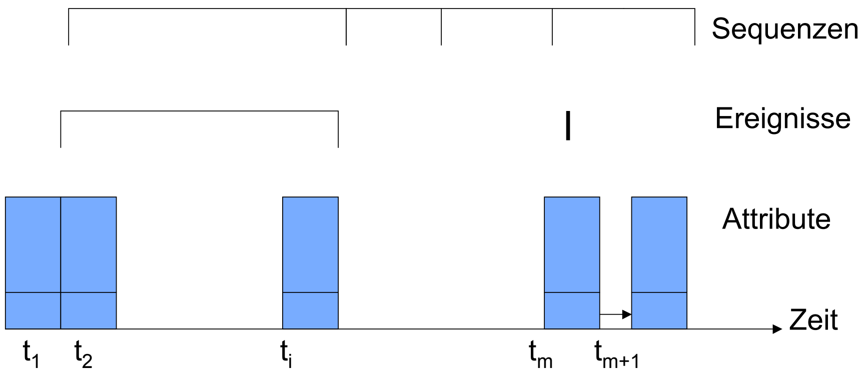
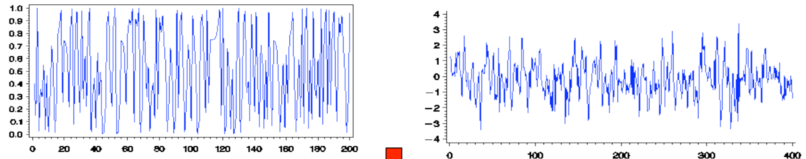
Zeitphänomene



Beispiele für Zeitreihen

- Messwerte von einem Prozess
  - Intensivmedizin
  - Aktienkurse
  - Wetterdaten
  - Roboter



Kontinuierliche Messung in z.B. Tagen, Stunden, Minuten, Sekunden

Beispiele für Ereignisse

- Datenbankrelationen
  - Vertragsdaten, Verkaufsdaten, Benutzerdaten
  - Lebenssituation (Einkommen, Alter)

Verkäufe	Monat	Anzahl	Verkäufer	...
	Juni	256	Meier	...
	...	...	...	...

Ereignisse mit Zeitangaben in Jahren, Monaten, Tagen

Granularität

- Eine Granularität ist eine Abbildung von natürlichen Zahlen (oder Zeichenketten) auf Teilmengen der Zeitwerte, so dass gilt:
  1. Wenn i < j und G(i), G(j) nicht leer, dann ist jedes Element von G(i) kleiner als alle Elemente von G(j).
  2. Wenn i < k < j und G(i) und G(j) nicht leer, dann ist G(k) auch nicht leer.

1. Der Index i,k,j bezeichnet eine Kodierung der Zeiteinheiten. Die Zeiteinheiten überlappen sich nicht.
2. Die Teilmengen von Indizes folgen aufeinander. Tage, Arbeitstage, Wochen, Semester, Kalenderjahre sind Zeiteinheiten.

- Beispiel: Jahre seit 2000 sei definiert als G mit G(i)={} für i<1, G(1)=alle Zeit im Jahre 2000, G(i+1)= alle Zeit in 2001,...

Temporale Module

- Temporales Modulschema (R,G), wobei R ein Relationenschema ist und G eine Zeitgranularität.
- Temporales Modul (R,G,p), wobei p die Zeitfensterabbildung von natürlichen Zahlen auf Tupel in der Zeiteinheit ist.
- Zu einer Zeiteinheit G(i) liefert p alle Tupel, die in der entsprechenden Zeit gelten.
- Beispiel: Sei in R das Jahresgehalt für Mitarbeiter und sei G Jahre seit 2000, dann liefert p für i=1 alle Gehälter im Jahre 2000.

Temporale Datenbank

- Das Schema einer temporalen Datenbank ist eine Menge von temporalen Modulschemata.
- Eine Menge von temporalen Modulen bildet eine temporale Datenbank.

Claudio Bettini, Sushil Jajodia, Sean X. Wang (1998)

„Time Granularities in Databases, Data Mining, and Temporal Reasoning“
Springer

Beispiel

t1(kurs)=CS50=t2(kurs)
p(1993-5-26)=[CS50, 3, Woo, 2 000, 50]
p(1993-5-30)=[CS50, 3, Woo, 2 000, 45] ...
G sei Tag als Einheit, G(1)= 1993-5-26, G(2)= 1993-5-30
H sei Kalenderwoche als Einheit H(22)={1993-5-26, 1993-5-27, 1993-5-28, 1993-5-29, 1993-5-30, 1993-5-31, 1993-6-1}

kurs	credits	wimi	gehalt	#studis	tag
CS50	3	Woo	2 000	50	1993-5-26
CS50	3	Woo	2 000	45	1993-5-30
CS50	3	Woo	2 500	48	1993-6-2
CS50	3	Lee	2 000	46	1993-6-13
CS50	3	Lee	2 000	44	1993-6-16
CS50	3	Lee	2 000	43	1993-6-20

Partielle Ordnungen der Granularität

- Wann ist eine Granularität feiner als eine andere?
  Z.B. Tag, Woche
- Wann ist eine Granularität eine Untergranularität einer anderen?
  Wenn es für jedes G(i) einen Index j gibt, so dass G(i)=H(j), dann ist G Untergranularität von H.
- Wann deckt eine Granularität eine andere ab?
  Wenn der Bildbereich von G im Bildbereich von H enthalten ist, dann wird G von H abgedeckt.

