Available online at www.sciencedirect.com
ScienceDirect
Data & Knowledge Engineering 60 (2007) 51–70
DATA & KNOWLEDGE ENGINEERING
www.elsevier.com/locate/datak

CLICKS: An effective algorithm for mining subspace clusters in categorical datasets ☆

Mohammed J. Zaki [a,*], Markus Peters [b], Ira Assent [b], Thomas Seidl [b]

[a] *Department of Computer Science, Rensselaer Polytechnic Institute, Lally 307, 110 8th St., Troy, NY 12180-3590, United States*
[b] *RWTH-Aachen, Germany*

Available online 3 March 2006

Abstract

We present a novel algorithm called CLICKS, that finds clusters in categorical datasets based on a search for k-partite maximal cliques. Unlike previous methods, CLICKS mines subspace clusters. It uses a *selective vertical method* to guarantee complete search. CLICKS outperforms previous approaches by over an order of magnitude and scales better than any of the existing method for high-dimensional datasets. These results are demonstrated in a comprehensive performance study on real and synthetic datasets.

Keywords: Clustering; Categorical data; k-Partite graph; Maximal cliques

1. Introduction

Clustering is one of the central data mining problems; it aims to find "naturally" occurring groups of points in a given dataset. Clustering of numeric (or real-valued) data has been widely studied, but categorical (or discrete-valued, symbolic) data has received relatively less attention. There are several challenges in clustering categorical attributes: (i) *No Natural Order:* The lack of an inherent natural order on the individual domains, renders a large number of traditional similarity measures ineffective. (ii) *High Dimensionality:* Practical examples suggest that categorical data can have many attributes, requiring methods that scale well with dimensionality. (iii) *Subspace Clusters:* Many categorical datasets, especially sparse ones, do not exhibit clusters over the full set of attributes, thus requiring subspace clustering methods.

In this paper, we present CLICKS,[1] a novel algorithm for mining categorical (subspace) clusters. Our main contributions are: (1) We present a novel formalization of categorical clusters. We summarize the dataset as a

☆ This work was supported in part by NSF Career Award IIS-0092978, DOE Career award DE-FG02-02ER25538, and NSF grants EIA-0103708 and EMT-0432098.
* Corresponding author. Tel.: +1 518 276 6340; fax: +1 518 276 4033.
E-mail addresses: zaki@cs.rpi.edu (M.J. Zaki), peters@informatik.rwth-aachen.de (M. Peters), assent@informatik.rwth-aachen.de (I. Assent), seidl@informatik.rwth-aachen.de (T. Seidl).
[1] CLICKS is an anagram of the bold letters in **S**ubspace **CL**uster**I**ng of **C**ategorical data via maximal **K**-partite cliques.

0169-023X/$ - see front matter
doi:10.1016/j.datak.2006.01.005

k-partite graph, and mine maximal k-partite cliques, which after post-processing correspond to the clusters. The k-partite maximal clique mining method is interesting in its own right. (2) CLICKS uses a *selective vertical expansion* approach to guarantee complete search; no valid cluster is missed. It also merges overlapping cliques to report more meaningful clusters. (3) CLICKS addresses the main shortcomings of existing methods. Unlike many previous categorical clustering algorithms, CLICKS can mine subspace clusters. Furthermore, it imposes no domain constraints and is scalable to high dimensions. (4) CLICKS outperforms existing approaches by over an order of magnitude, especially for high-dimensional datasets. These results are demonstrated in a comprehensive performance study on real and synthetic datasets.

2. Preliminaries

Let $A_1, \ldots, A_n$ be a set of *categorical attributes* and $D_1, \ldots, D_n$ a set of *domains*, where $D_i = \{v_{i_1}, \ldots, v_{i_m}\}$ is the domain for attribute A_i, and $D_i \cap D_j = \emptyset$ for $i \neq j$. A *dataset* is a subset of the Cartesian product of the attribute domains, given as $\mathscr{D} \subseteq D_1 \times \cdots \times D_n$. The number n of attributes is also referred to as the *dimensionality* of the dataset. An element $\mathbf{r} = (r.A_1, \ldots, r.A_n) \in D$ is called a *record*, where $r.A_i \in D_i$ refers to the value for attribute A_i in $\mathbf{r}$. Each record also has a unique record id (rid), given as $\mathbf{r}.id$.

