Knowl Inf Syst (2008) 16:29–51
DOI 10.1007/s10115-007-0121-3

REGULAR PAPER

Clustering multidimensional sequences in spatial and temporal databases

Ira Assent · Ralph Krieger · Boris Glavic · Thomas Seidl

Received: 24 May 2007 / Revised: 21 October 2007 / Accepted: 15 November 2007
Published online: 16 January 2008

Abstract Many environmental, scientific, technical or medical database applications require effective and efficient mining of time series, sequences or trajectories of measurements taken at different time points and positions forming large temporal or spatial databases. Particularly the analysis of concurrent and multidimensional sequences poses new challenges in finding clusters of arbitrary length and varying number of attributes. We present a novel algorithm capable of finding parallel clusters in different subspaces and demonstrate our results for temporal and spatial applications. Our analysis of structural quality parameters in rivers is successfully used by hydrologists to develop measures for river quality improvements.

Keywords Data mining · Clustering · Spatial and temporal data · Multidimensional sequences

1 Introduction

Environmental sensors produce data streams at successive time points which are often archived for further analysis. Applications like stock market analysis or weather stations gather ordered sequences of different values in large temporal databases. Weather stations for example use multiple sensors to measure, e.g., barometric pressure, temperature, humidity, rainfall. Many other scientific research fields like observatories and seismographic stations archive similar spatial or spatial-temporal sequences.

I. Assent (✉) · R. Krieger · B. Glavic · T. Seidl
Data Management and Exploration Group, RWTH Aachen University, Aachen, Germany
e-mail: assent@cs.rwth-aachen.de

R. Krieger
e-mail: krieger@cs.rwth-aachen.de

B. Glavic
e-mail: glavic@cs.rwth-aachen.de

T. Seidl
e-mail: seidl@cs.rwth-aachen.de

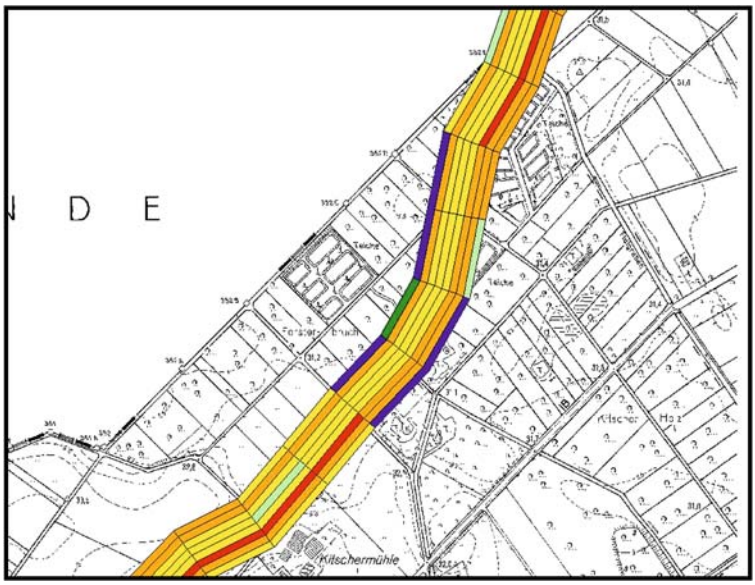


Fig. 1 GIS illustration of eight out of 19 river attributes. Each attribute is depicted as one of the parallel river sequences. Colors denote the categorical values in each attribute in the respective subsequence, from blue for "one" (best quality) to red for "seven" (worst quality)

As one application example, we focus on hydrological data. In a current project of the European Union on renaturation of rivers, the structural quality of river segments is analyzed. For a spatial database of German rivers, about 120.000 one-hundred-meter segments were evaluated according to 19 different structural criteria, e.g., quality of the riverbed [20]. They were mapped to quality categories, where a value of "one" indicates perfect quality, while a value of "seven" indicates most severe damages. Figure 1 illustrates 8 of the 19 attributes of a sample river segment in GIS (geographic information system) representation. The sequence order of the segments is given by the flowing direction of the rivers.

As the project aims at a quality improvement over the next decades, packages of measures have been suggested for different structural damages. They have been formalized in rules specifying the attribute value constraints and the attributes influenced positively by execution of the respective measure. An example constraint might be that a certain segment has good quality (categories one to three) riverbed and riverbanks and poor river bending (categories five to seven). This could be improved by measures like adding deadwood to positively influence river bending.

Finding and analyzing these patterns helps hydrologists summarize the overall state of rivers, give compact representations of typical situations and review the extent to which these situations are covered by measures envisioned. They can identify those patterns which are problematic, i.e., have low quality ratings, but are not yet covered by measures. In a follow-up step, these measures are annotated by time and cost information. This is used to generate an overview over the state of rivers as it might be in the near future if the measures are put into action.

From a computer science point of view, finding the intrinsic structure of these multidimensional sequences is a twofold task:

– detect frequent patterns within sequences for all possible subsequence lengths (note that we cannot know the pattern length a priori),
– then detect parallel occurrences of these patterns.

Patterns are ranges of values (which correspond to several categories of river quality structure) found in several (sub-)sequences. Pattern analysis has to take into account that the data is subjective and fuzzy, because structural quality of rivers was mapped by different individuals. Our approach is based on weighting by kernel densities in a density-based clustering approach. This effectively detects fuzzy sequence patterns. These patterns are clustered efficiently for arbitrary lengths using monotonicity properties. We transform these sequence pattern clusters into a cluster position space such that mining parallel patterns can be reduced to efficient FP-tree frequent itemset mining.

This paper is organized as follows: we review related work in Sect. 2. Basic definitions in Sect. 3 lay a sound foundation for the proposed clustering method in Sect. 4. We describe and disucss our algorithmic concept in Sect. 5. The experimental evaluation in Sect. 6 demonstrates the effectiveness of our approach on real world and synthetic datasets. Efficiency is shown and parametrization is evaluated. We conclude our paper in Sect. 7, summarizing our results and anticipating future work.

2 Related work

Numerous clustering methods have been proposed in the literature, including partitioning clustering, e.g., the well-known k-means algorithm [21]. These algorithms require the specification of the number of clusters to be found and can only detect convex cluster regions. Categorical clustering methods work well for categorical data where the notion of neighborhood is not meaningful [12,23].

Density-based algorithms use a function to determine the density of the neighborhood of each point and use a connectivity-notion to assign similar points to the same cluster [5,8,15]. Density-based clustering is robust to noise since it clusters only those points or sequences above some noise threshold as discussed in [2,7]. Moreover, it naturally incorporates neighboring objects into its cluster definition.

The analysis of sequence data has recently gained a lot of attention. Recordings of data at successive time points have been studied in time series mining; e.g. [9,18]. Most of these approaches, however, aim at finding patterns of values which do not have to directly follow one another, but may have other values in between, as in sequential frequent itemset mining [1,3].

Motif mining searches those patterns which have the highest count of similar subsequences [6,22]. Matching within some range is used to determine the frequency. However, neighbors are not weighted and the range is fixed for all sequences. Moreover, parallel patterns are not discussed since the application targeted is one-dimensional time series. While noise is removed in motif discovery as well, we found density-based clustering to be more useful in handling fuzzy data, because density-based clusters automatically adapt to different ranges of fuzziness.

3 Cluster model

In this section we formalize our notion of patterns in subsequences. We define a suitable cluster notion which reflects that our database consists of ordinal-valued sequences with some degree of noise.

Density-based clustering, which tries to separate dense areas (clusters) from sparse ones (noise) reflects the requirements of applications such as the river data scenario in that arbitrary cluster shapes may be found, the number of clusters does not need to be fixed a priori and noise is labeled as such, i.e., it does not have to be assigned to any cluster [8,16].

3.1 Subsequence patterns and clusters in one attribute

As noted before, it is crucial to determine subsequence clusters of arbitrary sequence lengths. We define sequence patterns of arbitrary length in single attributes and subsequently model parallelism between these clusters.

Definition 3.1 Sequence pattern:

- A tupel $S = (s_1, \ldots, s_k)$ of k subsequent values at positions 1 through k is called a **sequence** of length k in one attribute.
- A database **DB** is a set of sequences $\{S^1, \ldots, S^z\}$.
- We denote a **subsequence** of S from position i to j by $S[i, j] = (s_i, \ldots, s_j)$.
- Whenever we are not interested in the concrete positions of a sequence, but merely in its values, we call this a **pattern** $\mathbf{P} = \langle p_1, \ldots, p_k \rangle$ in one attribute.
- A pattern $\mathbf{P}$ occurs in a database if there is a sequence $S \in \mathbf{DB}$ and a position $i \in \mathbb{N}$ with $\mathbf{P} = S[i, i + \mathrm{k} - 1]$.
- The **support** of a pattern $\mathbf{P}$ is the number of its occurrences in the sequences of the database **DB**: $sup(\mathbf{P}) = |\{i \in \mathbb{N} \text{ and } S \in \mathbf{DB}, \text{ where } \mathbf{P} = S[i, i + \mathrm{k} - 1]\}|$.