Schemaentwurf

1. Die Attribute tag und kurs sollen Schlüssel sein.
2. Die funktionale Abhängigkeit kurs → credits soll gegeben sein.
3. Das Gehalt eines Mitarbeiters ändert sich nicht innerhalb eines Monats.
4. Mitarbeiter wechseln sich nicht innerhalb derselben Woche ab.

kurs	credits	wimi	gehalt	#studis	tag
CS50	3	Woo	2 000	50	1993-3-3
CS50	3	Woo	2 000	45	1993-3-8
CS50	3	Woo	2 500	48	1993-4-5
CS50	3	Lee	2 000	46	1993-4-10
CS50	3	Lee	2 000	44	1993-5-7
CS50	3	Lee	2 000	43	1993-5-12

Oh!

Anomalien

- Redundanz: credits, gehalt
- Einfügeanomalie: Woos Gehalt in einem Tupel ändern und in den anderen desselben Monats lassen...
- Löschanomalie: Wenn der Kurs gelöscht wird, verlieren wir die Mitarbeiternamen...

kurs	credits	wimi	gehalt	#studis	tag
CS50	3	Woo	2 000	50	1993-5-26
CS50	3	Woo	2 000	45	1993-5-30
CS50	3	Woo	2 500	48	1993-6-2
CS50	3	Lee	2 000	46	1993-6-13
CS50	3	Lee	2 000	44	1993-6-16
CS50	3	Lee	2 000	43	1993-6-20

Dekomposition

wimi	gehalt	monat
Woo	2 000	1993-5
Woo	2 500	1993-6
Lee	2000	1993-6

kurs	credits
CS50	3

kurs	wimi	kalender woche
CS50	Woo	22
CS50	Woo	23
CS50	Lee	24
CS50	Lee	25

kurs	#studis	tag
CS50	50	1993-5-26
CS50	45	1993-5-30
CS50	48	1993-6-2
CS50	46	1993-6-13
CS50	44	1993-6-16
CS50	43	1993-6-20

Temporale funktionale Abhängigkeiten

- Sei (R, G, p) ein temporales Modul, dann gilt $X \rightarrow_H Y$ gdw. wenn gilt
  1. $t1(X) = t2(X)$
  2. t1 in p(i) und t2 in p(j)
  3. Es gibt ein z mit $G(i) \cup G(j) = G(i,j)$ und $G(i,j) \subseteq H(z)$

  dann $t1(Y)=t2(Y)$.

kurs $\rightarrow_{\text{kalenderwoche}}$ wimi
kurs $\rightarrow_{\text{tag}}$ #studis
wimi $\rightarrow_{\text{monat}}$ gehalt

Beispiel1

kurs $\rightarrow_{kalenderwoche}$ wimi gilt, denn wenn
1. t1(kurs)=CS50=t2(kurs),
2. t1 in p(datum1) und t2 in p(datum5)
3. G(i)=datum1, G(j)=datum5, {datum1, datum5} in H(z)={datum1, datum2, datum3, datum4,datum5, datum6, datum7}

Dann t1(wimi)=t2(wimi)

kurs	credits	wimi	gehalt	#studis	tag
CS50	3	Woo	2 000	50	1993-5-26
CS50	3	Woo	2 000	45	1993-5-30
CS50	3	Woo	2 500	48	1993-6-2
CS50	3	Lee	2 000	46	1993-6-13
CS50	3	Lee	2 000	44	1993-6-16
CS50	3	Lee	2 000	43	1993-6-20

Beispiel2

kurs $\rightarrow_{monat}$ #studis gilt nicht, denn
1. t1(kurs)=CS50=t2(kurs),
2. t1 in p(1993-5-26) und t2 in p(1993-5-30)
3. G(i)= 1993-5-26, G(j)= 1993-5-30, G(i,j) in H(mai)

aber t1(#studis)=50 und t2(#studis)=45

kurs	credits	wimi	gehalt	#studis	tag
CS50	3	Woo	2 000	50	1993-5-26
CS50	3	Woo	2 000	45	1993-5-30
CS50	3	Woo	2 500	48	1993-6-2
CS50	3	Lee	2 000	46	1993-6-13
CS50	3	Lee	2 000	44	1993-6-16
CS50	3	Lee	2 000	43	1993-6-20

Temporaler Oberschlüssel

- Eine Menge von Attributen X heißt temporaler Oberschlüssel eines Moduls (R, G), wenn $X \rightarrow_G R$ logisch aus der Menge der temporalen funktionalen Abhängigkeiten folgt.
- $X \rightarrow_G Y$ folgt logisch aus TFD, wenn für jedes Modul, in dem alle Abhängigkeiten in TFD gelten, auch $X \rightarrow_G Y$ gilt.
- Wenn zwei Tupel zu (R,G) in derselben Zeiteinheit von G in den Attributen X dieselben Werte haben, dann sind sie insgesamt gleich.
- Trivialerweise ist stets R ein Oberschlüssel zu (R,G).

