosinusfunktionen und der Imaginärteil den Anteil der Sinusfunktionen in der jeweiligen Frequenz f an.

Diskrete Fourier Transformation (DFT)

- Durch die inverse DFT wird das ursprüngliche Signal x wieder hergestellt:

$$x_t = \frac{1}{\sqrt{n}} \sum_{f=0}^{n-1} X_f \cdot e^{\frac{i \cdot 2 \cdot \pi \cdot f \cdot t}{n}}$$

$t = 0, \ldots, n-1$ (t: Zeitpunkte)

$[x_t] \leftrightarrow [X_f]$ bezeichnet ein **Fourier-Paar**,
d.h. $\text{DFT}([x_t]) = [X_f]$ und $\text{DFT}^{-1}([X_f]) = [x_t]$.

- Die DFT ist eine **lineare Abbildung**, d.h. mit $[x_t] \leftrightarrow [X_f]$ und $[y_t] \leftrightarrow [Y_f]$ gilt auch:
  - $[x_t + y_t] \leftrightarrow [X_f + Y_f]$ und
  - $[ax_t] \leftrightarrow [aX_f]$ für ein Skalar $a \in IR$

- **Energie einer Sequenz**
  - Die Energie $E(c)$ von c ist das Quadrat der Amplitude: $E(c) = |c|^2$.
  - Die Energie $E(x)$ einer Sequenz x ist die Summe aller Energien über die Sequenz: $E(x) = \|x\|^2 = \sum_{t=0}^{n-1} |x_t|^2$

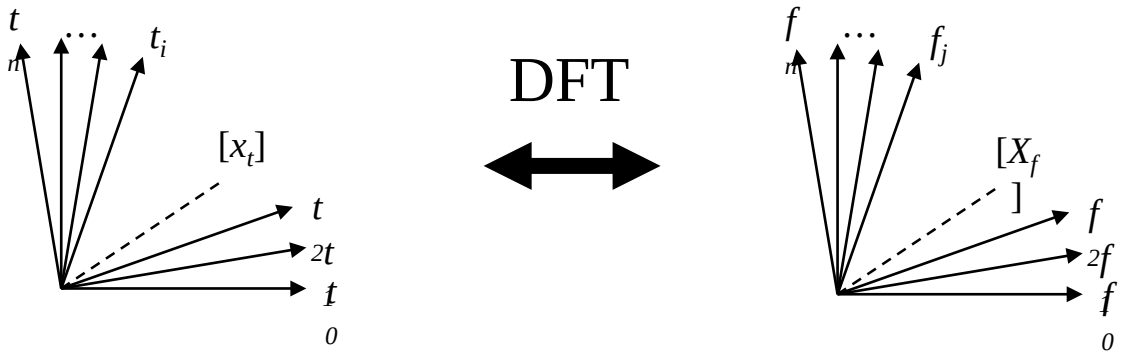
Diskrete Fourier Transformation (DFT)

Satz von Parseval: Die Energie eines Signals im Zeitbereich ist gleich der Energie im Frequenzbereich.

Formal: Sei X die DFT von x, dann gilt:

$$\sum_{t=0}^{n-1} \left| x_t \right|^2 = \sum_{t=0}^{n-1} \left| X_f \right|^2$$

- Folge aus Parsevals Satz und der Linearität der DFT: Die Euklidische Distanz zweier Signale x und y stimmt im Zeit- und im Frequenzbereich überein: $\| x - y \|^2 = \| X - Y \|^2$



„Zeitbereich (-raum)“ „Frequenzbereich (-raum)“

Diskrete Fourier Transformation (DFT)

Grundidee der Anfragebearbeitung:

- Als Ähnlichkeitsfunktion für Sequenzen wird die Euklidische Distanz verwendet:
  $dist(x, y) = \|x - y\| = \sqrt{\sum_{t=0}^{n-1} |x_t - y_t|^2}$
- Der Satz von Parseval ermöglicht nun, die Distanzen im Frequenz- statt im Zeitbereich zu berechnen: $dist(x,y) = dist(X,Y)$
- In der Praxis haben die tiefsten Frequenzen die größte Bedeutung.
- Die ersten Frequenz-Koeffizienten enthalten also die wichtigste Information.
- Für den Aufbau eines Index werden die transformierten Sequenzen gekürzt, d.h. von $[X_f]$, $f = 0, 1, \ldots, n-1$ werden nur die ersten c Koeffizienten $[X_f < c]$, $c < n$, indexiert.