Let $S_j \subseteq D_{i_j}$ a subset of values for attribute A_{i_j}. A k-subspace is defined as the cross-product $S = S_1 \times \cdots \times S_k$ of some subset of k attributes $A_{i_1}, \ldots, A_{i_k}$. Each S_j is called a *projection* of S on attribute A_{i_j}. If $k = n$, then the n-subspace is also called a *full-space*. Given any two subspaces $X = X_1 \times \cdots \times X_m$ and $Y = Y_1 \times \cdots \times Y_n$, we say that X is *contained* within Y, denoted as $X \subseteq Y$, iff $m \leqslant n$ and $\forall i \in [1,m]$ there exists a unique $j \in [1,n]$, such that $X_i \subseteq Y_j$. Given any collection $\mathscr{S}$ of subspaces, $M \in \mathscr{S}$ is a *maximal subspace* iff there does not exist $M' \in \mathscr{S}$, such that $M \subset M'$.

Let $S = S_1 \times \cdots \times S_k$ be a k-subspace, with $k \leqslant n$. A record $\mathbf{r} = (r.A_1, \ldots, r.A_n) \in \mathscr{D}$ *belongs* to S, denoted $\mathbf{r} \in S$, iff $r.A_{i_j} \in S_j$ for all $j \in [1,k]$. The *support* of S in dataset $\mathscr{D}$ is defined as the number of records in the dataset that belong to it; it is given as $\sigma(S) = |\{\mathbf{r} \in \mathscr{D} : \mathbf{r} \in S\}|$. S is called a *frequent subspace* if $\sigma(S) \geqslant \sigma^{\min}$, where $\sigma^{\min}$ is some user-defined minimum support threshold.

Under attribute independence, the *expected support* [20] of S in $\mathscr{D}$ is given as $E[\sigma(S)] = |\mathscr{D}| \cdot \prod_{j=1}^{k} \frac{|S_j|}{|D_{i_j}|}$. Additional a-priori information about the expected dataset distribution can also be integrated at this point by modifying the definition for $E[\sigma(S)]$.

Let $\alpha \in \mathbb{R}^+$. Define a *density indicator* function $\delta_\alpha(S)$ as follows: $\delta_\alpha(S) = 1$ iff $\sigma(S) \geqslant \alpha \cdot E[\sigma(S)]$, otherwise $\delta_\alpha(S) = 0$. S is called a *dense subspace* iff $\delta_\alpha(S) = 1$, that is, if its expected support exceeds its actual support by a user-defined factor α.

Two sets of projections S_i and S_j, are called *strongly connected* iff $\forall v_a \in S_i$ and $\forall v_b \in S_j$, the 2-subspace $\{v_a\} \times \{v_b\}$ is dense. $S = S_1 \times \cdots \times S_k$ is called a *strongly connected subspace* iff S_i is strongly connected to S_j for all $1 \leqslant i < j \leqslant k$.

Definition 1 (*Categorical Cluster*). Let $\mathscr{D}$ be a categorical dataset and $\alpha \in \mathbb{R}^+$. The k-subspace $C = (C_1 \times \cdots \times C_k)$ is a (*subspace*) *cluster* over attributes $A_{i_1}, \ldots, A_{i_k}$ iff it is a maximal, dense, and strongly connected subspace in $\mathscr{D}$. The projection C_i is also called the *cluster projection* of C on attribute A_{i_j}. If $k < n$, then C is called a *subspace cluster* or a k-cluster, otherwise C is called a *full-space cluster*.

Our cluster definition requires the k-subspace C be dense, i.e., it should enclose more points than expected under attribute independence. Also only maximal clusters are mined to reduce the amount of redundant information. Ideally one would like to discover only maximal, dense subspaces as clusters. However notice that density is not *downward closed* (i.e., for a dense subspace Y there may exist a subspace $X \subset Y$, such that X is not dense), which makes it difficult to prune the search space. However, we believe that dense subspaces are more informative than say purely frequent ones. To make the search for dense spaces tractable, our cluster definition uses the notion of strong connectedness, which is downward closed. This enables new candidate subspaces to be extended from existing (strongly connected) ones, leading to more efficient search.

Example 2. Consider the sample dataset $\mathscr{D}$ given in Fig. 1 with a total of three categorical attributes A_1, A_2, A_3 and six displayed records. Here $D_1 = \{a_1, a_2, a_3\}, D_2 = \{b_1, b_2, b_3\}, D_3 = \{c_1, c_2, c_3\}$. Let $\alpha = 2.5$. Assuming that attributes and their values are independent the expected support for any pair of values, i.e., the 2-subspace

ID	A_1	A_2	A_3
1	a_1	b_1	c_1
2	a_2	b_3	c_2
3	a_2	b_3	c_3
4	a_2	b_1	c_1
5	a_2	b_3	c_3
6	a_3	b_3	c_3
	$\vdots$		

Fig. 1. Sample categorical dataset.