Temporale Projektion

- Sei M=(R,G,p) und $X \subseteq R$.
  πX (m) ist die Projektion auf (X, G, p1), wobei für alle i p1(i) = πX (p(i))
  mit der üblichen Projektion πX.

Für alle Schnappschüsse i werden die Tupel in m auf die Attribute X projiziert. Das Ergebnis ist $m' = \bigcup_i \pi\, X$

- Sei F die Menge der temporalen funktionalen Abhängigkeiten und Z eine Menge von Attributen, dann ist πZ (F) = $\{X \rightarrow_H Y \mid F \Rightarrow X \rightarrow_H Y, XY \subseteq Z\}$.

Man hat die temporalen funktionalen Abhängigkeiten mit den Attributen in Z.

Temporale Boyce-Codd Normalform

- Sei M=(R,G) ein temporales Modulschema mit F als Menge der temporalen funktionalen Abhängigkeiten.
  M ist in temporaler BCNF, wenn für jede temporale funktionale Abhängigkeit $X \rightarrow_H Y$, die aus F logisch folgt (wobei $X, Y \subseteq R$, $Y \not\subseteq X$, mindestens eine Zeiteinheit von G wird von einer in H abgedeckt) gilt:
  1. $X \rightarrow_G R$ folgt logisch aus F, dh. X ist ein temporaler Oberschlüssel
  2. Für alle $i \neq j$ von G gilt <u>nicht</u>: $X \rightarrow Y \in \pi G(i,j)$ (F).
1. ist die temporale Version der üblichen Oberschlüsselbedingung.
2. verhindert, dass es temporale funktionale Abhängigkeiten mit H gibt, wobei zwei Zeiteinheiten von G durch eine Zeiteinheit von H abgedeckt werden.

technische universität dortmund
Fakultät für Informatik
LS 8

Beispiel

- M=(Kurse, tag, p)
- F:{kurs→credits, wimi$\rightarrow_{monat}$ gehalt, kurs$\rightarrow_{woche}$ wimi, kurs$\rightarrow_{tag}$ #studis}
- F⇒ kurs$\rightarrow_{woche}$ wimi (wobei kurs, wimi $\subseteq$ Kurse, wimi $\not\subseteq$ kurs, ein Tag wird von einer Woche abgedeckt)
- Es soll gelten:
  1. kurs $\rightarrow_{tag}$ credits, wimi, gehalt, #studis -- stimmt
  2. Die temporale Relation auf alle Paare von Tagen projiziert, gibt es dort nicht die funktionale Abhängigkeit kurs→wimi -- stimmt nicht!

  Es gibt zwei Tage derselben Woche, so dass dort kurs→wimi gilt.

technische universität dortmund
Fakultät für Informatik
LS 8

Was wissen wir jetzt?

- Wir haben die Zeit aus einer Datenbankperspektive gesehen.
- Normalerweise wird ein Zeitattribut in einer Datenbank gar nicht anders als andere Attribute behandelt.
- Das kann aber zu irreführenden oder redundanten Schemata führen, wenn wir eigentlich mehrere Granularitäten der Zeit haben.
- Deshalb arbeitet der Bereich der temporalen Datenbanken daran, alle Formalisierungen der Datenbanken auf eine besondere Berücksichtigung der Zeit hin zu erweitern.
- Gesehen haben wir funktionale Abhängigkeiten, Projektion und Normalform.

technische universität dortmund
Fakultät für Informatik
LS 8

Zum Behalten

- Selbst bei normalen Datenbanken sollte man bei Zeitstempeln aufpassen:
  - Gibt es unterschiedliche Granularitäten? Tag, Woche, Monat
  - Besser ist nur <u>eine Granularität je Tabelle</u>, für verschiedene Granularitäten besser verschiedene Tabellen anlegen!
- Wenn unterschiedliche Granularität vorhanden ist:
  - Welche Attribute sind bei welcher Zeiteineit veränderlich?
  - Wenn Attribute bei einer Zeiteinheit nicht verändert werden können, sollen sie auch nicht mit dieser gestempelt werden!
  - Attribute sollen nur mit <u>der Granularität aufgeführt werden, bei der sich ihre Werte ändern!</u>

Lernaufgaben für Ereignisse

- Wie finde ich Ereignisse in Zeitreihen?
- Wie finde ich Episoden (häufige Mengen von Ereignissen in partieller Ordnung) in Ereignissequenzen?
  Wie will ich die Zeit in den Sequenzen darstellen:
  - Absolute Dauer
  - Zeit zwischen Prämisse und Konklusion
  - Relation zwischen Zeitintervallen (vor, während, nach...)

