Recognizing episodes in sequences

- first problem: given a sequence and an episode, find out whether the episode occurs in the sequence
- finding the number of windows containing an occurrence of the episode can be reduced to this
- successive windows have a lot in common
- how to use this?
- an incremental algorithm

Parallel episodes

- for each candidate α maintain a counter $\alpha.\mathit{event\_count}$: how many events of α are present in the window
- When $\alpha.\mathit{event\_count}$ becomes equal to $|\alpha|$, indicating that α is entirely included in the window
  - save the starting time of the window in $\alpha.\mathit{inwindow}$
- when $\alpha.\mathit{event\_count}$ decreases again, increase the field $\alpha.\mathit{freq\_count}$ by the number of windows where α remained entirely in the window

Algorithm

Input: A collection $\mathcal{C}$ of parallel episodes, an event sequence $\mathbf{s} = (s, T_s, T_e)$, a window width *win*, and a frequency threshold *min_fr*.

Output: The episodes of $\mathcal{C}$ that are frequent in $\mathbf{s}$ with respect to *win* and *min_fr*.

Method:

```
// Initialization:
for each α in C do
    for each A in α do
        A.count := 0;
        for i := 1 to |α| do contains(A, i) := ∅;
for each α in C do
    for each A in α do
        a := number of events of type A in α;
        contains(A, a) := contains(A, a) ∪ {α};
    α.event_count := 0;
    α.freq_count := 0;
```

Algorithm Method:

```
// Recognition:
for start := T_s − win + 1 to T_e do
    // Bring in new events to the window:
    for all events (A, t) in s such that t = start + win − 1 do
        A.count := A.count + 1;
        for each α ∈ contains(A, A.count) do
            α.event_count := α.event_count + A.count;
            if α.event_count = |α| then α.inwindow := start;
    // Drop out old events from the window:
    for all events (A, t) in s such that t = start − 1 do
        for each α ∈ contains(A, A.count) do
            if α.event_count = |α| then
                α.freq_count := α.freq_count − α.inwindow + start;
            α.event_count := α.event_count − A.count;
        A.count := A.count − 1;
// Output:
for all episodes α in C do
    if α.freq_count/(T_e − T_s + win − 1) ≥ min_fr then output α;
```

Theorem 1 *Algorithm 102 works correctly.*

Proof *We consider the following two invariants. (1) For each event type A that occurs in any episode, the variable A.count correctly contains the number of events of type A in the current window. (2) For each episode α, the counter α.event_count equals $|\alpha|$ exactly when α occurs in the current window.* □

Complexity

Assume that exactly one event takes place every time unit.

Assume candidate episodes are all of size l, and let n be the length of the sequence.

Theorem 2 *The time complexity of Algorithm 102 is* $\mathcal{O}((n + l^2)\,|\mathcal{C}|)$.

Proof *Initialization takes time* $\mathcal{O}(|\mathcal{C}|\, l^2)$.

How many accesses to α.event_count *on lines 7 and 14.*

In the recognition phase there are $\mathcal{O}(n)$ *shifts of the window. In each shift, one new event comes into the window, and one old event leaves the window. Thus, for any episode* α, α.event_count *is accessed at most twice during one shift.*

The cost of the recognition phase is thus $\mathcal{O}(n\,|\mathcal{C}|)$. □

Serial episodes

- use state automata that accept the candidate episodes
- example: episode A B A B

General episodes

different alternatives

Window width (s)	Serial episodes Count	Serial episodes Time (s)	Injective parallel episodes Count	Injective parallel episodes Time (s)
10	16	31	10	8
20	31	63	17	9
40	57	117	33	14
60	87	186	56	15
80	145	271	95	21
100	245	372	139	21
120	359	478	189	22

Table 4.1: Results of experiments with s_1 using a fixed frequency threshold of 0.003 and a varying window width

Frequency threshold	Serial episodes Count	Serial episodes Time (s)	Injective parallel episodes Count	Injective parallel episodes Time (s)
0.1	0	7	0	5
0.05	1	12	1	5
0.008	30	62	19	14
0.004	60	100	40	15
0.002	150	407	93	22
0.001	357	490	185	22

Table 4.2: Results of experiments with s_1 using a fixed window width of 60 s and a varying frequency threshold