$$dist_c(x, y) = \sqrt{\sum_{f=0}^{c-1} |x_f - y_f|^2} \leq \sqrt{\sum_{f=0}^{n-1} |x_f - y_f|^2} = dist(x, y)$$

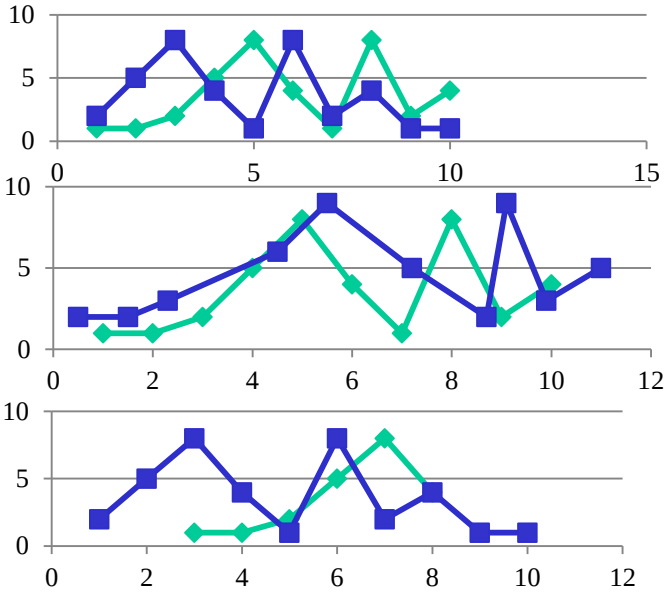
- Im Index kann dann eine untere Schranke der echten Distanz berechnet werden:

  Filter-Refinement:
  - Filterschritt auf gekürzten Zeitreihen (mit Indexunterstützung),
  - Refinement auf kompletten Zeitreihen

Distanzen auf Zeitreihen

Probleme: Welche Zeitpunkte sollen verglichen werden?

- Offset am Anfang: S2 ist zu S1 zeitlich verschoben.
- Taktung der Messwerte: Zeitpunkte der Messungen sind unterschiedlich.
- Länge der Zeitreihe: Dauer der Messung ist unterschiedlich.



- Bei gleicher Taktung und Länge können Zeitreihen als Vektoren verglichen werden. (Dimension = Zeitpunkt)

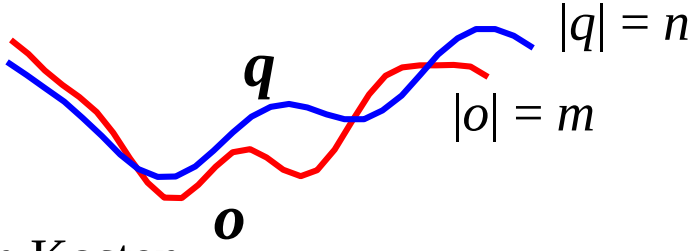
$$Dist_{timeseries}(S1, S2) = \sum_{t=1}^{T} dist_{obj}(s_{1t}, s_{2t})$$

- Bei Variabler Länge, Taktung und bei Offsets, Adaption der Edit-Distanz für Sequenzen => ***Dynamic Time Warping***

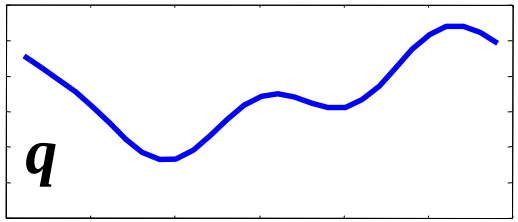
Dynamic Time Warping Distanz

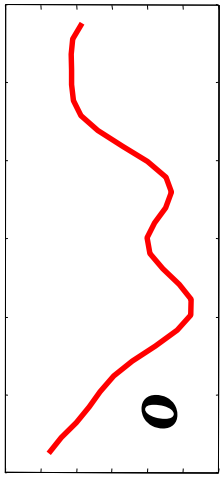
Berechnung:

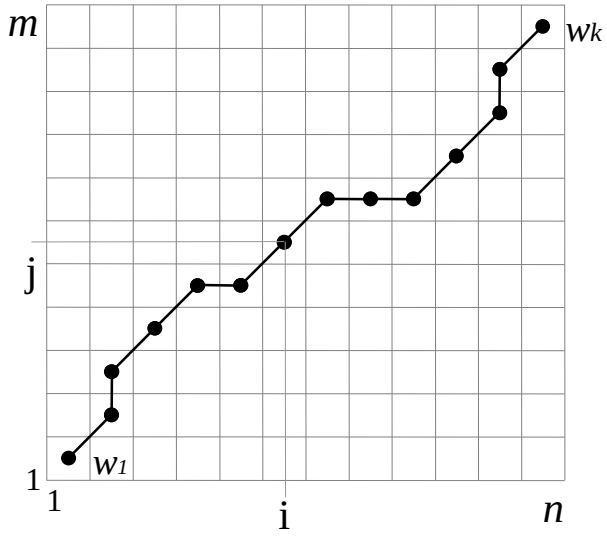
- Gegeben: Zeitreihen q und o unterschiedlicher Länge
- Finde Mapping von allen q_i auf o mit minimalen Kosten

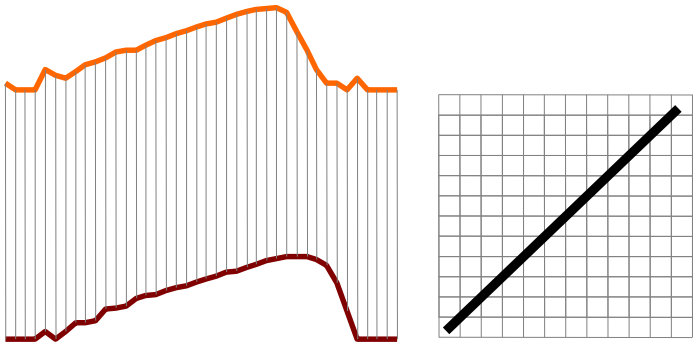


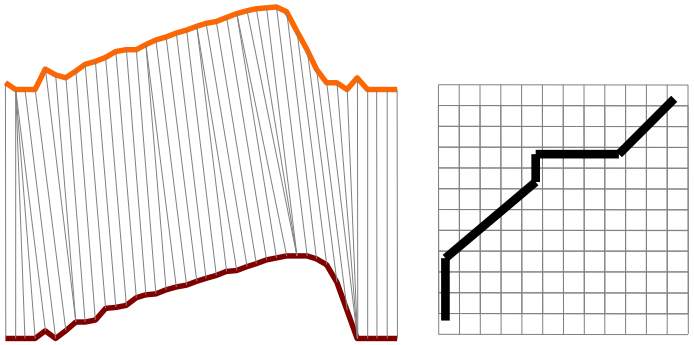
Suchmatrix











Dynamic Time Warping Distanz

Suchmatrix

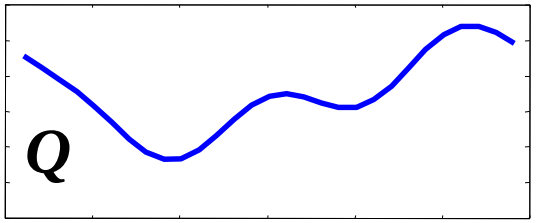
- Alle möglichen Mappings von q auf o können als „warping“ Pfad in der Suchmatrix aufgefasst werden
- Von all diesen Mappings suchen wir den Pfad mit den niedrigsten Kosten