Lernaufgaben

Lernaufgaben bei einer gegebenen Sequenz von Ereignissen:

(Menge von Ereignissen in partieller Ordnung)

1. Finde häufige Episoden in Sequenzen [Mannila et al.]
   - Wenn A auftritt, dann tritt B in der Zeit T auf [Das et al.]
2. Beziehungen zwischen Zeit-Intervallen lernen [Höppner]
   - A startet vor B, B und C sind gleichzeitig, C und D überlappen sich, D endet genau, wenn E anfängt ...

WINEPI

- E sind Attribute, genannt Ereignistypen.
- Ein Ereignis e ist ein Paar (A, t), wobei A in E und t integer.
- Eine Beobachtungssequenz s ist ein Zeitraum von Ts bis Te mit einer Folge s, die aus Ereignissen besteht:
  s=$(<(A_1, t_1), (A_2, t_2), \ldots, (A_n, t_n)>, Ts, Te)$ wobei $t_i \leq t_{i+1}$
  und $Ts \leq t_i < Te$ für alle i=1...n
- Es geht darum, häufige Episoden in Sequenzen zu finden. Analog zu APRIORI.
- Anwendungen aus der Telekommunikation: Einbruchsversuche in ein Netzwerk, häufige Klickfolgen bei einer Web site, Nutzungsprofile,...

(Heikki Mannila, Hannu Toivonen, Inkeri Verkamo "Discovery of frequent episodes in event sequences", Tech. Report C-1997-15 Univ. Helsinki)

Fenster

- Ein Fenster w der Breite win ist ein Tripel (w, ts, te) und enthält die Ereignisse (A, t), bei denen ts ≤ t < te und ts ≤ Te und te > Ts. ACHTUNG, kein Tippfehler! Randereignisse werden so richtig gezählt, sonst kämen sie in weniger Fenstern vor als Ereignisse in der Mitte der Folge.

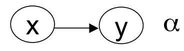
- Die Menge aller Fenster W(s,win) hat die Kardinalität Te-Ts + win-1.

Beispiel

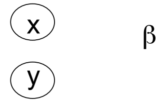
- **s**=(s, 29, 68)
  s=<(E,31), (D, 32), (F,33), (A,35), (B, 37), (C,38),(E,39),(F,40),...,(D,67)>
- Fensterbreite 5 ergibt z.B. die Folge:
  (<(A,35), (B, 37), (C,38),(E,39)>, 35,40)
  4 Ereignisse kommen in den 5 Zeitpunkten vor
  Das Ereignis, das an Zeitpunkt 40 vorkommt, ist nicht im Fenster (s, 35,40), sondern erst in dem (s, 36, 41).
- Das erste Fenster ist ({},25, 30) und das letzte ist (<(D,67)>,67,72).
- (D,67) kommt in 5 Fenstern der Breite 5 vor.
  Genauso oft wie etwa (B,37).
- Es gibt 68-29+5-1= 43 Fenster.

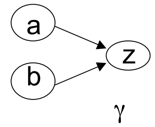
Episoden

- α=(V,≤, g) ist eine serielle Episode, wenn für alle x, y in V gilt: x ≤ y oder y ≤ x. V ist eine Menge von Knoten. g: V → E.
- β=(V, ≤, g) ist eine parallele Episode, wenn die Ordnungsrelation trivial ist (gilt nie).
- β=(V, ≤, g) ∠ γ=(V', ≤', g'),
  wenn es eine eindeutige Abbildung f gibt, f: V→ V'
  so dass g(v)=g'(f(v)) für alle v in V und
  für alle v,w in V mit v ≤ w gilt f(v) ≤'f(w).
- Beispiel: β ist eine Unterepisode von γ, weil
  f(x)=a, f(y)=b
  ≤ ist egal.



g(x)=E,
g(y)=F





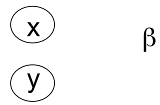
Episode <u>ist in</u> Folge

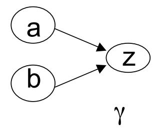
- Eine Episode α=(V,≤, g) ist in einer Folge (occurs in)
  s=(<(A_1, t_1), (A_2, t_2), ..., (A_n, t_n)>, Ts, Te), wenn
  - Es gibt eine eindeutige Abbildung h:V → {1,...,n} so dass g(x)= $A_{h(x)}$ für alle x in V und
  - Für alle x,y ∈V mit x ≠y und x ≤ y gilt: $t_{h(x)} \leq t_{h(y)}$

Beispiel

s=(<(A,35), (B, 37), (C,38),(E,39)>, 35,40)