$\{v_i\} \times \{v_j\}$ from different attributes A_i and A_j, is $E[\sigma(\{v_i\} \times \{v_j\})] = 1/3 \times 1/3 \times 6 = 0.67$. Thus any pair of values that occurs at least 2 times (i.e., $\sigma(\{v_i\} \times \{v_j\}) = 2$) will be dense (since $2/0.67 = 2.98 > \alpha$). Thus for $\alpha = 2.5$ there is only one full-space cluster in this dataset ($\{a_2\} \times \{b_3\} \times \{c_3\}$); there is an additional subspace cluster: ($\{b_1\} \times \{c_1\}$).

On the other hand, if we use $\alpha = 1.5$, then any pair of values that occurs once will be considered dense. Thus for $\alpha = 1.5$, there are 3 full-space clusters in this dataset: ($\{a_1, a_2\} \times \{b_1\} \times \{c_1\}$), ($\{a_2, a_3\} \times \{b_3\} \times \{c_3\}$), and ($\{a_2\} \times \{b_3\} \times \{c_2, c_3\}$). There are two additional subspace clusters: ($\{a_2\} \times \{b_1, b_3\}$), and ($\{a_2\} \times \{c_1, c_2, c_3\}$). Note that $\{b_3\} \times \{c_2\}$ is not a subspace cluster because is not maximal (it is included in $\{a_2\} \times \{b_3\} \times \{c_2, c_3\}$). Note also that an interesting property of our approach is that the clusters found for higher α will always be contained in a lower α, which can allow the user to produce a cluster hierarchy.

2.1. The CLICKS approach

CLICKS models a categorical dataset as a k-partite graph where the vertex set (attribute values) is partitioned into k disjoint sets (one per attribute) and edges exist only between vertices in different partitions, indicating dense relationships. In essence, the adjacency matrix of the k-partite graph serves as a compressed representation of the data, and can fit into main memory for even very large datasets. CLICKS then maps the categorical clustering problem to the problem of enumerating maximal k-partite cliques in the k-partite graph.

Definition 3 (*k-Partite Graph and Clique*). Let $\mathscr{D}$ be a categorical dataset over attributes $A_1, \ldots, A_n$ and $V = \bigcup_{i=1}^{n} D_i$. The undirected graph $\Gamma_{\mathscr{D}} = (V, E)$ where $(v_i, v_j) \in E \iff \delta_\alpha(\{v_i\} \times \{v_j\}) = 1$ is called the *k-partite graph* of $\mathscr{D}$. A subset $C \subseteq V$ is a *k-partite clique* in $\Gamma_{\mathscr{D}}$ iff every pair of vertices $v_i \in C \cap D_i$ and $v_j \in C \cap D_j$ (with $i \neq j$) are connected by an edge in $\Gamma_{\mathscr{D}}$. If there is no $C' \supset C$ such that C' is a k-partite clique in $\Gamma_{\mathscr{D}}$, C is called a *maximal k-partite clique*. A clique C is *dense* if $\delta_\alpha(C) = 1$ in $\mathscr{D}$.

Lemma 4. *Given a categorical dataset $\mathscr{D}$ and a k-subspace $C = C_1 \times \cdots \times C_k$ with $C_j \subseteq D_{i_j}$ over attributes $A_{i_1}, \ldots, A_{i_k}$. C is a k-cluster in $\mathscr{D}$ if and only if C is a maximal, dense k-partite clique in $\Gamma_{\mathscr{D}}$.*

Proof. By Definition 3, for dataset $\mathscr{D}$ and k-partite graph $\Gamma_{\mathscr{D}} = (V, E)$, $(v_a, v_b) \in E \iff \delta_\alpha(\{v_i\} \times \{v_j\}) = 1$. Thus C is a clique iff C is strongly connected. Finally both definitions require $\delta_\alpha(C) = 1$ and that C be maximal. □

Example 5. Consider the example dataset $\mathscr{D}$ shown in Fig. 1; let $\alpha = 2.5$. From Example 2 we know that each pair of values that occurs at least 2 times is dense in $\mathscr{D}$, and thus there is an edge between such vertices in $\Gamma_{\mathscr{D}}$. The corresponding k-partite graph of $\mathscr{D}$ is shown in Fig. 2, using bold edges. It clearly has two clusters, one

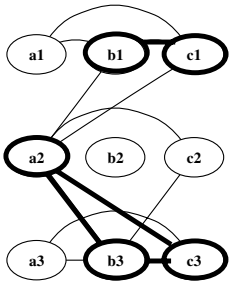


Fig. 2. k-Partite graph of $\mathscr{D}$.

full-space and one sub-space. If $\alpha = 1.5$, then some other (thin) edges will be added to the graph. Mining this new graph will produce the larger set of clusters mentioned in Example 2. It should be clear that clusters mined at $\alpha = 2.5$ are contained in those at $\alpha = 1.5$, since a lower α only *adds* edges to $\Gamma_{\mathscr{D}}$.