When searching for prevailing patterns, it is important to notice that merely counting of sequence patterns is not sufficient in many scenarios. It is crucial to account for two factors: first, mapping of river structures may be blurred by people's subjective decisions on unclear category boundaries. Second, measures and their constraints may be applicable over several categories and cannot always be fitted exactly to these categorical boundaries. Moreover, small deviations in few segments may be tolerable for the application of a certain measure if this results in longer river courses treatable by a single measure. Hydrologists are thus interested in including "similar" sequences in frequency notions.

Density-based clustering tries to separate dense areas (clusters) from sparse ones (noise). The density of a pattern is determined by evaluating its neighborhood according to some distance function.

Any L_p-Norm $\left(L_p(\mathbf{Q}, \mathbf{Q}') = \sqrt[p]{\sum_{i=1}^{k} (q_i - q_i')^p}\right)$ between patterns $\mathbf{Q}$ and $\mathbf{Q}'$ can be used, yet the Manhattan norm (L_1) has shown to work well in preliminary experiments. We use a weighting function to ensure that with greater distance to the pattern evaluated, the influence of neighbors decreases. Any series of monotonously decreasing values can be used as a weighting function, since distances between nominal sequences are always discrete. All kernel-estimators known from statistics [14] are constantly falling functions. Experiments have shown that weighting functions based on Gaussian kernels ($W_\sigma^{\mathbf{P}}(\mathbf{Q}) = \mathsf{exp}\ (-\mathsf{dist}(\mathbf{P}, \mathbf{Q})^2/2\sigma^2)$) perform well in many applications. Using weighting functions based on kernel estimators provides us with a natural way of defining the set of similar sequences to be included in the density evaluation. Whenever the weights assigned drop below a certain significance threshold τ, these sequences should not be considered in the density estimation. For example, Gaussian kernels assign (possibly very small) density values to all patterns in the database, which can be cut off below some tiny value, such as $\tau = 0.01$ or less. This way, excess density computations can be avoided.

Definition 3.2 Neighborhood and density:

- The τ-**neighborhood** of a pattern in one attribute $\mathbf{P}$, $N_\tau(\mathbf{P})$, is defined as the set of all patterns $\mathbf{Q}$ which at least have the influence τ on the pattern $\mathbf{P}$ evaluated: $N_\tau(\mathbf{P}) = \{\mathbf{Q}, \text{where } W_\sigma^{\mathbf{P}}(\mathbf{Q}) \geq \tau\}$.
- The **density** of $\mathbf{P}$ is the weighted sum of patterns in the neighborhood

$$\mathsf{density}(\mathbf{P}) = \sum_{\mathbf{Q} \in N_\tau(\mathbf{P})} W_\sigma^{\mathbf{P}}(\mathbf{Q}) * sup(\mathbf{Q})$$

- $\mathbf{P}$ is **dense** with respect to a density threshold δ iff $\mathsf{density}(\mathbf{P}) \geq \delta$.

We are now ready to formalize our cluster notion. As mentioned before, clusters should consist of similar, dense sequences of the same length. Sequences within one cluster should therefore be within certain parameterizable boundaries.

Definition 3.3 Cluster:
A set $\mathbf{C} = \{\mathbf{P}_1, \ldots, \mathbf{P}_m\}$ of m patterns $\mathbf{P}_i$ is a cluster of length k with respect to a density threshold δ and a compactness parameter γ iff:

- for all patterns $\mathbf{Q}$ of length k not in $\mathbf{C}$: $\mathbf{C} \cup \{\mathbf{Q}\}$ is not a cluster (***Maximality***)
- for all patterns $\mathbf{P}_i$: $\mathsf{density}(\mathbf{P}_i) \geq \delta$ (***Density***)
- for any patterns $\mathbf{P}_i, \mathbf{P}_j \in \mathbf{C}$ there is a chain of patterns $(\mathbf{Q}_1, \ldots, \mathbf{Q}_v) \in \mathbf{C}$ such that $\mathbf{Q}_1 = P_i, \mathbf{Q}_v = P_j$ and $\forall\, r\ \mathsf{dist}(\mathbf{Q}_r, \mathbf{Q}_{r+1}) \leq \gamma$ (***Compactness***)

Put informally, we are thus looking for clusters which are as large as possible (maximality), whose elements are all dense (density) and at most γ apart from each other (compactness). We set γ to one in categorical settings.

Example Figure 2 illustrates the definition of density and gives an example for a density calculation of pattern $\mathbf{P} = \langle 1, 2\rangle$. The upper part visualizes two sequences S and T with two exemplary attributes, the riverbed and the bank. In the lower part of the figure the density-value for a pattern $\langle 1, 2\rangle$ is calculated. We assume a significance threshold of $\tau = 0.2$. The Gaussian weighting function drops below $\tau = 0.2$ for sequences having a distance higher than 1.8 from the point evaluated. Thus in our ordinal setting only sequences with a distances of or less 1 have significant influence: $\mathsf{exp}(-1^2/2) \approx 0.6 > \tau$ and $\mathsf{exp}(-2^2/2) \approx 0.13 < \tau$. In our example the τ-neighborhood of pattern $\langle 1, 2\rangle$ contains the sequence $\langle 1, 2\rangle$ itself starting at two positions S_1 and S_6 (distance zero) and $\langle 1, 3\rangle$ starting at four positions S_3, S_8 and T_2, T_5 (distance one). Thus the density-value for $\langle 1, 2\rangle$ is $2 * W_\sigma^{\langle 1,2\rangle}(\langle 1, 2\rangle) + 4 * W_\sigma^{\langle 1,2\rangle}(\langle 1, 3\rangle) = 2 * 1 + 4 * 0.6 = 4.4$. For a density-threshold of e.g. $\delta = 3$ the pattern $\langle 1, 2\rangle$ is considered dense.

3.2 Multiclusters in parallel patterns

Single sequence patterns give insight into the inherent structure in individual attributes. In multidimensional sequence databases these patterns have to be extended to parallel patterns. River measures may affect several structural properties, e.g. the river bank on the left and the right as well as the river bending. Similarly, constraints are often formulated for several attributes as well. Likewise, in other applications, situations which require specific measures are typically described via several sensor values.

We define multiclusters as frequent parallel occurrences of single attribute clusters. Frequency is measured in terms of simultaneous occurrences, i.e. the number of positions in

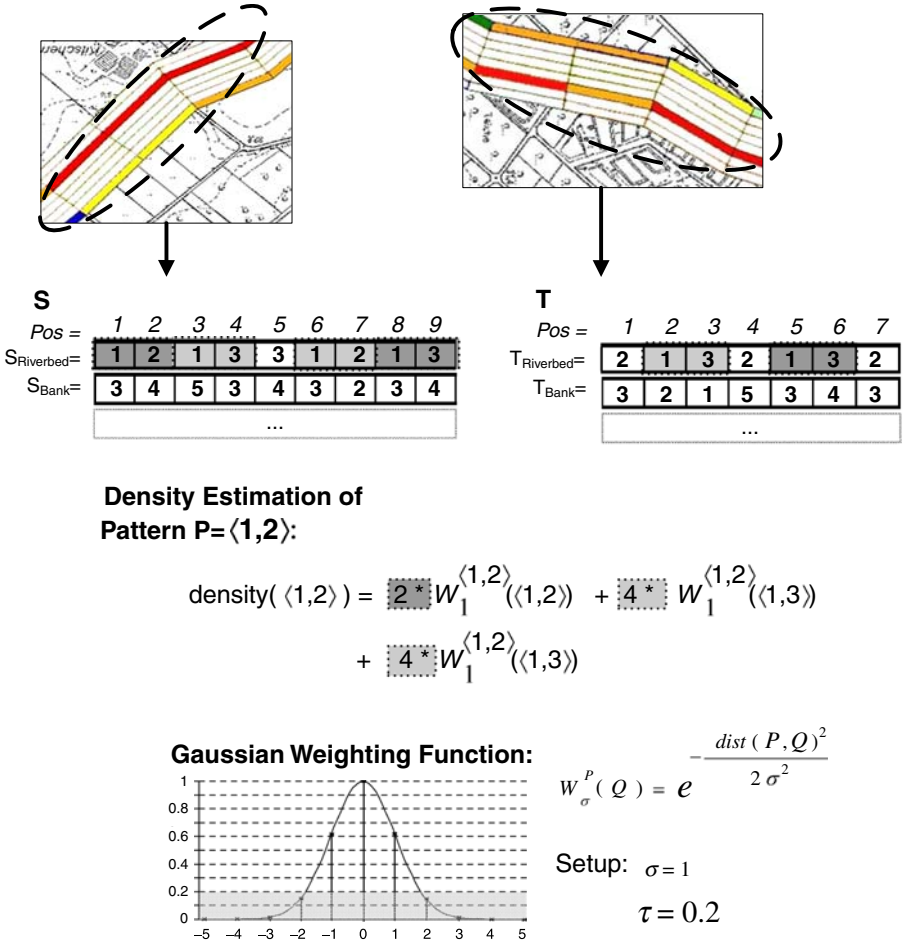


Fig. 2 Example multicluster in the river database

any sequence, where clusters are detected in different attributes. We formalize our notion of parallel clusters as follows:

Definition 3.4 Multicluster:

A set of sequence clusters $\mathbf{C_1}, \ldots, \mathbf{C_n}$ of length k from n different attributes is a multicluster **MC** iff:

- $\mathbf{C_1}, \ldots, \mathbf{C_n}$ are **parallel** at some position i,
  i.e., $\forall j \in \{1, \ldots, n\}$ a pattern $P_j \in \mathbf{C_j}$ occurs at position i
- $\mathbf{C_1}, \ldots, \mathbf{C_n}$ occur **frequently** together,
  i.e., $|\{\text{i} \in \mathbb{N},\ \mathbf{C_1}, \ldots, \mathbf{C_n}$ are parallel at position i$\}| \geq \phi$.