Episode size	Number of episodes	Number of candidate episodes	Number of frequent episodes	Match
1	287	287.0	30.1	11 %
2	82 369	1 078.7	44.6	4 %
3	$2 \cdot 10^7$	192.4	20.0	10 %
4	$7 \cdot 10^9$	17.4	10.1	58 %
5	$2 \cdot 10^{12}$	7.1	5.3	74 %
6	$6 \cdot 10^{14}$	4.7	2.9	61 %
7	$2 \cdot 10^{17}$	2.9	2.1	75 %
8	$5 \cdot 10^{19}$	2.1	1.7	80 %
9	$1 \cdot 10^{22}$	1.7	1.4	83 %
10–		17.4	16.0	92 %

Table 4.3: Number of candidate and frequent serial episodes in $\mathbf{s}_1$ with frequency threshold 0.003 and averaged over window widths 10, 20, 40, 60, 80, 100, and 120 s

Experiences in alarm correlation

Useful in

- finding long-term, rather frequently occurring dependencies,
- creating an overview of a short-term alarm sequence, and
- evaluating the consistency and correctness of alarm databases
- discovered rules have been applied in alarm correlation
- lots of rules are trivial

Chapter 5: Minimal occurrences of episodes

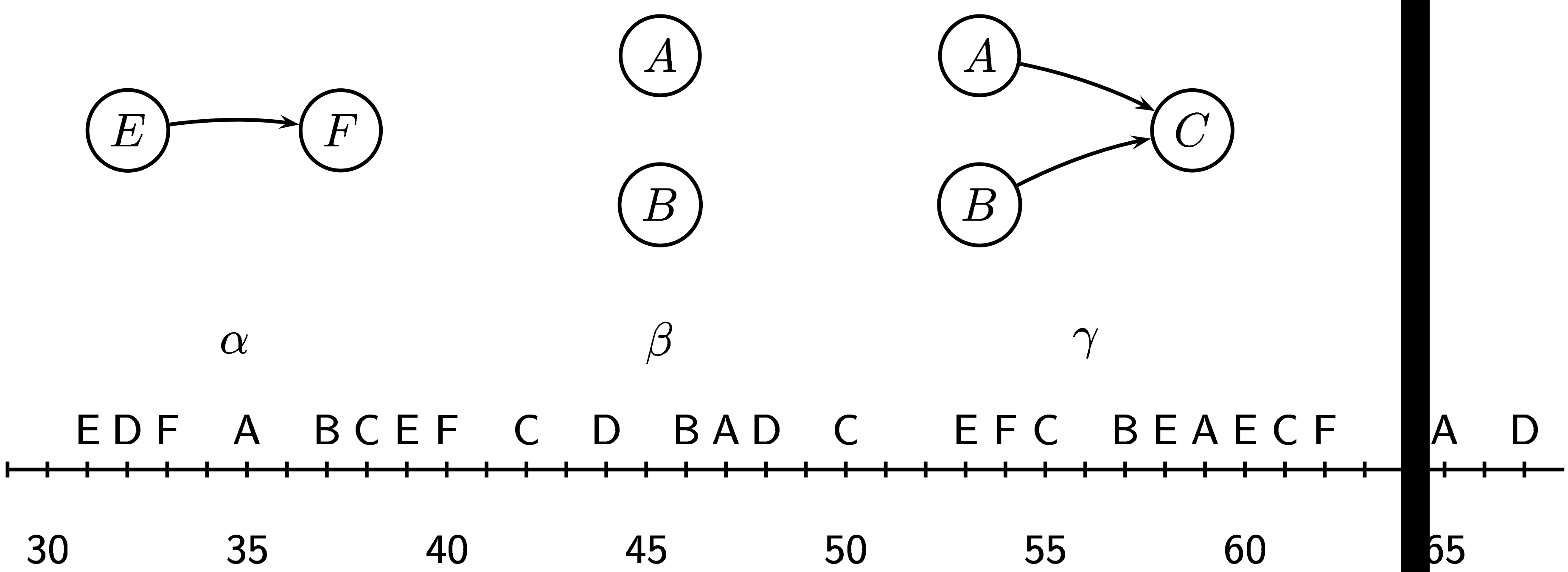
5. Minimal occurrences of episodes

- an alternative approach to discovery of episodes
- no windows
- for each potentially interesting episode, find out the exact occurrences of the episode
- advantages: easy to modify time limits, several time limits for one rule ("if A and B occur within 15 seconds, then C follows within 30 seconds")
- disadvantages: uses lots of space