3. Related work

Many full-space clustering algorithms have been proposed in the past; we refer the reader to any good book on data mining [1]. We focus here only on the relevant research in subspace mining and categorical clustering.

Note that our k-partite model is different from the notion of bi-clusters [2], which uses a bi-partite model for $\mathscr{D}$, where one vertex set U denotes the set of record identifiers, and the other set $V = \bigcup_{i=1}^{n} D_i$, the set of all attribute values. We can then add an edge between $u \in U$ and $v \in V$ iff v appears in record u. One could claim that a cluster is equivalent to a maximal bipartite clique (X, Y) with $X \subseteq U$ and $Y \subseteq V$. However, this bi-partite model can never mine a bipartite clique of the form $(X, \{a_1, a_2, b_1, c_1\})$, corresponding to the cluster $(\{a_1, a_2\} \times \{b_1\} \times \{c_1\})$ from Example 2, since no record $u \in U$ can be connected to both a_1 and a_2 (they never co-occur). If we consider such edges, then the bi-partite model becomes essentially the same as our k-partite model.

3.1. Subspace clustering

Since CLIQUE [3] introduced the problem, many subspace clustering techniques have been proposed [4–8]. Also relevant is the problem of detecting bi-clusters from numeric data, especially in bioinformatics applications [9]. Let R denote the set of record identifiers, and A the set of attributes. A *bi-cluster* is a pair (U, V), with $U \subseteq R$, $V \subseteq A$, such that the mean square error for values in the submatrix specified by rows in U and columns in V is below some threshold.

None of these methods are designed for categorical datasets, they work only for numeric data. Adapting them to mine categorical data introduces several problems that are not easily overcome. Methods like CLIQUE [3], MAFIA [4], and SCHISM [5] that discretize the numeric data, rely on the dense regions to be ordered (or contiguous), whereas no such order exists for categorical data. Methods such as DOC [6], PROCLUS [7], ORCLUS [7], bi-clustering [9] require numerical computations (e.g., mean, median, variance), and are thus inherently not well suited for categorical datasets. Methods based on DBSCAN [10], such as RIS [8] would have problems for the verification of the core object property in different subspaces.

3.2. Categorical clustering

Recently, a number of studies have focused on categorical clustering. With the exception of CACTUS [20], which itself mines only a limited class of subspace clusters, none of the previous methods can mine subspace clusters. Previous works on categorical data have focused more narrowly on binary or transactional data [11,12], on a framework to compress high-dimensional categorical datasets [13], and using hypergraph partitioning to cluster itemsets [14].

Note that approaches that cluster only binary/transactional data are not able to directly handle the kinds of clusters we propose here. For example, consider Fig. 1. If we create an item for each attribute-value pair (e.g., $A_1 = a_1$, $A_1 = a_2$, etc.), then a transactional clustering method can be employed. However these methods will typically not mine a cluster like $(\{a_1, a_2\} \times \{b_1\} \times \{c_1\})$ as shown in Example 2. The reason is that by definition a_1 and a_2 never co-occur in any transaction, so unless some appropriate notion of similarity between values of the same attribute is defined, these values will not be part of the same cluster.

k-Modes [15] is an extension to the k-means numeric clustering algorithm. Whereas it shares the scaling properties of k-means, it also inherits k-means' disadvantages, such as dependence on the seed clusters, and the inability to automatically detect the number of clusters.

COOLCAT [16] is based on the idea of entropy reduction within the generated clusters. It first *bootstraps* initial clusters using a sample of maximally dissimilar points and then adds the remaining points. To mitigate the high dependency on the selection order, periodic re-clustering of selected points is proposed. Another entropy based method was proposed in [17]. Starting from a seed clustering, they use genetic algorithms to heuristically improve the purity of the generated clusters. The quality of the resulting clusters depends on a-priori knowledge of the "importance" of the individual attributes toward the "natural" clustering. LIMBO [18] is a recent information theoretic clustering based on an information-bottleneck framework. Like the other methods it cannot find subspaces.