Put informally, we are thus looking for those positions which show a pattern contained in each of the clusters (Parallelism), which have a count equal to or greater than the threshold given (Frequency).

The frequency threshold ϕ reflects the number of positions where the multicluster is detected. It can be set in relation to the database size, i.e., the overall number of multidimensional

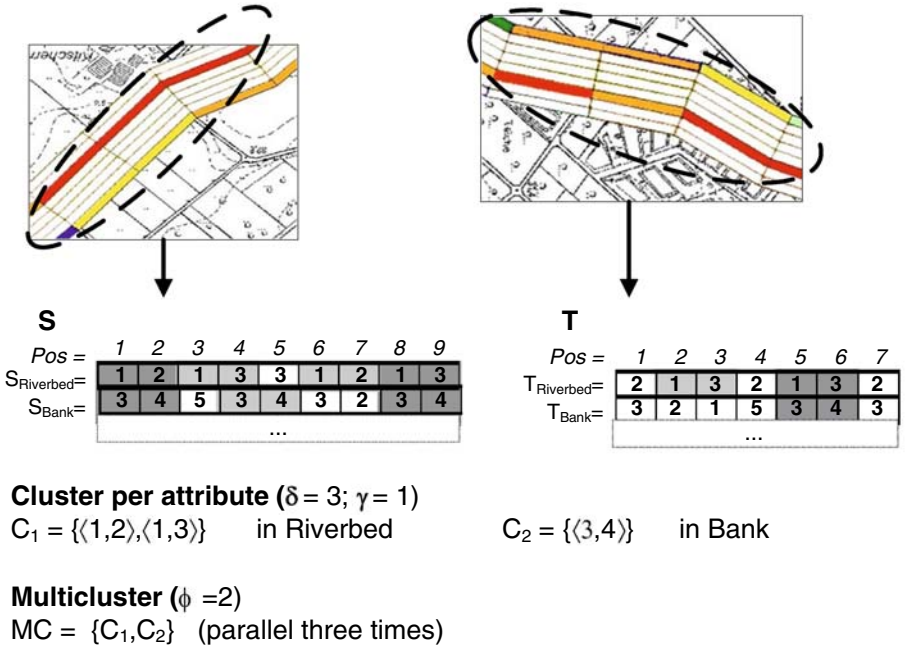


Fig. 3 Example of a multicluster

sequence segments (see Sect. 6). Hence, ϕ corresponds to the relative frequency of a multicluster (the support) and is a key parameter depending on the application. In general, increasing ϕ decreases the number of identified multiclusters, as more occurrences of the pattern are required. Multiclusters detected using a higher ϕ are far more typical for the data set. Our experiments suggest that an initial setting of about 1% of the data set serves as a good starting point for analysis. As more or less patterns are desired, the threshold is adapted accordingly.

Example In Fig. 3 we present a multicluster in two attributes. The multicluster MC consists of two clusters $C_1 = \{\langle 1, 2\rangle, \langle 1, 3\rangle\}$ in the riverbed attribute and $C_2 = \{\langle 3, 4\rangle\}$ in the bank attribute. They occur in positions S_1, S_8, T_5, thus three times. Assuming a frequency threshold of $\phi = 2$, MC is a multicluster.

4 Efficient cluster mining

To detect clusters of arbitrary length, a naive approach might be to simply re-run a density-based clustering algorithm for each length value to detect all possible clusterings. Obviously, this leaves room for efficiency improvement.

4.1 Monotonicity

We avoid excess clustering steps by exploiting monotonicity properties of clusters. We show that patterns which are not dense, cannot be part of longer dense patterns. We may safely prune them from consideration in longer pattern clustering.

We formalize this monotonicity first for patterns and then prove that this property holds for clusters themselves.

Theorem 4.1 ***Density monotonicity****: For any two patterns* $\mathbf{P}, \mathbf{Q}$ *of length* k *and their respective prefix/suffix* $\mathbf{P}', \mathbf{Q}'$ *of length* $k-1$ *holds:*

(1) $\mathbf{Q} \in N_\tau(\mathbf{P}) \Rightarrow \mathbf{Q}' \in N_\tau(\mathbf{P}')$
(2) $\mathsf{density}(\mathbf{P}) \leq \mathsf{density}(\mathbf{P}')$

For (1) we note that an L_p norm, $p \geq 1$ is the p-root of sum of absolute differences in sequence values. This means that a reduced sum of $k-1$ of these differences is necessarily smaller than or at most equal to a sum of all k differences.

Proof For a prefix/suffix $\mathbf{Q}', \mathbf{P}'$ of $\mathbf{Q}, \mathbf{P}$ we first show: $\mathsf{dist}(\mathbf{P}, \mathbf{Q}) \geq \mathsf{dist}(\mathbf{P}', \mathbf{Q}')$.
Dropping the first or last summand in our distance sum for $\mathbf{Q}, \mathbf{P}$, we obtain for the prefixes or suffixes $\mathbf{Q}, \mathbf{P}$:

$$\sqrt[p]{\sum_{i=1}^{k} |p_i - q_i|^p} \geq \sqrt[p]{\sum_{i=1}^{k-1} |p_i - q_i|^p} \quad \text{dropping the last summand: prefix}$$

$$\sqrt[p]{\sum_{i=1}^{k} |p_i - q_i|^p} \geq \sqrt[p]{\sum_{i=2}^{k} |p_i - q_i|^p} \quad \text{dropping the first summand: suffix}$$

$$\Rightarrow \mathsf{dist}(\mathbf{P}, \mathbf{Q}) \geq \mathsf{dist}(\mathbf{P}', \mathbf{Q}')$$

Thus the distance between two patterns is greater than the distance between the respective prefix or suffix. Using this fact we can prove (1):

$$\begin{aligned}
&\mathsf{dist}(\mathbf{P}, \mathbf{Q}) \geq \mathsf{dist}(\mathbf{P}', \mathbf{Q}') \\
&\Rightarrow W_\sigma^{\mathbf{P}}(\mathbf{Q}) \leq W_\sigma^{\mathbf{P}'}(\mathbf{Q}') \\
&\quad \text{(weighting functions are increasing with decreasing distance)} \\
&\Rightarrow (W_\sigma^{\mathbf{P}}(\mathbf{Q}) \geq \tau \Rightarrow W_\sigma^{\mathbf{P}'}(\mathbf{Q}') \geq \tau) \\
&\quad \text{(using Definition of } N_\tau) \\
&\Rightarrow (\mathbf{Q} \in N_\tau(\mathbf{P}) \Rightarrow \mathbf{Q}' \in N_\tau(\mathbf{P}')).
\end{aligned}$$

Part (2) is proven using a similar argument: the density is defined as a weighted sum of the support of patterns. Their shorter counterparts, prefix or suffix, are assigned smaller distance values (part 1) and therefore larger weights.

$$\begin{aligned}
\mathsf{density}(\mathbf{P}) &= \sum_{\mathbf{Q} \in N_\tau(\mathbf{P})} W_\sigma^{\mathbf{P}}(\mathbf{Q}) * \sup(\mathbf{Q}) \text{ (Definition of density)} \\
&\leq \sum_{\mathbf{Q}' \in N_\tau(\mathbf{P}')} W_\sigma^{\mathbf{P}}(\mathbf{Q}) * \sup(\mathbf{Q}') \\
&\quad \text{(Part 1: } \sup(\mathbf{Q}) \leq \sup(\mathbf{Q}') \text{ as a subsequence for } \mathbf{Q} \text{ also matches } \mathbf{Q}') \\
&\leq \sum_{\mathbf{Q}' \in N_\tau(\mathbf{P}')} W_\sigma^{\mathbf{P}'}(\mathbf{Q}') * \sup(\mathbf{Q}')
\end{aligned}$$

(Part 1: distance of shorter patterns is smaller, weights are larger)

$= \mathsf{density}(\mathbf{P}')$ (Definition of density)

□

Since density and neighborship are monotonous, we can conclude that clusters are monotonous as well:

Theorem 4.2 ***Cluster monotonicity****: For any cluster* $\mathbf{C}$ *of length* k*, there is a cluster* $\mathbf{C}'$ *of length* $k-1$ *such that for any pattern* $\mathbf{P}$ *and its prefix/suffix* $\mathbf{P}'$ *holds:*

$$\mathbf{P} \in \mathbf{C} \Rightarrow \mathbf{P}' \in \mathbf{C}'$$

Proof For any $\mathbf{P}$ in $\mathbf{C}$, we have that $\mathbf{P}'$ is dense (Lemma 4.1), and since the distance of shorter patterns is smaller (Proof of Lemma 4.1), there exists a chain of prefix/suffix patterns which guarantees compactness. This means that all prefixes/suffixes are part of a cluster (which might contain additional patterns due to the third requirement, completeness). □

Summing up, we know that any pattern or cluster which does not satisfy the density criterion, can be safely pruned from the search for longer patterns or clusters.