Definitions

- an episode α and an event sequence $\mathbf{s}$
- interval $[t_s, t_e)$ is a *minimal occurrence* of α in $\mathbf{s}$, if
  - α occurs in the window $\mathbf{w} = (w, t_s, t_e)$ on $\mathbf{s}$
  - α does not occur in any proper subwindow on $\mathbf{w}$
- *set of (intervals of) minimal occurrences* of an episode α:
  $mo(\alpha) = \{ [t_s, t_e) \mid [t_s, t_e)$ is a minimal occurrence of $\alpha \}$.

Example



Figures 5.1: Episodes and 5.2: The example event sequence s

$\textit{mo}(\beta) = \{[35, 38), [46, 48), [47, 58), [57, 60)\}$.

$\textit{mo}(\gamma) = \{[35, 39), [46, 51), [57, 62)\}$.

Episodes rules, new version

- *episode rule*: $\beta\,[\mathit{win}_1] \;\Rightarrow\; \alpha\,[\mathit{win}_2]$,
- β and α are episodes such that $\beta \preceq \alpha$
- win_1 and win_2 are integers
- if episode β has a minimal occurrence at interval $[t_s, t_e)$ with $t_e - t_s \leq \mathit{win}_1$, then episode α occurs at interval $[t_s, t_e')$ for some t_e' such that $t_e' - t_s \leq \mathit{win}_2$
- (old version: $\beta\,[w] \;\Rightarrow\; \alpha\,[w]$, in windows containing β)

- formally: $mo_{win_1}(\beta) = \{[t_s, t_e) \in mo(\beta) \mid t_e - t_s \leq win_1\}$
- given α and an interval $[u_s, u_e)$, define $occ(\alpha, [u_s, u_e)) =$ true if and only if there exists a minimal occurrence $[u'_s, u'_e) \in mo(\alpha)$ such that $u_s \leq u'_s$ and $u'_e \leq u_e$
- The confidence of an episode rule $\beta\,[win_1] \Rightarrow \alpha\,[win_2]$ is now

$$\frac{|\{[t_s, t_e) \in mo_{win_1}(\beta) \mid occ(\alpha, [t_s, t_s + win_2))\}|}{|mo_{win_1}(\beta)|}.$$

Example, cont.

- $\beta\,[3] \Rightarrow \gamma\,[4]$
- three minimal occurrences $[35, 38), [46, 48), [57, 60)$ of β of width at most 3 in the denominator
- Only $[35, 38)$, has an occurrence of α within width 4, so the confidence is $1/3$.
- rule $\beta\,[3] \Rightarrow \gamma\,[5]$ the confidence is 1.

Rule forms

- temporal relationships can be complex

Frequency and support

- previously: frequency = fraction of windows containing the episode
- no fixed window size
- several minimal occurrences within a window
- support of an episode: the number of minimal occurrences of an episode, $|mo(\alpha)|$

Rule discovery task

- an event sequence $\mathbf{s}$
- a frequency threshold *min_fr*
- a class $\mathcal{E}$ of episodes
- a set W of time bounds
- find all frequent episode rules of the form $\beta\,[win_1] \Rightarrow \alpha\,[win_2]$
- $\beta, \alpha \in \mathcal{E}$ and $win_1, win_2 \in W$.

Chapter 6: Episode discovery process

6. Episode discovery process

- The knowledge discovery process
- KDD process of analyzing alarm sequences
- Discovery and post-processing of large pattern collections
- TASA, Telecommunication Alarm Sequence Analyzer

The knowledge discovery process

Goal: discovery of useful and interesting knowledge

1. Understanding the domain
2. Collecting and cleaning data
3. Discovery of patterns
4. Presentation and analysis of results
5. Making onclusions and utilizing results

Pattern discovery is only a part of the KDD process (but the central one)

The knowledge discovery process

Questions implied by the KDD process model:

- How to know what could be interesting?
- How to ensure that correct and reliable discoveries can be made?
- How to discover potentially interesting patterns?
- How to make the results understandable for the user?
- How to use the results?