STIRR [19] uses a non-linear dynamical systems approach to categorical clustering. It encodes the dataset into a weighted graph (each attribute value corresponds to a weighted vertex), and iteratively propagates these weights; this intuitively corresponds to a similarity measure based on co-occurrence of values in the dataset. Upon convergence to the fix points, or the so-called *basins*, the final weights can be used to partition the data points, yielding the final clusters. The main weakness of STIRR is that the separation of attribute values by their weights is non-intuitive and the post-processing required to extract the actual clusters from the basin weights is non-trivial. The combination and local modification operations are also left to the user to find. Further, the mined clusters were shown to be incomplete in cases of overlapping cluster projections [20].

ROCK [21] is based on the number of *links* between two records; links capture the number of other records that the two are both sufficiently similar to. ROCK heuristically optimizes a cluster quality function with respect to (w.r.t.) the number of links in an agglomerative hierarchical fashion. The base algorithm is cubic in the dataset size making it unsuitable for large problems.

CACTUS [20] mines categorical clusters utilizing a summary information of the dataset. The cluster definition is essentially the same as in Definition 1. It first computes cluster projections onto the individual attributes. To reduce the complexity of this step, the authors assume the existence of a *distinguishing number* (κ), the minimum size of an attribute-value set (the so-called *distinguishing set*) that uniquely occur within only one cluster. CACTUS computes distinguishing sets on each attribute and then extends them to find cluster candidates over multiple attributes. These candidates have to be validated against the original dataset. The extension step, though described in the paper, was not implemented by the authors, but our augmented implementation showed a severe performance impact over the cactus baseline version (see Section 5.2). The authors proposed to apply the MDL pruning approach used in [3] for subspace clustering, but it was also never implemented. Note that whereas CLICKS uses the cluster definition proposed in CACTUS, the encoding of the dataset as a k-partite graph and the clusters as k-partite cliques is entirely novel, and we believe, more natural (e.g., we do not have to impose the distinguishing set constraint). It is interesting to note that the proposed CACTUS extension in [20], discovers subspace clusters only in the ordered subspaces (A_1, A_2), $(A_1, A_2, A_3), \ldots, (A_1, \ldots, A_n)$, whereas, CLICKS mines subspace clusters over any subset of attributes. Thus CLICKS overcomes the three main problems with CACTUS, namely (1) "unnatural" distinguishing set assumption, (2) no extension step after cluster projections are found, and (3) mining of a restricted class of subspace clusters.

4. The CLICKS algorithm

Given a dataset $\mathscr{D}$ and a user-specified threshold $\alpha \in \mathbb{R}^+$, we are interested in mining all full-space and subspace clusters (i.e., all maximal, dense, and strongly connected subspaces) in $\mathscr{D}$.

As we noted earlier, density is not downward closed, and thus cannot be used directly to prune the search for valid subspaces. Instead we will use the strongly-connected property to mine $\mathscr{L}$, the set of all maximal

k-partite cliques in $\Gamma_{\mathscr{D}}$. We then follow-up with a validation step, that verifies whether $\delta_\alpha(C) = 1$ for all cliques $C \in \mathscr{L}$. This two-step approach is very efficient, but it is not complete, since it is possible that some maximal clique $C \in \mathscr{L}$ is not dense, whereas its subset $C' \subset C$ might be dense. To guarantee completeness CLICKS uses as another step the *selective vertical expansion* technique to enumerate subspaces of a non-dense maximal clique. Our experiments show that most of the final clusters can be found using only the first two steps, but if completeness is desired, all clusters will be guaranteed to be found for an additional cost. It should be noted that even with selective vertical expansion CLICKS is faster than previous categorical clustering methods. Note that CLICKS can mine maximal k-partite cliques for any $1 \leqslant k \leqslant n$; if $k = n$, the discovered cliques are clusters over the full set of dimensions, and if $k < n$ then the discovered cliques are subspace clusters. We also note that CLICKS is flexible enough to mine only (maximal) frequent clusters if so desired (a minor change in the pre-processing step accomplishes this).

The basic CLICKS approach consists of the three principal stages, shown in Fig. 3, as follows:

- *Pre-processing*: We create the k-partite graph from the input database $\mathscr{D}$. We also rank the attributes for efficiency reasons.
- *Clique detection*: We enumerate all the maximal k-partite cliques in the graph $\Gamma_{\mathscr{D}}$.
- *Post-processing*: We verify the support of the candidate cliques within the original dataset to form the final clusters. If completeness is desired we perform selective sub-clique expansion of non-dense maximal cliques to find the true maximal, dense cliques. Moreover, the final clusters are optionally merged if they have significant overlap.