- Mit g(x)=A, g(y)=B und
  h(x)=1, h(y)=2 ist β in s.
  Es gibt mehrere Abbildungen, so dass β in s ist, weil die Ordnung trivial ist.
- Mit g(a)=A, g(b)=B, g(z)=C und
  h(a)=1, h(b)=2, h(z)=3 ist γ in s
  $t_{h(a)} \leq t_{h(z)}$ und $t_{h(b)} \leq t_{h(z)}$





Häufigkeit einer Episode

- Die Häufigkeit einer Episode α in einer Folge *s* bei einer Fensterbreite win ist

$$fr(\alpha, s, win) = \frac{\left|\{w \in W(s, win) \mid \alpha \text{ ist in } w\}\right|}{\left|W(s, win)\right|}$$

- Wir setzen einen Schwellwert min_fr, so dass α nur häufig ist, wenn fr(α,s,win)≥min_fr.
- Die Menge der häufigen Episoden wird geschrieben als $\mathcal{F}$(s,win,min_fr).

WINEPI: Regeln generieren

- Gegeben eine Menge E von Ereignistypen, eine Ereignisfolge s über E, eine Klasse $\mathcal{E}$ von Episoden, eine Fensterbreite win, ein Schwellwert min_fr und einer min_conf
- Finde Episodenregeln.

1. Berechne $\mathcal{F}$(s, win, min_fr); /* Finde häufige Episoden */
2. For all α in $\mathcal{F}$(s, win, min_fr) do /* Generiere Regeln */
3. for all $\beta \angle \alpha$ do
4. if fr(α)/fr(β) ≥ min_conf then
5. gib aus $\beta \rightarrow \alpha$ mit conf=fr(α)/fr(β);

WINEPI: Finde häufige Episoden

- Gegeben eine Menge E von Ereignistypen, eine Ereignisfolge s über E, eine Klasse $\mathcal{E}$ von Episoden, eine Fensterbreite win und ein Schwellwert min_fr
- Finde die Menge häufiger Episoden $\mathcal{F}$(s,win,min_fr).

1. C_1:={$\alpha \in \mathcal{E}$ | | α |=1 }; /*Erste Kandidaten */
2. ℓ:= 1;
3. While $C_\ell \neq$ { } do
4. $\mathcal{F}_\ell$:={$\alpha \in C_\ell$ | fr(α, s, win) ≥min_fr}; /*Datenbankdurchlauf*/
5. ℓ:= ℓ+1;
6. C_ℓ:={$\alpha \in \mathcal{E}$ | | α |=ℓ und für alle $\beta \in \mathcal{E}$ mit $\beta \angle \alpha$, | β | < ℓ gilt $\beta \in \mathcal{F}_{|\beta|}$}; /*Kandidatengenerierung*/
7. For all ℓ do $\mathcal{F}_\ell$ ausgeben;

Repräsentation

- Episode als Vektor
  - sortiert lexikografisch (parallele Episoden) oder
  - sortiert nach ≤ (serielle Episoden)

  α= A A B C wird geschrieben: $\alpha[1]$=A $\alpha[2]$=A $\alpha[3]$=B $\alpha[4]$=C
- Sortierter Array für die Menge der Episoden
  $\mathcal{F}_\ell[1]$ erste Episode der Länge ℓ
  - sortiert nach gemeinsamen Unterepisoden der Länge ℓ-1

  $\mathcal{F}_4$: [1] A A B C
  [2] A A B D
  [3] A A B F
  - D.h.:Wenn $\mathcal{F}_\ell$[i] und $\mathcal{F}_\ell$[j] in den ersten ℓ-1 Ereignissen übereinstimmen, dann auch alle $\mathcal{F}_\ell$[k] mit i< k < j.

  $\mathcal{F}_4[1]$, $\mathcal{F}_4[3]$ stimmen in den ersten 3 Ereignissen überein, so auch $\mathcal{F}_4[2]$.

Kandidatengenerierung -- Idee

- Aus häufigen Episoden sollen um eins längere Episoden generiert werden.
- Die längste Abfolge von Sequenzen i=1,...,m mit denselben ℓ-1 Ereignissen heißt ein Block.
- Innerhalb eines Blockes werden alle Episoden (an ℓter Stelle) kombiniert, um solche der Länge ℓ+1 zu generieren.