Episode discovery process for alarm sequences

Collecting and cleaning the data

- Can take a lot of time
- Collection of alarms rather easy
- Data cleaning? Inaccuracy of clocks
- Missing data?
- What are the event types?
  - Alarm type? Network element? A combination of the two?
- How to deal with background knowledge: network topology, object hierarchies for network elements
- "Alarm predicates": properties of alarms

Discovery of patterns

Strategy:

1. Find *all* potentially interesting patterns
   $\Rightarrow$ lots of rules
2. Allow users to explore the patterns iteratively and interactively

1. All potentially interesting patterns
   - Episodes: combination of alarms
   - Association rules: what are alarms like
   - Frequency and confidence thresholds
   - Background knowledge coded into alarm predicates in various alternative ways
   - Network topology used to constrain patterns

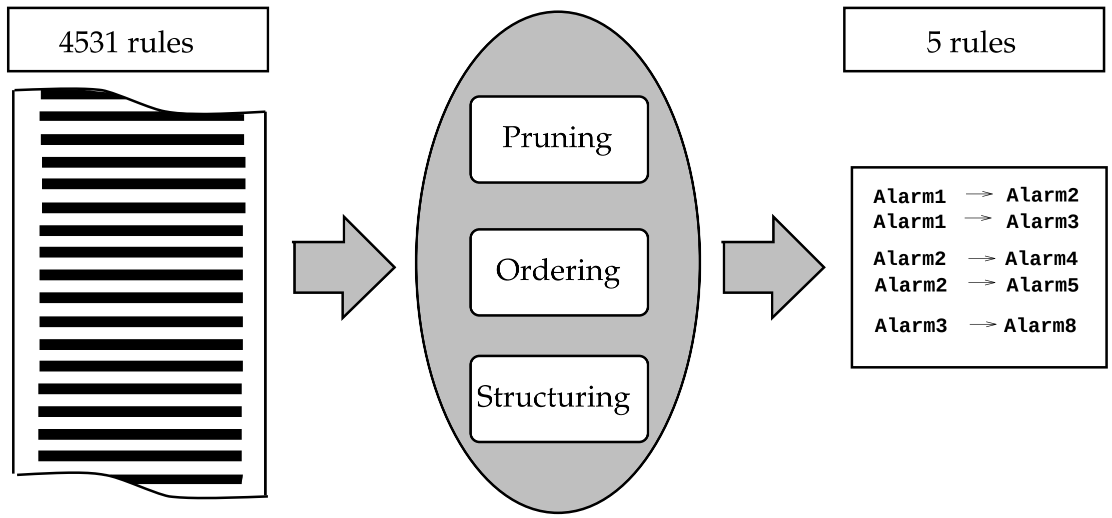
Presentation and analysis of results

There can be lots of rules

- only a small part is really interesting
- – subjective
  – hard to define in advance
  – can depend on the case
- also expected regularities (or their absence) can be of interest