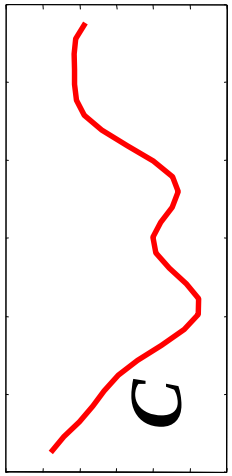
$$DTW(q,o) = \min\left\{ \sqrt{\sum_{k=1}^{K} w_k} \Big/ K \right\}$$

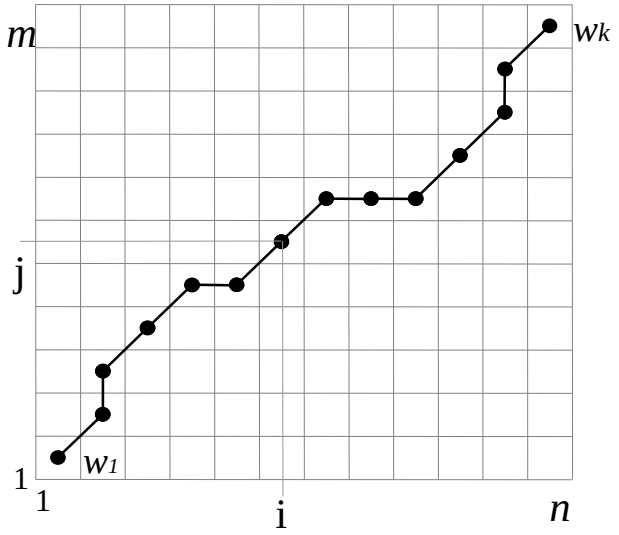
- Dynamisches Programmieren

  => Laufzeit $(n \cdot m)$

  (vgl. Edit Distanz)