$\mathcal{F}_\ell$ → $C_{\ell+1}$

i,j↓ ℓ→	1	2...	ℓ
1	A	B	C
...			
m	A	B	F
m+1	A	C	D

$\mathcal{F}_\ell$.blockstart[1]=1
$\mathcal{F}_\ell$.blockstart[2]=1
...
$\mathcal{F}_\ell$.blockstart[m]=1
$\mathcal{F}_\ell$.blockstart[m+1]=m+1

WINEPI: Kandidatengenerierung1

- Gegeben ein sortiertes Array $\mathcal{F}_\ell$ von häufigen parallelen Episoden der Länge ℓ
- Finde ein sortiertes Array paralleler Episoden der Länge ℓ+1 als Kandidaten.


```
1.  C_{ℓ+1}:={ };
2.  k:=0;
3.  If ℓ=1 then for x:=1 to |F_ℓ| do F_ℓ.blockstart[h]=1;
4.  For i:=1 to |F_ℓ| do              /*Ein i nach dem anderen durchgehen */
5.      Current_blockstart:=k+1;
6.      For (j:=i; F_ℓ.blockstart[i]= F_ℓ.blockstart[j];j:=j+1) do /*j läuft */
7.          For x:=1 to ℓ do α[x]:= F_ℓ[i][x]; α[ℓ+1]:= F_ℓ[j][ℓ];
8.          For y:=1 to ℓ-1 do   /* Unterepisoden sollen in F_ℓ vorkommen*/
9.              For x:=1 to y-1 do β[x]:= α[x];
10.             For x:=y to ℓ do β[x]:= α[x+1];
11.             If β ist nicht in F_ℓ, then gehe zum nächsten j in Zeile 6,
        else speichere α als Kandidat.
12.     k:=k+1;
13.     C_{ℓ+1}[k]:=a;
14.     C_{ℓ+1}.blockstart[k]:=current_blockstart;
15. Output C_{ℓ+1};
```


Komplexität der Kandidatengenerierung

- Theorem: Die Kandidatengenerierung hat die Komplexität $O(\ell^2 \, |\mathcal{F}_\ell|^2 \log |\mathcal{F}_\ell|)$.
- Beweis: Zeile 3 braucht $O(|\mathcal{F}_\ell|)$.
  Die äußere Schleife (Zeile 4) wird $O(|\mathcal{F}_\ell|)$ mal durchlaufen. Die innere Schleife (Zeile 6) wird $O(|\mathcal{F}_\ell|)$ mal durchlaufen. In den Schleifen werden Kandidaten (Zeile 7) und Unterepisoden (Zeile 8-10) konstruiert in der Zeit
  $O(\ell+1+\ell(\ell-1))$.
  Die ℓ-1 Unterepisoden werden in $\mathcal{F}_\ell$ gesucht (Zeile 11). Da $\mathcal{F}_\ell$ sortiert ist, gelingt dies in $O(\ell \log |\mathcal{F}_\ell|)$.
  $O(|\mathcal{F}_\ell| + |\mathcal{F}_\ell| \, |\mathcal{F}_\ell| \, (\ell^2 + \ell(\ell-1)) \, \ell \log |\mathcal{F}_\ell|) = O(\ell^2 \, |\mathcal{F}_\ell|^2 \log |\mathcal{F}_\ell|)$.
  Q.e.d.

Datenbankdurchlauf -- Idee

- Contains(A,a) enthält alle Episoden, in denen der Ereignistyp A genau a mal vorkommt. So werden parallele Episoden über ihre Attribute indexiert.
- α.event_count speichert, wie viele Ereignisse von α in Fenster w vorkommen.
- Wenn $|\alpha|$ Ereignisse in w vorkommen, speichern wir ts von w in α.in_window. Das war der Anfang eines Fensters mit der vollständigen Episode.
- Wenn α.event_count abnimmt, wird α.freq_count um die Anzahl von Fenstern erhöht, in denen die gesamte Episode vorkam, d.h. α.event_count = $|\alpha|$. So wird bei jeder Episode hochgezählt, in wie vielen Fenstern sie vorkam.

Beispiel

C_3[1].event_count=3
C_3[1].in_window=35
C_3[1].freq_count

w=(<(A,35), (B, 37), (C,38),(E,39)>, 35,40)

Contains(B,1) C_3[1], C_3[2], C_3[3]

Contains(B,2) C_3[2]

Contains(C,1) C_3[1], C_3[4]

Contains(A,1) C_3[1], C_3[2], C_3[3], C_3[4]

Contains(D,1) C_3[4]

Contains(F,1) C_3[3]

C_3

	1	2	3
1	A	B	C
2	A	B	B
3	A	B	F
4	A	C	D

Update der Fenster

- Beim Verschieben der Fenster von w nach w' bleiben die meisten Ereignisse dieselben: nur ein Ereignis kommt hinzu und ein Ereignis verschwindet.
  - Alle Episoden mit dem neuen Ereignistyp A können über contains(A,1) erreicht und ihr event_count um 1 erhöht werden.
  - War bereits ein Vorkommen von A in Fenster w, so können die passenden Episoden über contains(A,2) erreicht und ihr event_count um 1 erhöht werden.

Datenbankdurchlauf

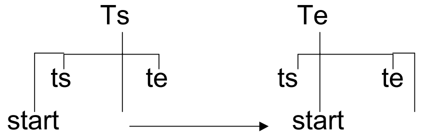
- Gegeben: Eine Sammlung von Episoden $\mathcal{C}$, eine Ereignissequenz **s**=(s, Ts, Te), eine Fensterbreite win, eine Häufigkeitsschranke min_fr.
- Finde die Episoden von $\mathcal{C}$, die häufig in **s** vorkommen bzgl. win und min_fr.