4.2 Indexing subsequence patterns and clusters

In order to efficiently calculate the density value for a pattern it is crucial to quickly retrieve the neighborhood of a subsequence. We introduce an index structure for patterns which supports neighborhood queries and density estimators.

Since existing index structures do not work for patterns of different length, the hierarchical index structure illustrated in Fig. 4 was developed. The index structure is tailored to the MC algorithm in that the bottom-up approach is supported: queried patterns always grow in length.

The index is constructed by scanning once over each sequence. Each pattern of a fixed starting length determined is added to the index structure. A pattern is represented by a path of labeled nodes from the root to a leaf. To later determine the density value of a pattern the support is annotated at each corresponding leaf node. Further on the index structure stores the position list (all starting points) for each pattern. For scalability purposes the position list is stored on hard disk. In addition, a cluster identifier (e.g., $C1$) or the flag *unclassified* (UC) is stored at each leaf node (Fig. 4). The MC algorithm uses this flag to efficiently calculate the transitive closure (compactness in Definition 3.3) for a dense pattern. The density value for a pattern can be calculated by summing up and weighting the support of all patterns in the corresponding neighborhood. The index structure efficiently supports neighborhood queries by selecting the appropriate node ranges while descending the tree.

Example In Fig. 4 we assume a parameter setting of $\tau = 0.2$ with a weight of 1 for patterns having distance zero and a weight of 0.3 for patterns having distance one. The density value for pattern $\langle 1, 3\rangle$ can then be calculated in the following way: Starting at the root node the node range ("1"−"2") must be considered. When processing node "2", for instance, all patterns containing this node have a distance of at least one (distance from $\langle 1, 3\rangle$ to $\langle 2, *\rangle$ is greater than or equal to one). Since the maximal distance according to τ is bounded by 1, the only pattern which must be considered below node "2" is the pattern $\langle 2, 3\rangle$. In the same way all other patterns within the specified range can be determined (in this example $\langle 1, 2\rangle$, $\langle 1, 3\rangle$

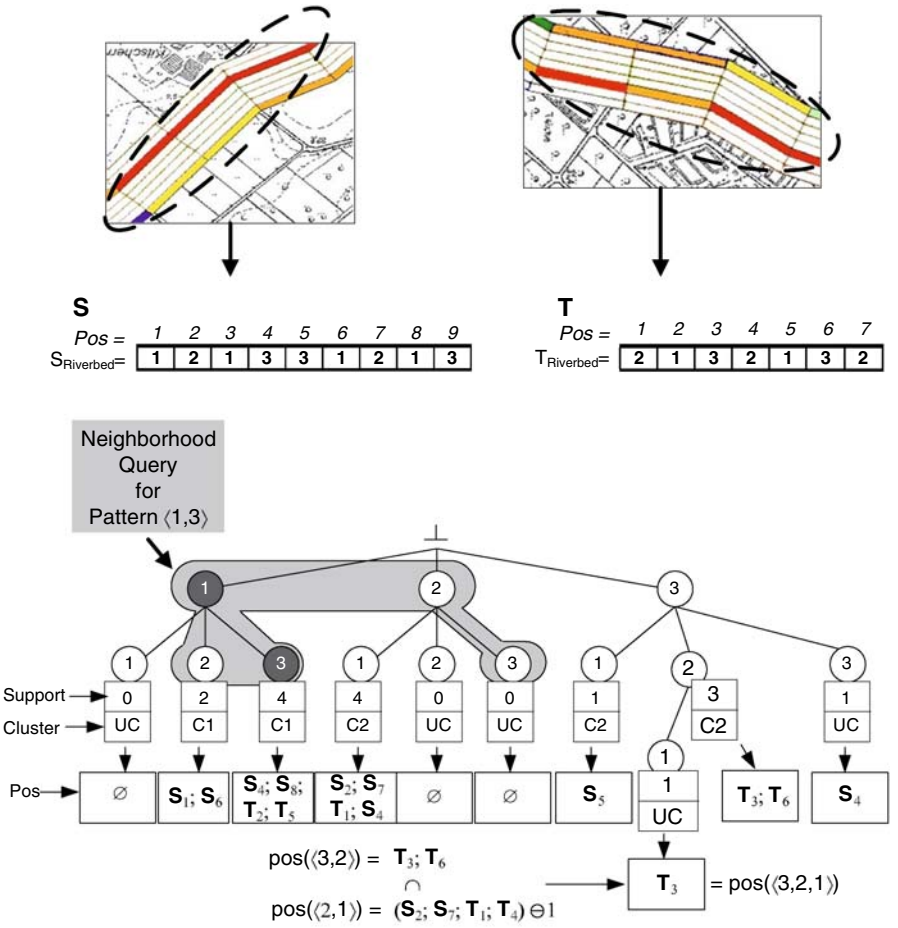


Fig. 4 Hierarchical index structure

and $\langle 2, 3\rangle$). Finally the density value for the pattern $\langle 1, 3\rangle$ can be calculated by summing up the support of the leafs for each weighted subsequence:

$$\begin{aligned}\text{density}(\langle 1,3\rangle) &= W^{\langle 1,3\rangle}(\langle 1,2\rangle) \times \sup(\langle 1,2\rangle) + W^{\langle 1,3\rangle}(\langle 1,3\rangle) \times \sup(\langle 1,3\rangle) \\ &\quad + W^{\langle 1,3\rangle}(\langle 2,3\rangle) \times \sup(\langle 2,3\rangle) \\ &= 0.3 \times 2 + 1.0 \times 4 + 0.3 \times 0 = 4.6.\end{aligned}$$

4.3 Clusters of arbitrary length

To identify clusters of arbitrary length the MC algorithm works bottom-up by first identifying short clusters and then successively searching for longer clusters. This section proposes a method to generate new longer cluster candidates by combining appropriate shorter clusters.

The MC algorithm elongates the investigated patterns by one in each step. Thus the index structure must be able to calculate the position list and support for constantly growing patterns.

This is achieved by the algorithm: the position list for a pattern **P** of length k can be calculated by using its $k-1$ prefix and suffix. Intuitively, the pattern **P** can only occur in a sequence if its prefix starts at exactly its starting position and its suffix ends where **P** ends. This means that no extra database scans are necessary to determine its occurrence—we simply take a look at all the starting positions of its prefix and determine its intersection with its suffix shifted by one. Since both are shorter by one, they must have been processed earlier.

Example Figure 4 illustrates this algorithm for the pattern $\mathbf{P} = \langle 3, 2, 1\rangle$. Its prefix is the pattern $\langle 3, 2\rangle$, its suffix $\langle 2, 1\rangle$. We can compute the positions of pattern **P** simply by shifting the position list of the suffix one back and intersecting it with the position list of its prefix. The prefix $\langle 3, 2\rangle$ occurs in positions T_3, T_6, while its suffix $\langle 2, 1\rangle$ is found at starting positions S_2, S_7, T_1, T_4. Any sequence elongated by one which contains this suffix has to start one position earlier to end with this suffix. Any occurrence of $\langle 3, 2, 1\rangle$ will thus start at $\{T_3, T_6\} \cap \{S_1, S_6, T_0, T_3\} = \{T_3\}$. And indeed $\langle 3, 2, 1\rangle$ is found at this position as we can verify in the illustration of **T**.

As we can see, patterns are efficiently extended on the fly when a longer pattern is accessed for the first time.

5 Multicluster algorithm

Our Multicluster algorithm exploits both monotonicity properties and density computation on the index. Working in two steps, each monotonicity property on subsequence patterns and clusters are used. The first step searches for clusters of arbitrary length while the second step combines parallel clusters.