⇒ iteration is necessary

⇒ support for personal views is needed



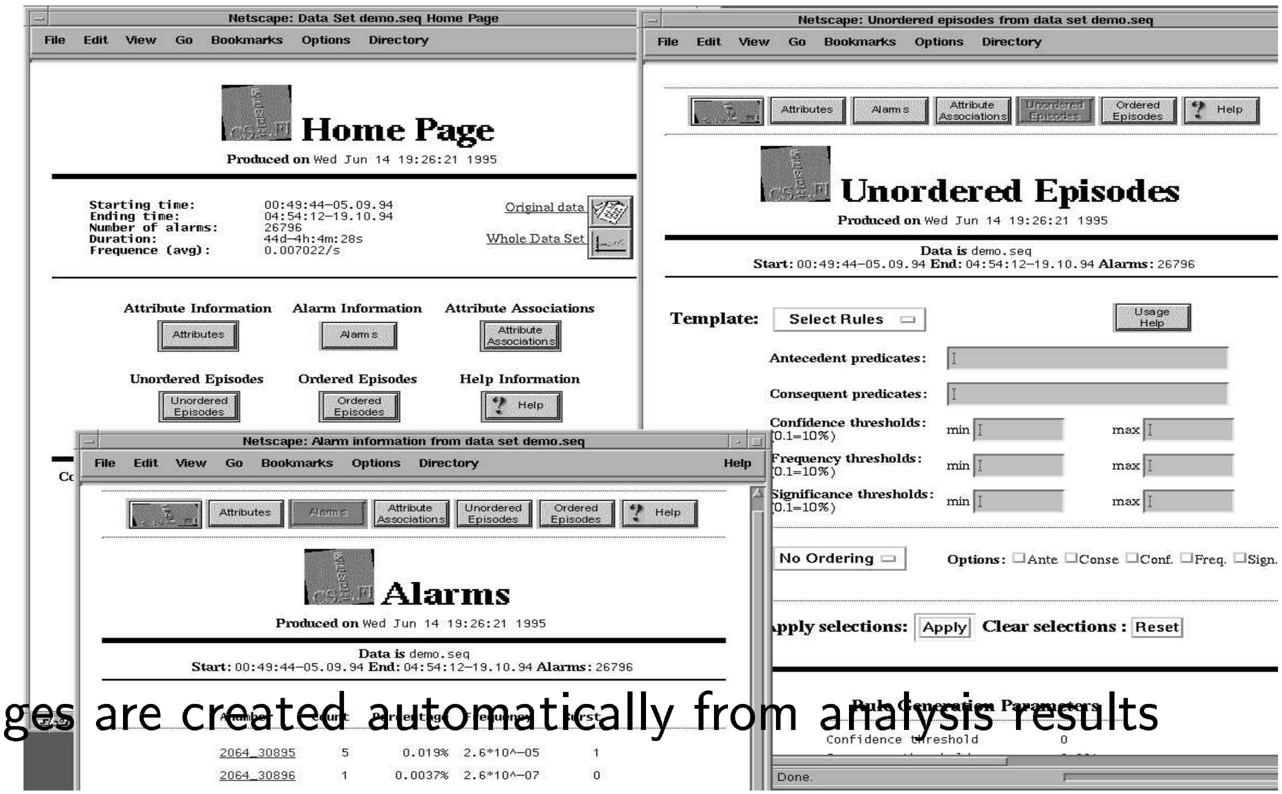
Pruning and ordering:

- alarm predicates on the left or right side
- confidence, frequency, statistical significance

Structuring:

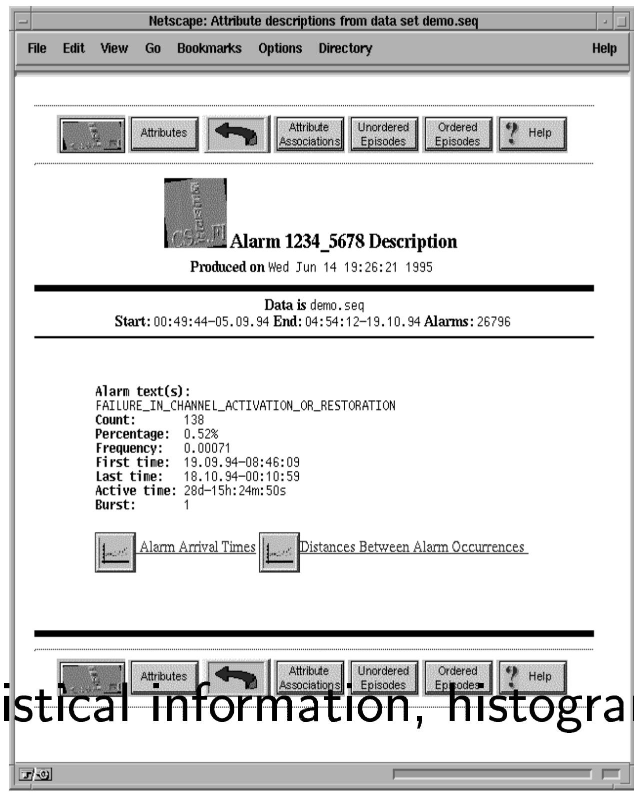
- clusters, hierarchies, etc.