Datenbankdurchlauf1: Initialisierung

1. For each α in C do
2. For each A in α do /* Initialisieren mit 0 */
3. A.count:=0;
4. For i:=1 to $|\alpha|$ do contains(A,i):={ };
5. For each α in C do /* Struktur aufbauen */
6. For each A in α do
7. a:=Anzahl von Ereignissen des Typs A in α;
8. contains(A,a):=contains(A,a) $\cup$ {α};
9. α.event_count:=0; /* Initialisieren mit 0 */
10. α.freq_count:=0;

Datenbankdurchlauf2: neue Ereignisse

1. For start:=Ts – win+1 to Te do /* neue Ereignisse in w' */
2. For all (A, t) in s mit t=start+win – 1 do
3. A.count:=A.count+1;
4. For each α in contains(A,A.count) do
5. α.event_count:= α.event_count+A.count;
6. If α.event_count= $|\alpha|$ then α.in_window:=start;



Datenbankdurchlauf3: alte Ereignisse

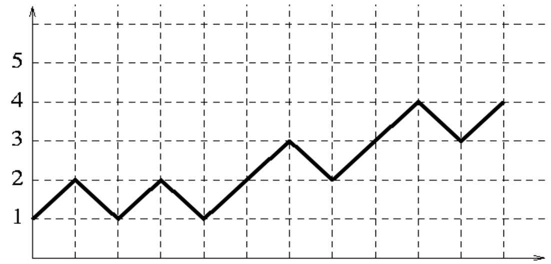
1. For all (A, t) in s mit t=start – 1 do
2. For each α in contains(A,A.count) do
3. If α.event_count= $|\alpha|$ then
4. α.freq_count:= α.freq_count- α.in_window+start;
5. α.event_count:= α.event_count – A.count;
6. A.count:=A.count – 1;
7. For all Episoden α in C do /* Ausgabe*/
8. If α.freq_count/(Te-Ts+win-1) min_fr then output α;

Komplexität des Datenbankdurchlaufs

- Theorem: Die Komplexität des Datenbankdurchlaufs für parallele Episoden ist O((n+ℓ^2) $|C|$), wobei alle Episoden die Länge ℓ haben und n die Länge der Sequenz ist.
- Initialisierung braucht O((n+ℓ^2) $|C|$).
  In den innersten Schleifen bei neuen Ereignissen (Zeile 4) und bei alten Ereignissen (Zeile 5) wird so oft auf α.event_count zugegriffen wie sich das Fenster verschiebt: O(n). Dies kann allen Episoden passieren: $|C|$. Der update wegen neuer und alter Ereignisse braucht also O(n $|C|$).
  Q.e.d.

Clustering Vorbereitung

Zeitreihe $s = (x_1,\ldots,x_n)$ in Subsequenzen $s_i = (x_i,\ldots,x_{i+w-1})$ aufteilen

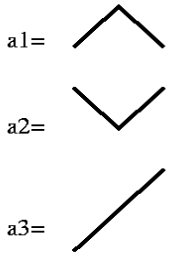


Fenster der Bereite w = 3

⇨ Schritt 2

Clustering

Distanzmaß $d(s_i,s_j)$: Entfernung zwischen zwei Subsequenzen
Bsp.: Euklidischer Abstand $(\Sigma(x_i-y_i)^2)^{0,5}$
Konstante $d > 0$: gibt an, wie groß der Unterschied zwischen den Subsequenzen sein darf

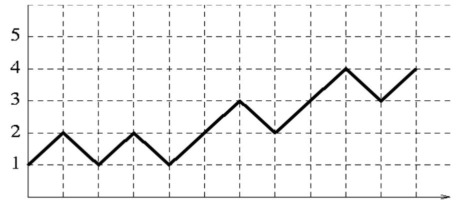


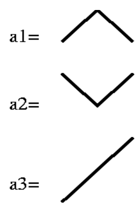
Bilde aus der Menge aller Subseqenzen Cluster $C_1,\ldots C_k$

Jedes Cluster erhält ein Symbol $a_1,..a_k$ („Shapes")

Anwendung des Clustering

Die Serie $s = (x_1,\ldots,x_n)$ kann jetzt mit Hilfe der shapes beschrieben werden („diskretisiert")





Original time series = (1, 2, 1, 2, 1, 2, 3, 2, 3, 4, 3, 4)
Window width = 3
Discretized series = (a1, a2, a1, a2, a3, a1, a2, a3, a1, a2)