5.1 Step one: subsequence clustering

The MC algorithm is presented in Fig. 5. First, the index structure is created using a parameter $length_{start}$ (can be set to one to mine all patterns where desired). After the construction step the index structure contains one entry for each pattern of length $length_{start}$. Next the first cluster candidate is generated. The first candidate contains all patterns, since all of them might be dense and hence could belong to a cluster. A depth first search on the index structure efficiently retrieves all different patterns stored in the database (method *queryAllEntries*).

Next the clustering loop starts which discovers all clusters from $length_{start}$ to $length_{max}$ (may be set to *infinity*). For every length all patterns of all cluster candidates are tested if they satisfy the density property. Each unclassified dense pattern is then expanded to a cluster by the method *ExpandToCluster*. This method creates a new cluster and assigns each dense pattern within the transitive closure (w.r.t. γ) to this new cluster.

Neighborhood queries are necessary to determine the density value (method *isDense*) and the transitive closure of a subsequence (method *ExpandToCluster*). Since all patterns of shorter length are no longer necessary after each clustering step the position lists and flags stored for these patterns can then be released (method *prune*).

After a set of clusters is discovered for a specific length the cluster set is stored in the result set $clustSetResult$ and new cluster candidates ($candidateSet$) are determined by using the old cluster set ($clustSet$). Figure 4 also illustrates the generation of a cluster candidate. This step utilizes the monotonicity property of subsequence patterns in Lemma 4.2: only subsequence patterns whose prefix and suffix are member of an already discovered cluster (and hence are dense) must be considered as members of a new cluster candidate.

```
ClusterSet MC-Clustering(Database db, int length_start, int length_max)
  Index index(db);
  index.initialCreate(length_start);          // create the index using all length_start sequences
  ClusterSet candidateSet    := { index.queryAllEntries() }; // all sequences are candidates
  ClusterSet clustSetResult  := ∅;           // result set of subsequence clusters
  foreach length from length_start to length_max do
      ClusterSet clustSet  := ∅;             // store new cluster sets

      /* Generate new clusters based on each candidate cluster */
      foreach clustCand in candidateSet do
          markUnclassified(clustCand);       // marks all sequences unclassified
          foreach seq in clustCand do

              /* Check if sequence is dense and expand to Cluster */
              if isUnclassified(seq) and isDense(seq, index) then
                   clustSet := clustSet ∪ ExpandToCluster(seq, clustCand, index);
              end if;

          end foreach;
      end foreach;

      index.prune(length); // discard unnecessary paths and position lists

      clustSetResult := clustSetResult ∪ clustSet; // store cluster

      /* Create new candidate clusters using monotonicity property */
      candidateSet := CreateCandidateCluster(clustSet, length+1, index);
  end foreach;

  return clustSetResult;
```

Fig. 5 Multicluster algorithm

```
ClusterSet CreateCandidateCluster(ClusterSet clustSet, int length, Index index)
  ClusterSet resultSet := ∅;
  foreach clust in clustSet do
    foreach seq in clust do

      /* query all sequences from index which start with Suffix(seq) */
      SequenceSet extSeqSet := index.prefixQuery( Suffix(seq) );
      ClusterSet  clustCand := ∅;

      foreach extSeq in extSeqSet do

        /* if extension is dense then put the extended sequence into the result set */
        if isDense(extSeq, index)) then
           clustCand := clustCand ∪ { seq[1]•extSeq };
        end if;
      end foreach;
      resultSet := resultSet ∪ clustCand;
  end foreach;

  return resultSet ;
```

Fig. 6 Creating Candidate Clusters

The algorithm to calculate the corresponding cluster candidates is based on the discovered clusters of the previous step (Fig. 6). The method *CreateCandidateCluster* loops over all patterns of all clusters and tries to extend each pattern. For this purpose the suffix of $length-1$ is extracted from each pattern and all patterns which start with the extracted suffix are queried from the index (*prefixQuery*). Each queried pattern which satisfies the density property is then used to create an elongated pattern by concatenating the appropriate prefix and suffix.

These queries are also supported by the proposed index structure. The elongated patterns are finally assigned to a new cluster candidate.

Example Consider a cluster containing only one pattern $\mathbf{P} = \langle 5, 1, 3, 7\rangle$. This cluster of length 4 is extended to a cluster candidate of length 5 in the following way: First a query using the suffix of $\mathbf{P}$, $\langle 1, 3, 7\rangle$, is performed. We assume that the intersection with all possible patterns of length 4 results in a set $\{\langle 1, 3, 7, 3\rangle, \langle 1, 3, 7, 9\rangle\}$. Next the patterns are checked whether they are dense or not. This can be achieved by simply evaluating the annotated cluster flag (a dense pattern must belong to a cluster). Let's assume $\langle 1, 3, 7, 9\rangle$ is dense and $\langle 1, 3, 7, 3\rangle$ is not dense. In this case the two patterns $\langle 5, 1, 3, 7\rangle$ and $\langle 1, 3, 7, 9\rangle$ are used to create the elongated pattern $\langle 5, 1, 3, 7, 9\rangle$ of length 5. This pattern is assigned to a new cluster candidate and returned to the MC clustering algorithm.

5.2 Step two: multiclustering

Having discovered dense patterns and combined them to clusters, we identify those clusters which are parallel in the dataset (see Definition 3.4). Since for any cluster of length k all corresponding clusters of length $k-1$ have previously been detected, we use the monotonicity property presented in Lemma 4.2.

An important property of multiclusters is that a subsequence of a specific length may belong to no more than one cluster. Hence any position in a sequence is the starting point for at most one density based cluster of a fixed length. This property is used to transform the sequence database from a value representation to a cluster representation. After this transformation it is possible to discover multiclusters by efficient frequent itemset mining techniques.

We use the following representation to apply the frequent itemset mining: clusters starting at the same position in different attributes are combined to one itemset. The clusters are identified by their cluster-ids. A frequent itemset contains often occurring multiclusters in which each cluster belongs to a different attribute.

Example Figure 7 illustrates the transformation process. In the upper part, three attributes of sequence $\mathbf{S}$ are shown, with clusters highlighted in gray colors. Assumed are the following sequence clusters: $C_1 = \{\langle 1, 2\rangle, \langle 1, 3\rangle\}$ in S_1, $C_2 = \{\langle 3, 4\rangle\}$ in S_2, $C_3 = \{\langle 2, 3\rangle\}$ in S_3. They are transformed according to their positions as follows: Position 1 in the sequence $\mathbf{S}$ would yield a transaction $\{1, 2, 3\}$ since all three clusters are parallel. Position two has no starting clusters, position three only cluster C_1 and so on (lower part of Fig. 7). We can therefore immediately derive the frequency of itemsets from the cluster information: the itemset $\{C_1\}$ occurs six times, $\{C_2\}$ four times, ..., and finally $\{C_1, C_2, C_3\}$ two times.

We use the Frequent Pattern (FP) growth algorithm proposed by Han et al. [13] for the extraction of frequent itemsets. The tree representation is very suitable for our task since database scans are avoided. Recall that the FP-tree builds a path annotated by support values for all frequent items. To obtain frequent itemsets conditional FP-trees are constructed iteratively for each frequent item. As mentioned above we use the following representation to apply the FP-tree: one dimensional subsequence clusters starting at a certain position are itemsets containing cluster-ids (see Fig. 7). The FP-tree supports efficiently determining frequent patterns w.r.t. ϕ. A frequent itemset contains often occurring correlated clusters in which each subsequence cluster belongs to a different dimension. Thus the result of the FP-tree algorithm contains multiclusters as described in Definition 3.4. In order to find multiclusters of any length the FP-tree algorithm is started for all different lengths for which clusters have been found. Since the FP-tree works extremely fast, clustering arbitrary lengths is efficient.