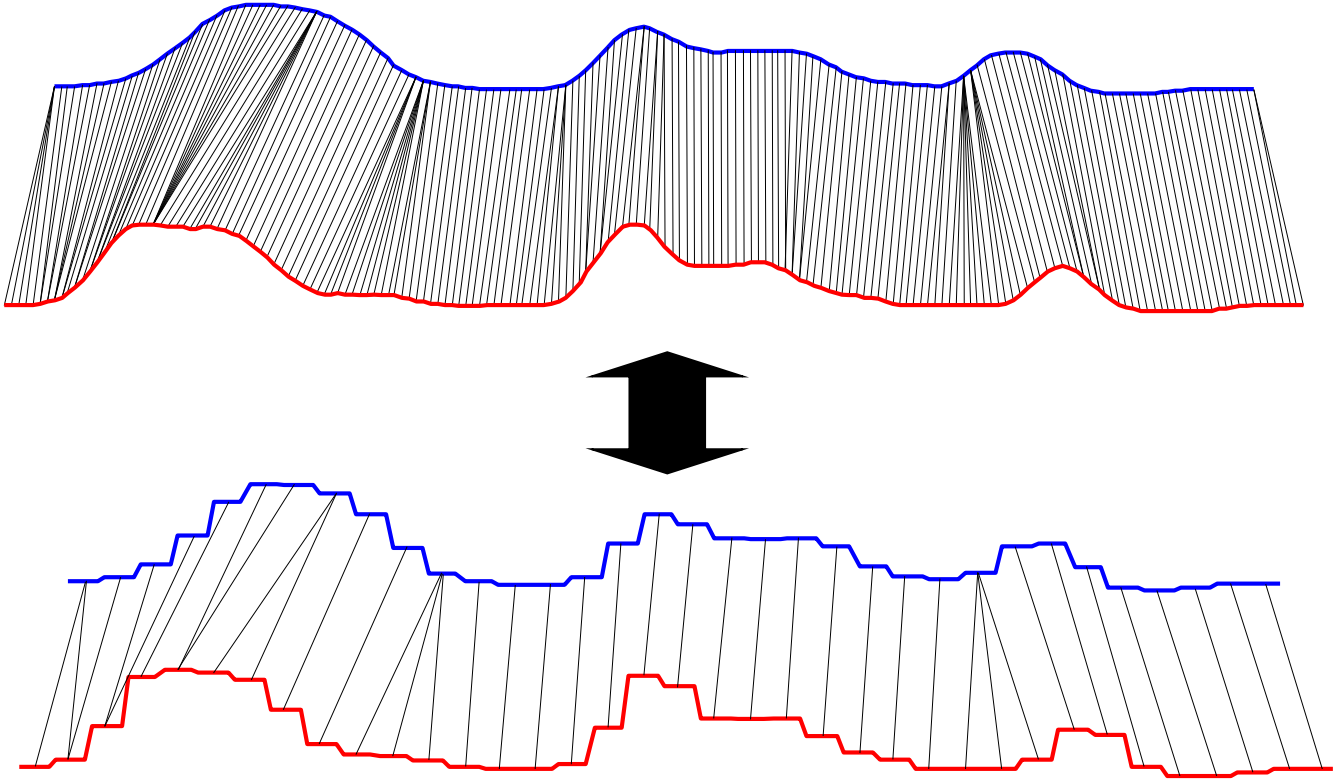




Approximative Dynamic Time Warping Distanz

Idee:

- Approximiere die Zeitreihen
  (komprimierte Repräsentation, Sampling, …)
- Berechne DTW auf den Approximationen



Statitische Modelle für die Zeit

Problem:

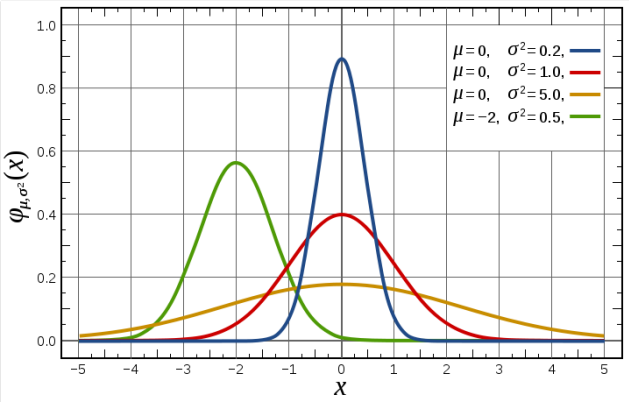
Wie kann man den Zeitpunkt der nächsten Aktion modellieren?

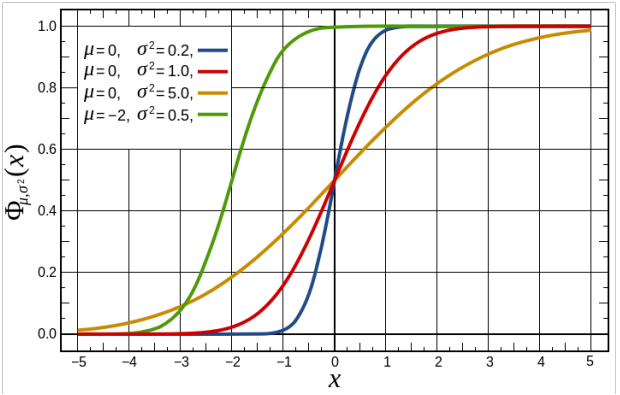
⇒ Statistische Modelle für die Zeit zwischen 2 Ereignissen werden benötigt.

⇒ Zeit ist eine kontinuierliche Variable => Wahrscheinlichkeitsdichtefunktion

⇒ Gesucht: Wahrscheinlichkeit, dass das nächste Ereignis *e* im Zeitraum $t+\Delta t$ stattfindet.