4.1. Pre-processing

In the pre-processing step we take as an input the categorical dataset $\mathscr{D}$ and the threshold α. In one scan of the dataset we collect the support of every attribute value, and also the support of every pair of attribute values. These support values are used to compute all the dense attribute value pairs; for each such pair $\{v_a\} \times \{v_b\}$ we add the edge (v_a, v_b) to $\Gamma_{\mathscr{D}}$, creating the full k-partite graph $\Gamma_{\mathscr{D}}$. Further, we rank the set of all attribute values using the notion of connectivity, defined below. This ranking is used later for efficient clique enumeration.

Given $\Gamma_{\mathscr{D}}$ and $V = \bigcup_{i=1}^{n} D_i = \{v_1, \ldots, v_m\}$, the *neighbors* of an attribute value v_j are defined as $N(v_j) = \{v_k \in V : (v_j, v_k) \in E\}$. Note also that, by definition, if $v_j, v_k \in D_i$ then $v_k \notin N(v_j)$, since values of the same attribute never co-occur. However, for the clique enumeration step, we have to consider all values of an attribute to be implicitly connected.

The *connectivity* of vertex $v_j \in D_i$ is defined as

$$\eta(v_j) = \begin{cases} N(v_j) \cup \{D_i \setminus v_j\} & \text{if } |N(v_j)| > 0 \\ 0 & \text{otherwise} \end{cases}$$

CLICKS(**Dataset** $\mathcal{D}$, α, $\sigma^{\mathcal{C}}$)
  AttributeValueRanking: $\mathcal{R} = \bigcup_{i=1}^{n} D_i$
  Clique $C = \emptyset$
  CliqueCollection $\mathcal{L} = \emptyset$

  PreProcess$(\mathcal{D}, \alpha, \Gamma_{\mathcal{D}}, \mathcal{R})$
  DetectMaxCliques$(\Gamma_{\mathcal{D}}, \mathcal{L}, \mathcal{R}, C)$
  PostProcess$(\mathcal{D}, \mathcal{L}, \alpha, \sigma^{\mathcal{C}})$
  return $\mathcal{L}$

Fig. 3. The CLICKS algorithm.

Intuitively, connectivity corresponds to the neighbors ($N(v_j)$) plus the remaining values of the attribute in question ($D_i \backslash v_j$). However, if a given value does not co-occur with values of other attributes it cannot be part of a k-partite clique; its connectivity should be zero. The connectivity of a clique C is given as follows: $\eta(C) = \bigcap_{v_j \in C} \eta(v_j)$, i.e., the connectivity common to all vertices in C.

The total order on all attribute values, given by the set $\mathscr{R} = \{v_{i_1}, \ldots, v_{i_m}\}$, such that $|\eta(v_{i_j})| \geqslant |\eta(v_{i_{j+1}})|$ for all $j \in [i_1, i_{m-1}]$, is called an *attribute-value ranking* of V. Given a seed clique, CLICKS adds a new vertex to the clique based on the next highest ranked value. This can significantly speed-up the search for maximal cliques.

4.2. Enumerating k-partite maximal cliques

The clique detection phase is based on a backtracking search, that at each step, adds only those vertices to a clique that are in the connectivity set of the clique. If more than one such vertex exists, the attribute value ranking is used to break the tie. CLICKS uses a recursive algorithm that at each stage tries to expand the current clique to ensure maximality. It is similar in spirit to the Bron–Kerbosch (BK) algorithm [22], but whereas BK enumerates regular cliques, CLICKS is designed for k-partite cliques. The pseudo-code for the clique detection phase is shown in Fig. 4.

Initially `DetectMaxCliques` is called with the empty clique C and the full, ranked attribute value set $\mathscr{R}$ as a list of possible vertices to be used for an extension. In general, $\mathscr{R}$ represents the set of vertices that can be used to extend the current clique C. Upon return, the clique collection $\mathscr{L}$ contains all maximal k-partite cliques in the dataset.

Note that `foreach` statements process attribute value rankings in descending order. The predicate $\Phi(C)$ evaluates to *true* iff (i) we want to mine subspace clusters, or (ii) we want to mine full space clusters and C contains at least one attribute value for every attribute of the dataset. Otherwise $\Phi(C)$ is *false*. The set $\mathscr{R}^D$ contains all elements of $\mathscr{R}$ that have their *deleted* flag set. Similarly, $\mathscr{R}^P$ is the subset of $\mathscr{R}$ that contains all elements that have their *processed* flag set.

`DetectMaxCliques` starts by checking if the current clique C is maximal (lines 1–2). If $\mathscr{R} = \emptyset$ then there are no more elements to extend C, thus C is potentially maximal. If in addition $\eta(C) = \emptyset$ then C is a maximal clique, since an empty connectivity set means there are no additional vertices connected to all vertices in C. The only test that remains to be done is whether full/sub-space cliques are desired. For subspace clusters $\Phi(C)$ is always true, whereas for full-space clusters $\Phi(C)$ is true only if C contains at least one value from each attribute. Thus, C is added to the set of maximal cliques $\mathscr{L}$ iff $\Phi(C)$ is true (line 2), and the search is continued at the previous level (line 3).