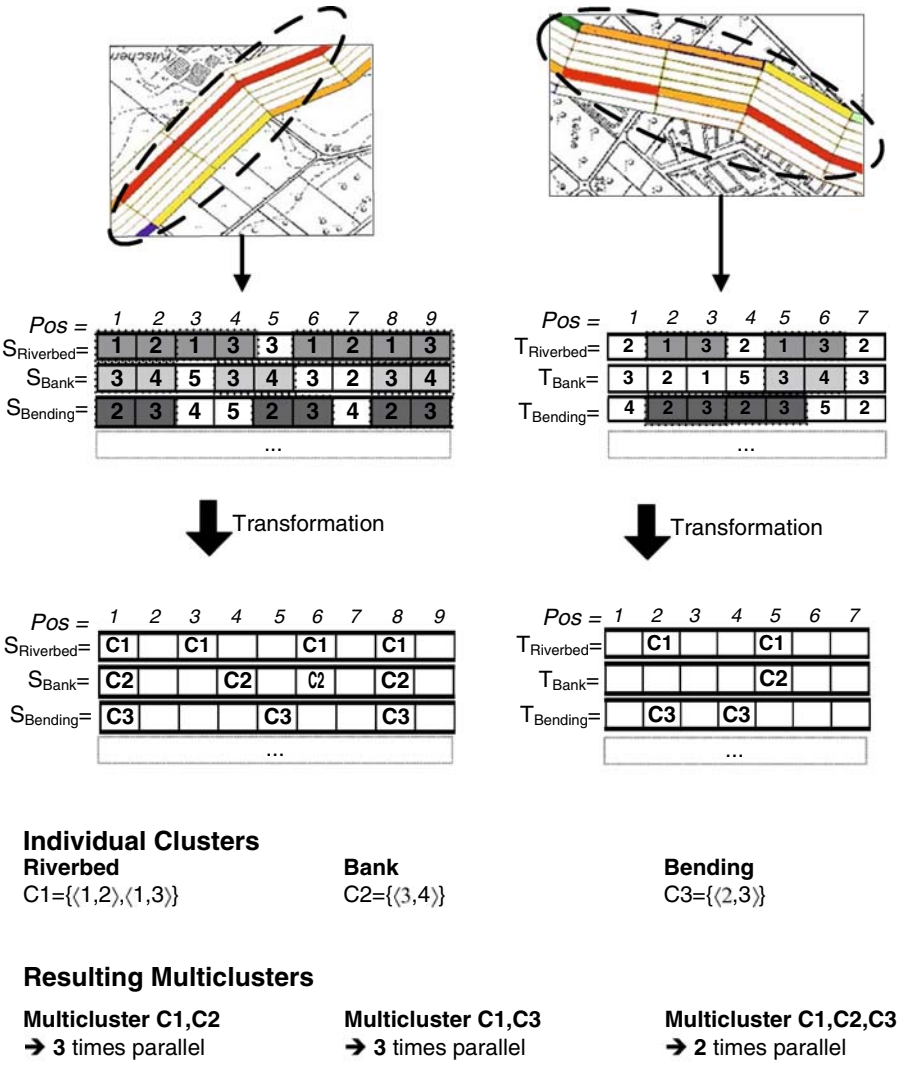


Fig. 7 Transformation example

5.3 Analytical evaluation

Subspace clustering is a highly complex task, as the number of subspaces is exponential in the number of attributes. Further on, detecting multiclusters of arbitrary length is quadratic in the sequence length. Consequently, multicluster detection is a challenging problem. To ensure efficiency, our algorithmic approach therefore bears on each of these aspects.

First, concerning arbitrary lengths of possible patterns, our index structure avoids re-building potentially frequent patterns from scratch. Only frequent patterns, not the actual sequences nor the infrequent patterns, are actually processed when elongating clusters. For each of these patterns, compact representations are stored and position lists are kept

on hard disk to reduce main memory usage. Second, only subspaces which contain clusters in single attributes are mined for multiclusters. This greatly reduces the number of potential combinations. Moreover, by mapping sequences to cluster ids, the FP-growth algorithm allows for fast detection of multiclusters. Memory requirements could be further reduced by applying a secondary storage extension of the FP-tree method as suggested e.g. in [11].

We validate this analysis in the experiments which show that our algorithm scales almost linearly in terms of both length and numbers of attributes on different data sets.

6 Experimental evaluation

We ran experiments on synthetic and real world data to evaluate the quality and performance. Synthetic data is used to evaluate different cluster parameters and to develop guidelines and heuristics for supporting users in setting up parameters. It also evidences the algorithm's scalability as well as its quality in detecting all generated clusters. Real world data demonstrates the usefulness of the results for domain experts. Our implementation is in Java, and experiments were run on Pentium 4 machines with 2.4 Ghz and 1 GB main memory.

6.1 Parameter setup

Several parameters can be used to tune our algorithm to domain specific needs. We therefore develop appropriate guidelines and heuristics for choosing reasonable parameter settings. To study the effect of different parameter setups on the clustering result we generated different synthetic data sets. For each data set we varied the number of sequence clusters and multiclusters contained in the data.

As suggested in Sect. 3 we use a Gaussian kernel as a weighting function. Thus the first parameter to set is the value for σ. We propose using a density plot to determine the effect of different σ values on the density for length two patterns. The data set used to demonstrate this heuristic is eight dimensional and contains four multiclusters. Figure 8 illustrates the density plot for one attribute of the synthetic dataset using $\sigma = 0.75$, 0.9, and 1.0, respectively. Four separate clusters can be seen, which mainly consist of patterns of similar values (on the diagonal). Decreasing the value of σ corresponds to splitting the rightmost cluster around coordinates (16,16) into two separate clusters (leftmost illustration). As this would result in two clusters which would be a lot less frequent and distinct from the remaining three clusters, splitting is considered inappropriate and a higher value for σ should be chosen. On the other hand, further increasing the value of σ corresponds to merging the two clusters at the center

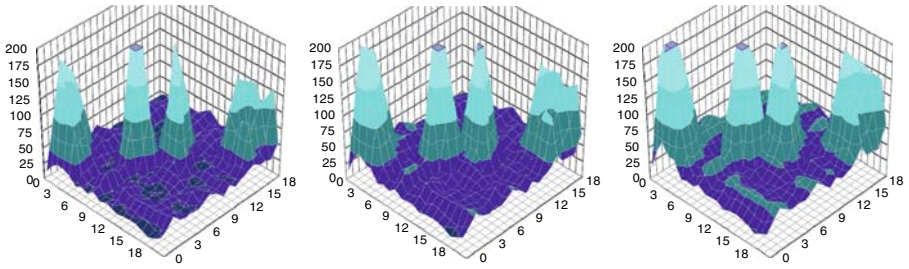


Fig. 8 Influence of σ on density ($\sigma = 0.75, 0.9, 1.0$ from left to right)

Param.	Usage	Setup	Exp. 1
δ	**Density Threshold** Separates noise from clusters.	see left	10
σ	**Expected Blur (Fuzziness)** Standard deviation in Gaussian Kernel. Value ~1 for categorical.	0.75-1.25	0.9
τ	**Significance Threshold** Cuts off insignificant weights. For very small value almost no error.	<<1	0.005
γ	**Compactness Parameter** Max. distance between patterns in cluster; ~1 for categorical.	1	1
ϕ	**Co-occurring Frequency** Correlates to data coverage of multicluster.	1% of the dataset	30

Fig. 9 Parameter guidelines

into a single cluster (rightmost illustration). This would create a larger and more frequent cluster than the remaining two. Hence, a lower value for σ is needed to separate these clusters.

These plots, as well as further experiments, suggests a choice for σ of about one. This allows separation of clusters as well as aggregation of similar values. A choice of one for σ also reflects the fact that a deviation in one value is generally deemed tolerable in many ordinal settings. An appropriate minimum density value can also be derived from the density plot. The ordinate value which separates the clusters from the "noise floor" can be visually determined and extrapolated to longer patterns. This leads to appropriate values for the density threshold δ (e.g., for this dataset $\delta = 10$). Note that the density plot can be created easily during the initialization of the index structure.

The remaining parameters can be determined in a straightforward manner. τ is set to very small values (0.01 or less) to cut off insignificant density values. Similar patterns should be clustered together which leads to a compactness threshold of one ($\gamma = 1$) in ordinal settings like the river dataset. Finally, experts are only interested in multiclusters which cover a significant part of the dataset (e.g., one percent minimum frequency). Figure 9 summarizes our heuristics and parameter settings for the experiments.

Using this parameter setting the MC algorithm indeed finds all multiclusters hidden in the synthetic dataset. This first experiment demonstrates the effectiveness of the MC algorithm and indicates that the developed heuristics help users find reasonable parameters.

6.2 Synthetic data

As mentioned in Sect. 4.1, a naive approach for detecting clusters of arbitrary length would be to re-run a density-based subspace clustering algorithm for all possible lengths. We compare the performance and the results of the MC algorithm with SUBCLU [16], an extension of the density-based DBSCAN [8] algorithm to subspace clustering. Since SUBCLU was not developed to cluster subsequences we had to extend the original implementation by the density notion presented in Sect. 3.