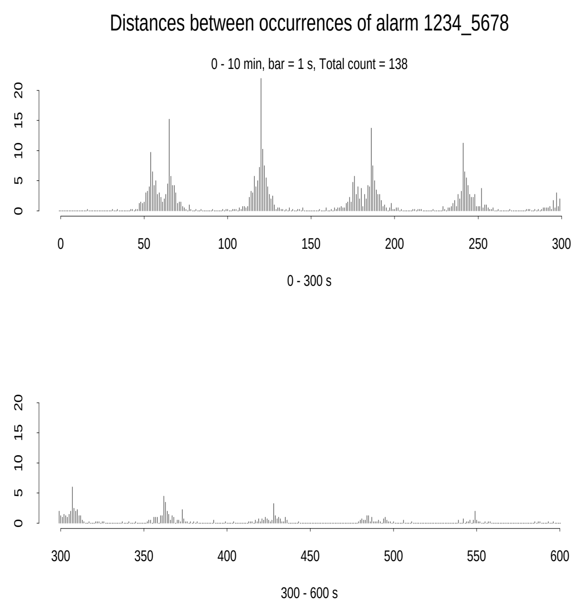
TASA: A KDD tool for alarm analysis



- pages are created automatically from analysis results

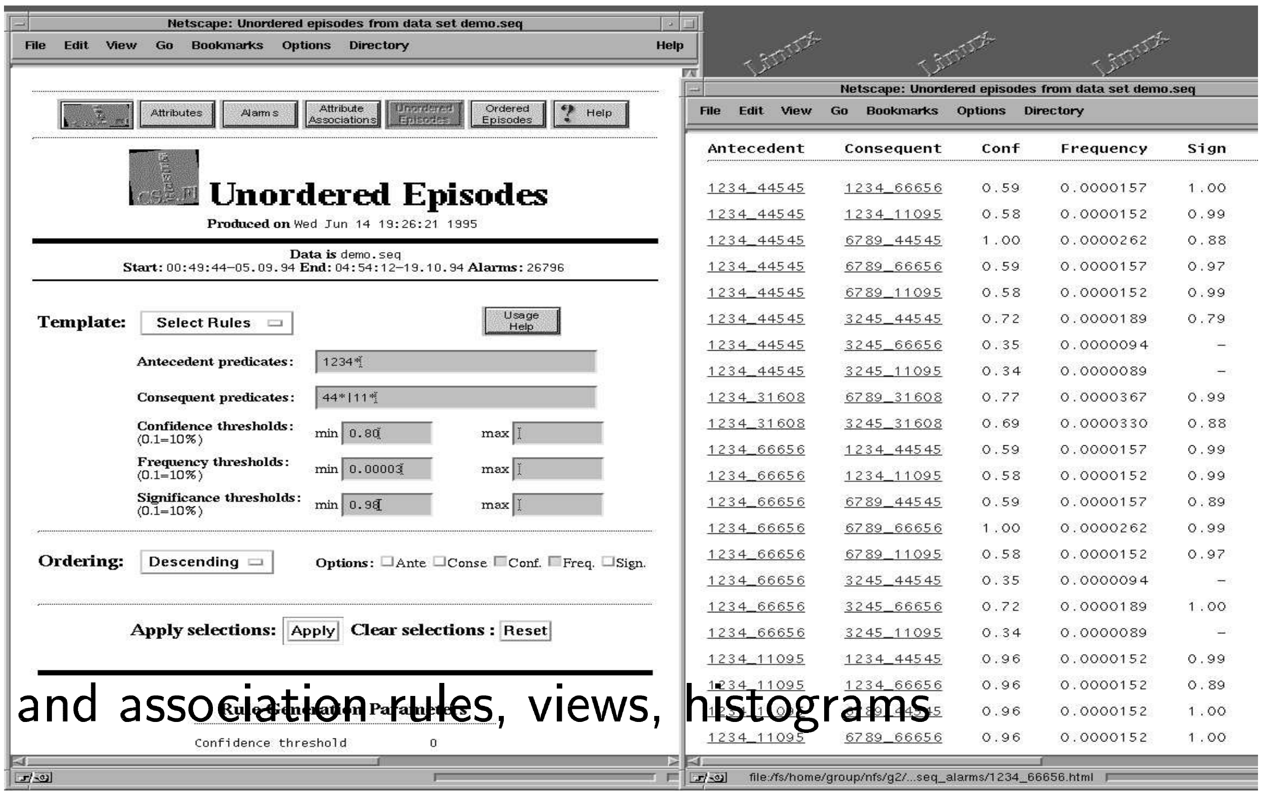
TASA: Giving an overview of data





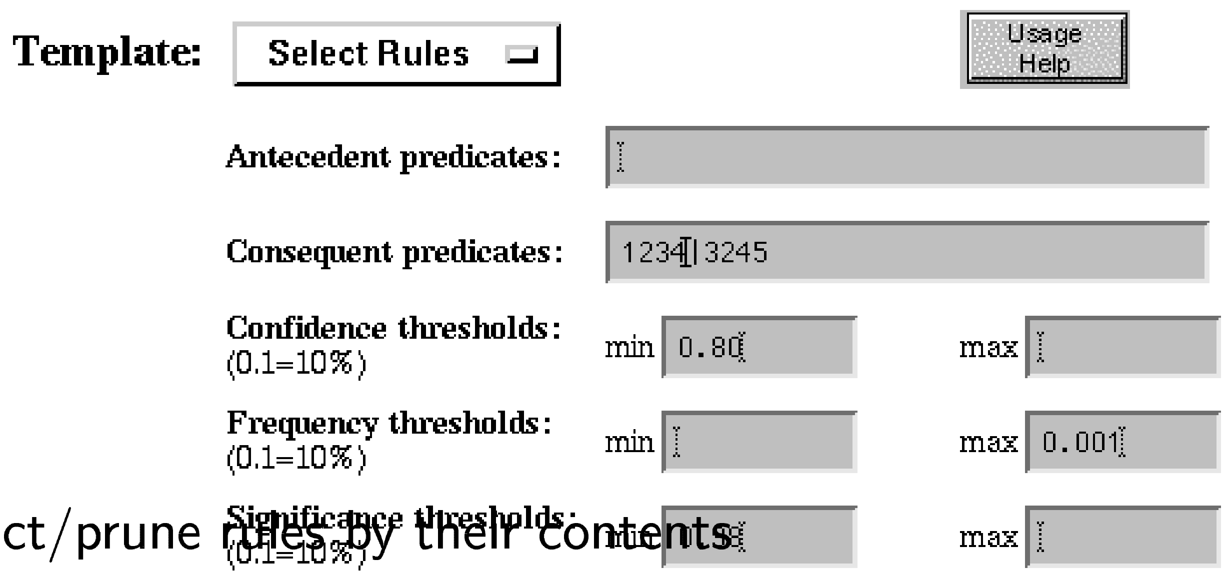
statistical information, histograms

TASA: Rule presentation



episode and association rules, views, histograms

TASA: Views with templates



- select/prune rules by their contents
  ⇒ **iteration!**
- criteria: left-hand/right-hand side of the rule, thresholds

Chapter 7: Generalized framework

7. Generalized framework

- given a set of patterns, a selection criterion, and a database
- find those patterns that satisfy the criterion in the database
- what has to be required from the patterns
- a general levelwise algorithm
- analysis in Chapter 8

Relational databases

- a *relation schema* R is a set $\{A_1, \ldots, A_m\}$ of *attributes*.
- each attribute A_i has a *domain* $Dom(A_i)$
- a *row* over a R is a sequence $\langle a_1, \ldots, a_m \rangle$ such that $a_1 \in Dom(A_i)$ for all $i = 1, \ldots, m$
- the ith value of t is denoted by $t[A_i]$
- a *relation* over R is a set of rows over R
- a *relational database* is a set of relations over a set of relation schema (the *database schema*)

Discovery task

- $\mathcal{P}$ is a set of *patterns*
- q is a *selection criterion*, i.e., a predicate $q : \mathcal{P} \times \{\mathbf{r} \mid \mathbf{r} \text{ is a database}\} \to \{\text{true, false}\}$.
- φ is *selected* if $q(\varphi, \mathbf{r})$ is true
- *frequent* as a synonym for "selected".
- give a database $\mathbf{r}$, the *theory* $\mathcal{T}(\mathcal{P}, \mathbf{r}, q)$ of $\mathbf{r}$ with respect to $\mathcal{P}$ and q is $\mathcal{T}(\mathcal{P}, \mathbf{r}, q) = \{\varphi \in \mathcal{P} \mid q(\varphi, \mathbf{r}) \text{ is true}\}$.