⇒ Beschreibung über kumulierte Wahrscheinlichkeitsdichtefunktion

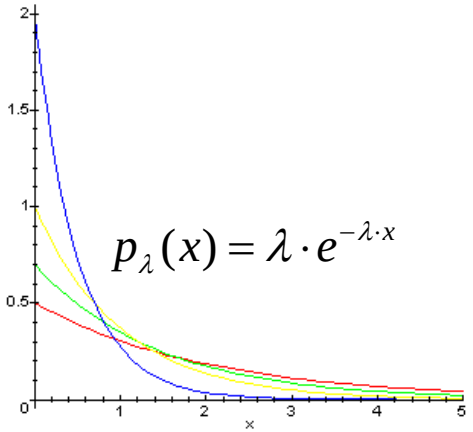




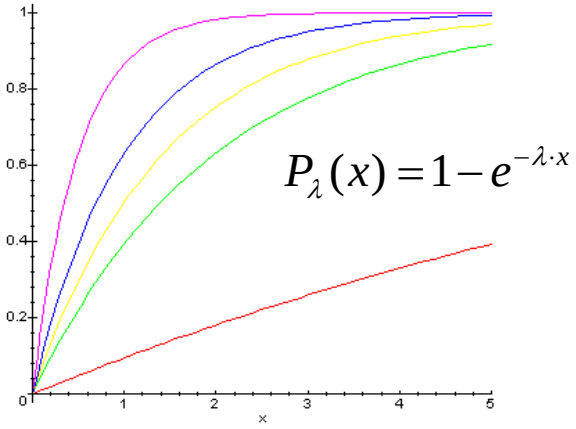
Homogene Poisson Prozesse

- Einfachster Prozess zur Modellierung von Zeit.
- Zeitpunkte zwischen 2 Ereignissen sind exponentiell verteilt. Wahrscheinlichkeitsdichte der Exponentialverteilung: $p_\lambda(x) = \lambda \cdot e^{-\lambda \cdot x}$
- Durch Integration erhält man die kumulierte Dichtefunktion, die die Wahrscheinlichkeit beschreibt, dass die nächste Aktion im Zeitintervall zwischen 0 … x stattfinden wird.

$$P_\lambda(x) = \int_0^x p_\lambda(t)dt = 1 - e^{-\lambda \cdot x}$$



Dichtefunktion der Exponential-Verteilung



kumulierte Dichtefunktion der Exponential-Verteilung

Parameterschätzung

Gegeben: Eine Trainingsmenge aus Zeitpunkten $X=\{x_1, \ldots, x_n\}$, die exponentialverteilt sind.

Gesucht: Der wahrscheinlichste Wert für den Intensitätsparameter λ.

Abschätzung über Maximum Likelihood

=> Suche den Wert für λ, bei dem die Wahrscheinlichkeit der Generierung von X am höchsten ist. Likelihood-Funktion L für Sample X:

$$L_X(\lambda) = \prod_{i=1}^{n} \lambda \cdot e^{-\lambda \cdot x_i} = \lambda^n \cdot e^{-\lambda \cdot \sum_{i=1}^{n} x_i} = \lambda^n \cdot e^{-\lambda \cdot n \cdot E(X)} \qquad mit \quad E(X) = \frac{\sum_{i=1}^{n} x_i}{n}$$

Ableiten der logarithmischen Likelihood nach λ und bestimmen der Nullstellen:

$$\frac{d}{d\lambda} \ln L(\lambda) = \frac{d}{d\lambda}(n \cdot \ln(\lambda) - \lambda \cdot n \cdot E(X)) = \frac{n}{\lambda} - n \cdot E(X)$$

$$\Rightarrow \lambda^* = \frac{1}{E(X)}$$

Lernziele

- Sequenzen und Zeitreihen
- Frequent Subsequenz Mining mit Suffix-Bäumen
- Distanzmaße auf Sequenzen
  - Hamming Distanz
  - Levenshtein Distanz
- Markow-Ketten
- Hidden Markow Ketten
- Zeitreihen und Vorverarbeitungsschritte
- Dynamic Time Warping
- Poisson Prozesse

Literatur

- Kyong Jin Shim, Jaideep Srivastava: **Sequence Alignment Based Analysis of Player Behavior in Massively Multiplayer Online Role-Playing Games (MMORPGs),** in Proceedings of the 2010 IEEE International Conference on Data Mining Workshops, 2010.

- Ben G. Weber, Michael Mateas: **A data mining approach to strategy prediction, i**n Proceedings of the 5th International Conference on Computational Intelligence and Games, 2009.

- K.T. Chen, J.W. Jiang, P. Huang, H.H. Chu, C.L. Lei, W.C. Chen: **Identifying MMORPG bots: A traffic analysis approach,** In Proceedings of the 2006 ACM SIGCHI International Conference on Advances in Computer Entertainment Tsechnology, 2006.