To evaluate the scalability of our MC algorithm in comparison with existing approaches, we varied the length of the data sequences and the number of sequences contained in the data base. Additionally we investigated the influence of the number of different labels per sequence domain on the performance of the MC algorithm. For this purpose we implemented a data

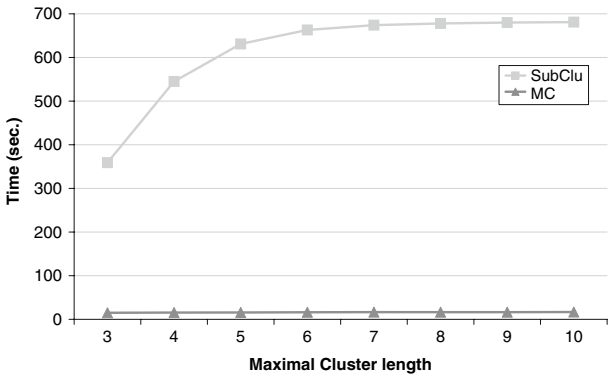


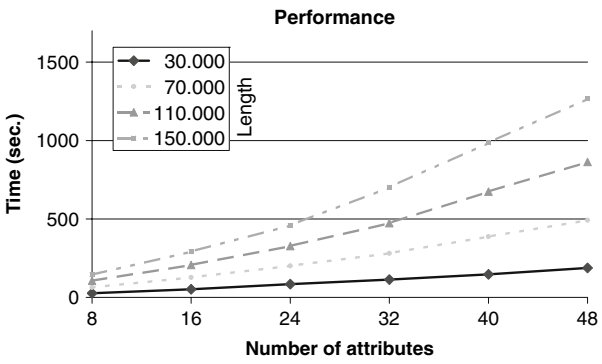
Fig. 10 Comparison with SUBCLU

generator which creates density connected clusters in each sequence. The data generator gets the number of clusters, and the size and length of the data base as input parameters. To parameterize the position and size of a cluster, a starting sequence and the number of pattern belonging to the cluster is specified for each cluster. To generate multiclusters some of these clusters are correlated to parallel multiclusters. All values not belonging to a sequence cluster are uniformly distributed based on the size of the corresponding domain to generate noise.

In our first experiment we generated a data set consisting of eight attributes with twenty different labels. We hid three to six clusters of length five to ten in each sequence. Each hidden multicluster has a minimum frequency of one percent and consists of three to six patterns. Figure 10 illustrates the runtime of both algorithms. Note that MC is faster by an order of magnitude. The runtime of both algorithms depends on the number of subsequences belonging to a cluster. Since longer subsequences are less often dense the time to investigate longer multiclusters is nearly constant. As far as the quality of the result is concerned, both algorithms discovered the main patterns of the six multiclusters hidden in the database.

In our second experiment we used the same hidden clusters as in our first experiment but varied the number of sequences contained in the data base. Figure 11 shows the result for four different sequence lengths (from 30,000 to 150,000). Even in 48 attributes the MC algorithm is capable to find multiclusters in reasonable time. Once again, we have only slightly more than linear behavior for increasing number of sequences.

Fig. 11 Scalability for number of attributes



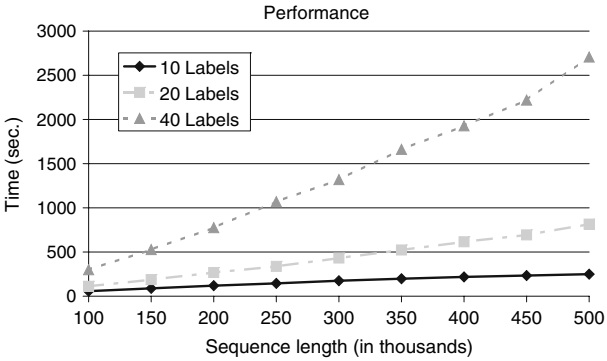


Fig. 12 Scalability for sequence length

To evaluate the scalability in terms of sequence lengths we again used the eight-dimensional data set from our first experiment and recorded the time for sequence lengths from 100,000 to 500,000. Additionally three different domains were used, depicted in Fig. 12 as separate lines for 10, 20 and 40 ordinal values per sequence attribute. In order to keep the experiments comparable we adjusted σ using our density plot heuristics from 0.6, 1.2, 2.4, respectively. The MC algorithm shows linear behavior for 10 labels and slightly superlinear behavior for 20 and 40 labels. Even the largest setup with 40 different categories and 500, 000 sequence segments takes less than 2, 800 s. This means that our algorithm is capable of handling very large databases with large domains.

6.3 Real world data

This section evaluates the effectiveness and efficiency of our algorithm on different real world datasets. Our analysis mainly focuses on the flowing water renaturation project of the European Union mentioned before. Additionally, we investigate the effectiveness of the MC algorithm on temporal data sets, using data from the "National Fire Danger Rating Retrieval System".

6.3.1 River data

The river data set consists of 120,000 river segments describing the structural properties of a river, such as the structure of the riverbed. Experts working on renaturing rivers are interested in finding co-occurring clusters to determine those subsequence patterns which occur repeatedly in the river database and which allow to derive broad potential improvement through measures designated. As a final result, experts should be capable of summarizing river segments into measure packages, ranking them according to the quality enhancements expected. This can then be used to create supra-regional schedules for political decision-making.

For the river dataset a value for σ between 0.7 and 0.85 has shaped up as a reasonable parameter. The MC algorithm identifies more than a thousand clusters of length four by using a value for σ of 0.7 with a corresponding value for δ. Many of those clusters do not show when mining clusters of length five. By using a higher value for σ more patterns are considered similar and cluster count drops between lengths 5 and 6. For this dataset experiments have shown that independent of the σ value, multiclusters in many attributes can be found only at length 4–6. Beyond this length, parallel patterns are rare. This an interesting result for

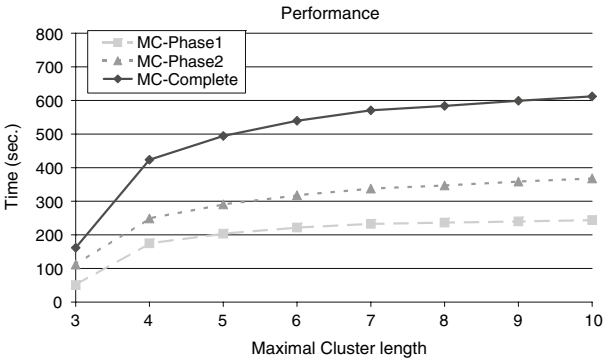


Fig. 13 Performance

the hydrologists studying this data. They find that they should develop sensible packages of measures in this range and estimate costs for packages of about 500 m river improvement.

Figure 13 illustrates the runtime of the MC algorithm for phase one (searching for clusters of arbitrary length) and for phase two (searching for multiclusters) separately as well as for both. As we can see, the time requirements for mining all clusters of arbitrary length are distributed rather evenly between the two phases. The total time for mining multiclusters of arbitrary pattern length demonstrates the efficiency of our approach. Note that with increasing maximal length, few additional dense patterns are detected such that the increase in time consumption slows down. The steepest ascent is for lengths of up to 6, which corresponds to our result findings.

Similar to the synthetic dataset, we also applied SUBCLU on this real world dataset. However, even the first iteration of SUBCLU for multiclusters of length 10 did not finished after 10 h. One reason why MC works extremely faster on the investigated dataset than SUBCLU are the time consuming neighborhood queries on the nineteen-dimensional data points. Even the use of index structures like the R-Tree does not speed up these neighborhood queries in these high dimensionalities. Another reason is the efficient combination of dimensions using an FP-tree as done by MC.

For hydrologists to see how the detected multiclusters are distributed and check in which areas the designed measures are applicable or where additional measure engineering is required, we visualize multiclusters in a geographical information system. An example for a cluster visualization is given in Fig. 14. Those river segments which are part of the cluster are marked by dark lines. The corresponding cluster summary is visualized on the left side. It indicates the cluster length of five river sections (top–bottom) as well as the cluster range of ten out of nineteen attributes (left–right).

Additional tools for hydrologists give detailed information beyond this summary. Experts may browse clusters for an overview of the attributes contained in clusters, their value distributions as well as their occurrence in the rivers. Moreover, individual attributes and river values may be checked for containment in clusters. By joining this information with the packages of measures designed, field experts can easily identify areas which are not yet met by measures.

Annotating the measures with cost and duration information, political decision making is supported. Hydrologists used the information derived from these clusters to build a decision support system [4] that gives concise summaries as well as detailed inspection of the state of the rivers as it is now as well as a prognosis for future development depending on the packages of measures chosen.

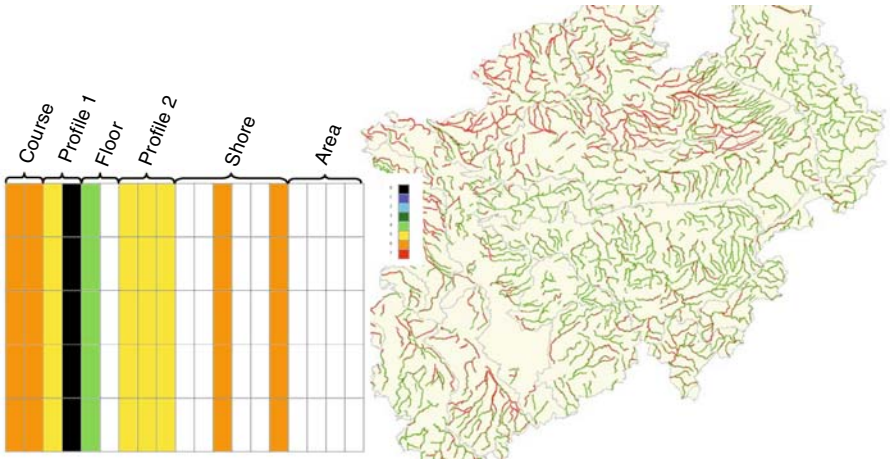


Fig. 14 Cluster visualization for river data

6.3.2 Weather data

The second real world dataset is retrieved from the "National Fire Danger Rating Retrieval System" which provides senor data from various weather stations. We use hourly weather variables from 1998 to 2005 about temperature, relative humidity, wind speed and direction, weather status etc. Overall the dataset contains fifteen variables measured at 27, 048 time points [10]. This weather dataset contains a mixture of real valued and categorical variables. The continuous attributes are transformed into ordered categories using the transformation technique presented in [19]. After normalization we used ten symbols for the quantization.