Example

finding all frequent item sets

- a set R a binary database r over R, a frequency threshold *min_fr*
- $\mathcal{P} = \{X \mid X \subseteq R\}$,
- $q(\varphi, r) =$ true if and only if *fr*$(\varphi, r) \geq$ *min_fr*

Selection predicate

- no semantics given for the patterns
- selection criterion takes care of that
- "$q(\varphi, \mathbf{r})$ is true" can mean different things:
- φ occurs often enough in $\mathbf{r}$
- φ is true or almost true in $\mathbf{r}$
- φ defines, in some way, an interesting property or subgroup of $\mathbf{r}$
- determining the theory of $\mathbf{r}$ is not tractable for arbitrary sets $\mathcal{P}$ and predicates q

Methodological point

- find all patterns that are selected by a relatively simple criterion—such as exceeding a frequency threshold—in order to efficiently identify a space of potentially interesting patterns
- other criteria can then be used for further pruning and processing of the patterns
- e.g., association rules or episode rules

Specialization relation

- $\mathcal{P}$ be a set of patterns, q a selection criterion over $\mathcal{P}$
- $\preceq$ a partial order on the patterns in $\mathcal{P}$
- if for all databases $\mathbf{r}$ and patterns $\varphi, \theta \in \mathcal{P}$ we have that $q(\varphi, \mathbf{r})$ and $\theta \preceq \varphi$ imply $q(\theta, \mathbf{r})$,
- then $\preceq$ is a *specialization relation* on $\mathcal{P}$ with respect to q
- $\theta \preceq \varphi$, then φ is said to be *more special* than θ and θ to be *more general* than φ
- $\theta \prec \varphi$: $\theta \preceq \varphi$ and not $\varphi \preceq \theta$
- the set inclusion relation $\subseteq$ is a specialization relation for frequent sets

Generic levelwise algorithm

- the *level* of a pattern φ in $\mathcal{P}$, denoted *level*(φ), is 1 if there is no θ in $\mathcal{P}$ for which $\theta \prec \varphi$.
- otherwise *level*(φ) is $1 + L$, where L is the maximum level of patterns θ in $\mathcal{P}$ for which $\theta \prec \varphi$
- the collection of frequent patterns of level l is denoted by $\mathcal{T}_l(\mathcal{P}, \mathbf{r}, q) = \{\varphi \in \mathcal{T}(\mathcal{P}, \mathbf{r}, q) \mid \textit{level}(\varphi) = l\}$.

Algorithm 7.6

Input: A database schema $\mathbf{R}$, a database $\mathbf{r}$ over $\mathbf{R}$, a finite set $\mathcal{P}$ of patterns, a computable selection criterion q over $\mathcal{P}$, and a computable specialization relation $\preceq$ on $\mathcal{P}$.

Output: The set $\mathcal{T}(\mathcal{P}, \mathbf{r}, q)$ of all frequent patterns.

Method:

1. compute $\mathcal{C}_1 := \{\varphi \in \mathcal{P} \mid \mathit{level}(\varphi) = 1\}$;
2. $l := 1$;
3. **while** $\mathcal{C}_l \neq \emptyset$ **do**
4. // Database pass:
5. compute $\mathcal{T}_l(\mathcal{P}, \mathbf{r}, q) := \{\varphi \in \mathcal{C}_l \mid q(\varphi, \mathbf{r})\}$;
6. $l := l + 1$;
7. // Candidate generation:
8. compute $\mathcal{C}_l := \{\varphi \in \mathcal{P} \mid \mathit{level}(\varphi) = l$ and $\theta \in \mathcal{T}_{\mathit{level}(\theta)}(\mathcal{P}, \mathbf{r}, q)$ for all $\theta \in \mathcal{P}$ such that $\theta \prec \varphi\}$;
9. **for** all l **do** output $\mathcal{T}_l(\mathcal{P}, \mathbf{r}, q)$;

Theorem 7.7 Algorithm 7.6 works correctly. □