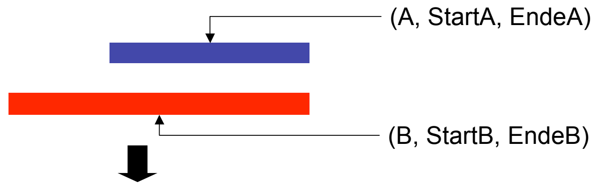
Primitive shapes after clustering

Regeln in diskreten Sequenzen

- Regeln der Form Wenn A auftritt, dann tritt B in der Zeit T auf einfach ableitbar mithhilfe APRIORI
- Berechnung in der Zeit m*k² möglich
  - (k=Anzahl der Symbole, m = #verschiedene Möglichkeiten für T)
- Erweiterung:
  - Wenn A_1 und A_2 und ... und A_h innerhalb der Zeit V auftritt, dann tritt B in der Zeit T auf
  - Microsoft ↓ (1), Microsoft ↑ (2) + Intel → (2) ⇒ IBM → (3)
  - Problem: Anzahl der Regeln steigt stark an

Beziehungen zwischen Ereignissen

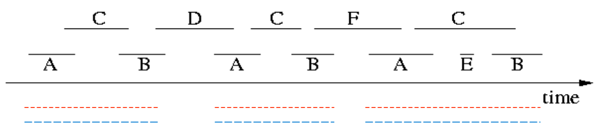
- Von James F. Allen wurden 13 verschiedene Intervallbeziehungen festgelegt:
  - A überlappt B, A beendet B, A vor B, A enthält B, ...
- Beispiel: A beendet B



StartB<StartA, EndeA = EndeB,

Beziehungen zwischen Zeit-Intervallen lernen [Höppner]

state interval sequence:



Darstellung der Beziehungen als Matrix:

R1

	A	B
A	=	b
B	a	=

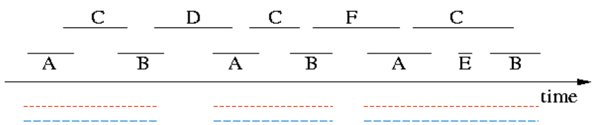
R2

	A	B	C
A	=	b	o
B	a	=	io
C	io	o	=

(abbreviations: a=after, b=before, o=overlaps, io=is-overlapped-by)

Regeln

state interval sequence:



Die Regeln sind von der Form **P → R**

Prämisse P

	A	B
A	=	b
B	a	=

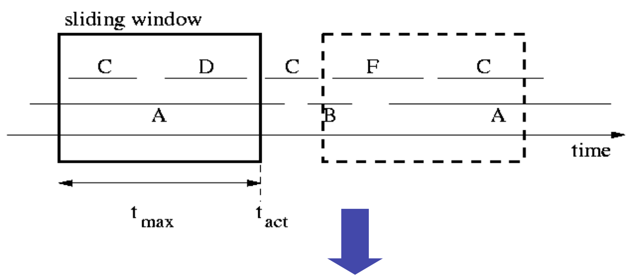
Regel R

	A	B	C
A	=	b	o
B	a	=	io
C	io	o	=

Beispiel: A, B, C sind Verträge verschiedener Kategorien

Häufige Muster finden

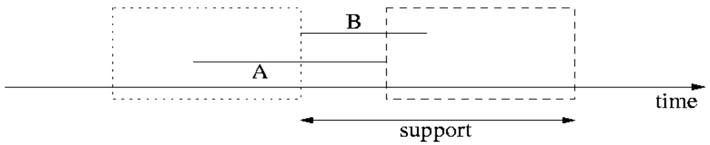
Muster muss im Fenster der Länge t_{max} beobachtbar sein



Der maximale Abstand zwischen den Ereignissen eines Muster ist begrenzt

Was bedeutet häufig?

Als Maß für die Häufigkeit von Mustern dient der „Support“



Ein Muster wird als häufig erachtet, wenn es einen Support > $supp_{min}$ hat

	A	B
A	=	o
B	io	=

Anwendung von APRIORI

- Ermittle den Support aller 1-Muster
- Im k-ten Lauf:
  - entferne alle Muster mit supp<$supp_{min}$
  - generiere aus den verbliebenen k-Mustern eine Menge von Kandidaten für k+1-Muster
  - ermittle den Support der Kandidaten im nächsten Lauf
- Wiederhole diese Schritte, bis keine häufigen Muster mehr gefunden werden können
- Generiere die Regeln aus den häufigen Mustern

Was wissen Sie jetzt?

- Man kann den Apriori Algorithmus für die Entdeckung von Zeitsequenzen anwenden.
- Der Ansatz von Gaudam Das et alii:
  - Fenster werden über die Zeitreihe geschoben
  - Die so erhaltenen Subsequenzen werden durch ein Distanzmaß ge-cluster-t. Es entstehen Muster wie aufsteigend, absteigend.
  - Mit den Mustern als Eingabe werden Assoziationsregeln gelernt.
- Der Ansatz von Frank Höppner:
  - Fenster werden über die Zeitreihe geschoben
  - Matritzen zu Allens Intervallen angelegt
  - Häufige, möglichst lange Sequenzen werden ermittelt und Assoziationsregeln gelernt.