If $\mathscr{R} \neq \emptyset$ then C can potentially be extended with vertices in $\mathscr{R}$. The outer loop (line 5) attempts to add a value v to C in an effort to create a yet larger clique C' (line 6). Note also that at any given point in time $\mathscr{R}$

```
DetectMaxCliques(Graph Γ_D, CliqueList L,
                 AttributeValueRanking R, Clique C)
1.  if (R = ∅) then
2.      if (η(C) = ∅ and Φ(C)) then L = L ∪ C
3.      return
4.  R^D = R^P = ∅
5.  foreach v in R − R^D − R^P do
6.      C' = C ∪ {v}; R' = ∅;
7.      R^D = R^D ∪ {v}
8.      foreach v' in R − R^D do
9.          if (v' ∈ η(v)) then R' = R' ∪ {v'}
10.     if (v is first value in R) then R^P = R'
11.     if (Φ(R' ∪ C')) then
12.         DetectMaxCliques(Γ_D, L, R', C')
```

Fig. 4. The CLICKS clique detection.

contains only those attribute values that are connected to C. Hence, adding $v \in \mathscr{R}$ to C will yield another clique C'. We mark v as deleted ($\mathscr{R}^D = \mathscr{R}^D \cup \{v\}$), indicating that it was already considered in the clique construction (line 7).

To maintain the condition that all attribute values in $\mathscr{R}$ are connected to C, a $\mathscr{R}'$ matching C' needs to be constructed before the recursive call. The inner `foreach` loop (line 8) scans all attribute values that were possible extensions to C and selects only those (line 9) that are in the connectivity set of v that was added to C in line 6. For the first vertex in $\mathscr{R}$, we maintain a list of nodes already considered in $\mathscr{R}^P$ (line 10).

Finally, the algorithm recurses on the newly created clique C' with its matching attribute value ranking $\mathscr{R}'$. If only full-dimensional clusters are to be detected we can prune part of the search space at this point; we can stop the recursion if the new clique C' cannot be extended to cover at least one value from all attributes, i.e., we recurse only if $\Phi(\mathscr{R}' \cup C')$ is true (lines 11–12).

Both $\mathscr{R}^D$ and $\mathscr{R}^P$ are also used for pruning. Consider two possible extensions v_1 and v_2 of a clique C. If an extension by v_1 was attempted before, the set of possible extensions to v_2 ($\mathscr{R}'$) does not need to contain v_1. If a clique containing both v_1 and v_2 exists, it was discovered when C was extended by v_1, because in that case v_1 and v_2 form a dense 2-subspace and, hence, v_2 was part of the $\mathscr{R}'$ accompanying v_1. The set $\mathscr{R}^D$ prunes these cases by recording every value that has already been used to extend C. Similarly, if v_2 was already part of the $\mathscr{R}'$ accompanying v_1, it need not be considered as an extension to C. This latter case is guarded against by the *processed* attribute values $\mathscr{R}^P$.

Example 6. Consider the k-partite graph encoding $\Gamma_{\mathscr{D}}$ in Fig. 2. An attribute value ranking of V is as follows: $a_2(7)$, $b_1(6)$, $c_1(6)$, $b_3(6)$, $c_3(5)$, $a_1(4)$, $c_2(4)$, $a_3(4)$, $b_2(0)$, where the connectivity cardinalities $|\eta(v)|$ are given in parentheses. Fig. 5 shows a run of `DetectMaxCliques` on this example. Vertices depicted without circles denote search paths that were pruned due to $\mathscr{R}^P$, whereas bold squares indicate that a maximal clique was found. By following the edges up to the root we can construct the corresponding cliques. The $\mathscr{R}'$ sets can be read from the figure by computing the union of all children of a node. For example, $\mathscr{R}'$ for clique $\{a_2, b_1\}$ (in the leftmost path) is $\{c_1, b_3, a_1\}$. This example shows all five full and subspace maximal cliques. For example $\{a_2, b_1, c_1, a_1\}$ is a full space clique, whereas $\{a_2, b_1, b_3\}$ is a subspace clique.