The weather dataset allows us to determine the quality of the MC algorithm. Based on the weather variables the "National Fire Danger Rating System" calculates some indices like the "Ignition Component" which relates the probability that a fire that requires suppression action will result if a firebrand is introduced into a fine fuel complex. These indices are influenced by many variables measured in the rating system. They form natural co-occurring patterns in the data which we use to demonstrate that the MC algorithm does find these multiclusters. By inspecting the result we indeed find the correlation between the index variables and all measured sensor data. Figure 15 (right side) illustrates an example cluster which nearly contains all attributes. The cluster overview (Fig. 15 left side) shows many clusters containing nearly all attributes (13 and 14) with a length of up to 7. Thus the MC algorithm indeed finds the intrinsic structure in a real valued dataset.

7 Conclusion and further work

Domain experts are supported in their need for pattern detection in sequence databases such as river quality records. The information derived using our approach is incorporated in a decision support system devised by hydrologists. Future work will concentrate on integrating GIS information to handle non-sequence spatial information as well and to extend the model to graph structures [17].

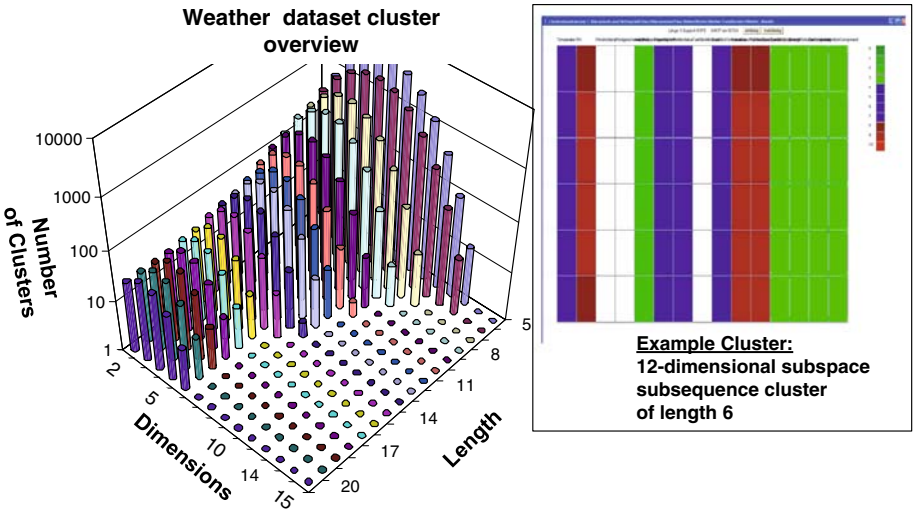


Fig. 15 Clustering of weather dataset

Acknowledgments We would like to thank K. Kailing, H.-P. Kriegel and P. Kroeger for providing the original source code of SUBCLU.

References

1. Agrawal R, Srikant R (1995) Mining sequential patterns. In: IEEE international conference on data engineering (ICDE), pp. 3–14
2. Assent I, Krieger R, Müller E, Seidl T (2007) DUSC: dimensionality unbiased subspace clustering. In: IEEE international conference on data mining (ICDM), pp. 409–414
3. Ayres J, Flannick J, Gehrke J, Yiu T (2002) Sequential pattern mining using a bitmap representation. In: ACM SIGKDD international conference on knowledge discovery and data mining, pp. 429–435
4. Bartusseck S (2005) Regelbasiertes Entscheidungsunterstützungssystem (DSS) zur Bewertung von Maßnahmenplänen gemäß EG-WRRL. Forum für Hydrologie und Wasserbewirtschaftung 10
5. Brecheisen S, Kriegel H, Pfeifle M (2006) Multi-step density-based clustering. Knowl Inf Sys 9(3):284–308
6. Coatney M, Parthasarathy S (2005) MotifMiner: efficient discovery of common substructures in biochemical molecules. Knowl Inf Sys 7(2):202–223
7. Denton A (2004) Density-based clustering of time series subsequences. In: IEEE international conference on data mining (ICDM)
8. Ester M, Kriegel H, Sander J, Xu X (1996) A density-based algorithm for discovering clusters in large spatial databases. In: ACM SIGKDD international conference on knowledge discovery and data mining, pp. 226–231
9. Faloutsos C, Ranganathan M, Manolopoulos Y (1994) Fast subsequence matching in time-series databases. In: ACM SIGMOD international conference on management of data, pp. 419–429
10. Georgia Forestry Commission (2005) Weather data retrieval. http://weather.gfc.state.ga.us
11. Grahne G, Zhu J (2004) Mining frequent itemsets from secondary memory. In: IEEE international conference on data mining (ICDM), pp. 91–98
12. Guha S, Rastogi R, Shim K (1999) A robust clustering algorithm for categorical attributes. In: IEEE international conference on data engineering (ICDE), pp. 512–521
13. Han J, Pei J, Yin Y (2000) Mining frequent patterns without candidate generation. In: ACM SIGMOD international conference on management of data, pp. 1–12
14. Hinneburg A, Keim D (1998) An efficient approach to clustering in large multimedia databases with noise. In: ACM SIGKDD international conference on knowledge discovery and data mining, pp. 58–65

15. Hinneburg A, Keim D (2003) A general approach to clustering in large databases with noise. Knowl Inf Sys 5(4):387–415
16. Kailing K, Kriegel H, Kröger P (2004) Density-connected subspace clustering for high-dimensional data. In: IEEE international conference on data mining (ICDM), pp. 246–257
17. Kailing K, Kriegel H, Schonauer S, Seidl T (2004) Efficient similarity search for hierarchical data in large databases. In: international conference on extending database technology (EDBT), pp. 676–693
18. Keogh E, Chakrabarti K, Pazzani M, Mehrotra S (2001) Locally adaptive dimensionality reduction for indexing large time series databases. In: ACM SIGMOD international conference on management of data, pp. 151–162
19. Lin J, Keogh E, Lonardi S, Chiu B (2003) A symbolic representation of time series, with implications for streaming algorithms. In: Workshop on research issues in data mining and knowledge discovery at ACM SIGMOD international conference on management of data, pp. 2–11
20. LUA NRW (2003) River quality data, http://www.lua.nrw.de.
21. MacQueen J (1967) Some methods for classification and analysis of multivariate observations. In: Berkeley symposium on mathematical statistics and probability, pp. 281–297
22. Patel P, Keogh E, Lin J, Lonardi S (2002) Mining motifs in massive time series databases. In: IEEE international Conference on data mining (ICDM)
23. Zaki M, Peters M, Assent I, Seidl T (2005) Clicks: An effective algorithm for mining subspace clusters in categorical datasets. In: ACM SIGKDD international conference on knowledge discovery and data mining, pp. 355–356

Author Biographies

Ira Assent is currently a PhD student and research assistant at RWTH Aachen University, Germany. Her research interests are in efficient similarity search and subspace clustering in large multimedia databases. In 2003 she received her Diplom (equivalent to MSc) in computer science from RWTH Aachen University.

Ralph Krieger received a Diploma degree in computer science from the RWTH Aachen University in 2002 (equivalent to MSc). From 2002 to 2003 he worked as a system developer for MICOS GmbH—Large Scale Security Systems. Currently he is working as a PhD student at the Data Management and Data Exploration Group of RWTH Aachen University. His research interest include clustering and subspace clustering of high-dimensional data and sequence databases. He is also interested in density estimation techniques for anytime classification.

Boris Glavic received a Diplom degree from RWTH Aachen University, Germany in 2005. Since then he is working as a PhD student at the Database Technology research group, University of Zürich. His research interests include data mining, data provenance and database implementation and indexstructures.

Thomas Seidl is a full professor for computer science and head of the data management and data exploration group at RWTH Aachen University, Germany. His research interests include data mining and data management in multimedia and spatio-temporal databases for applications from computational biology, medical imaging, mechanical engineering, computer graphics, etc. with a focus on content, shape or structure of complex objects in large databases. His research in the field of relational indexing aims at exploiting the robustness and high performance of relational database systems for complex indexing tasks. Having received his MS in 1992 from the Technische Universität München, Seidl received his PhD in 1997 and his venia legendi in 2001 from the University of Munich, Germany.