4.3. Post-processing

Once the set of all the maximal k-partite (or n-partite) cliques $\mathscr{L}$ have been mined, the post-processing phase involves a single scan of the dataset to count, for each candidate clique $C \in \mathscr{L}$, the number of transactions in the dataset that support it. If $\delta_\alpha(C) = 1$, i.e., the support of C is at least α times its expected support, then C is a valid clique, and CLICKS outputs it as a cluster. However, there are two challenges that remain: (1) a maximal clique may fail the density test, whereas one of its sub-cliques may be dense. To guarantee completeness, CLICKS allows an optional *selective vertical expansion* approach to explore the sub-cliques induced by a

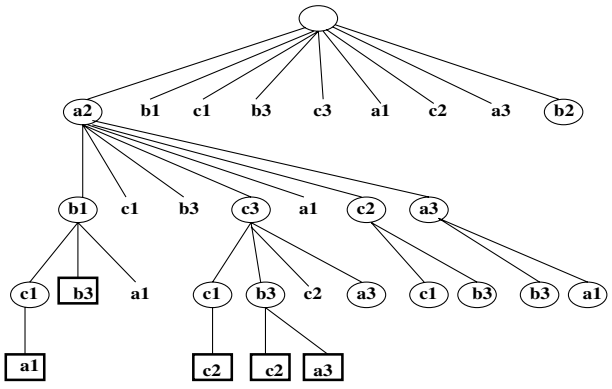


Fig. 5. Clique finding.

non-dense maximal clique; we give more details of this step in the next section. (2) There may be many overlapping cliques in $\mathscr{L}$. In this case, it is desirable to merge those cliques that have significant overlap into large cliques; we give details of this step below.

Note that overlapping cliques are mainly a result of the strict notion of strong connectedness for a cluster. For instance, consider a clique $C = C_1 \times \cdots \times C_k$, and consider a vertex v_m such that v_m is dense w.r.t. all subspaces except for one, say $C_j = \{v_1, \ldots, v_l\}$. Assume that v_m is dense w.r.t. all vertices in C_j except for v_a. In this case v_m cannot belong to the maximal clique C, but it may belong to another maximal clique C' that has a high degree of overlapping subspaces with C. If we detect such a case, it would be preferable to merge such cliques into a single cluster.

The enhanced post-processing step in CLICKS implements a novel method for merging a set of discovered maximal cliques based on their common *coverage*, i.e., the number of records that are common to that set of cliques. Let $\mathscr{L}$ be the set of maximal cliques mined from $\Gamma_{\mathscr{D}}$. For every clique $C^i \in \mathscr{L}$ let i denote its unique clique *id*. We define the term *cset* to denote any set of clique ids. Let $\mathscr{C}$ denote the database of csets obtained by replacing each record $\mathbf{r} \in \mathscr{D}$ with its cset, the set of clique ids that the record belongs to, given as $cset(\mathbf{r}) = \{i : \mathbf{r} \in C^i\}$. We can then mine the cset database $\mathscr{C}$ to obtain all the maximal frequent csets, denoted as $\mathscr{F}_{\mathscr{C}}$, that are above a minimum frequency threshold $\sigma^{\mathscr{C}}$, i.e., those csets that are co-supported by a minimum number of records. Note that $\mathscr{F}_{\mathscr{C}}$ can be efficiently mined using any maximal itemset mining method (such as GenMax [23]). For example, consider Fig. 1. Let $C^1 = \{a_2\} \times \{b_3\}$ and $C^2 = \{b_3\} \times \{c_3\}$ be the only maximal cliques in $\Gamma_{\mathscr{D}}$. Then the cset database $\mathscr{C}$ is given as: $\mathscr{C} = \{\{\}, \{1\}, \{1,2\}, \{\}, \{1,2\}, \{2\}, \ldots\}$. Mining $\mathscr{C}$ with minimum support $\sigma^{\mathscr{C}} = 2$ yields the maximal cset {1,2} suggesting that cliques C^1 and C^2 should be merged into one clique: $\{a_2\} \times \{b_3\} \times \{c_3\}$.

Every cset $X \in \mathscr{F}_{\mathscr{C}}$ is a potential set of cliques that can be merged. However $\mathscr{F}_{\mathscr{C}}$ may itself have overlapping maximal csets, and of various sizes. Clearly we need a ranking of csets so that merging can be done in a systematic manner. A good ranking measure is the coverage, i.e., the number of records in $\mathscr{D}$, that belong to the clique obtained after merging all cliques ids in a given cset. Unfortunately, computing the coverage for each $X \in \mathscr{F}_{\mathscr{C}}$ can be very expensive, since it would require multiple scans of the original database $\mathscr{D}$. Instead, we introduce an approximate measure of coverage, called *coverage weight*, that does not need to access the original database $\mathscr{D}$; it uses the clique support already computed from $\mathscr{D}$ in the validation step, and cset support computed while mining $\mathscr{F}_{\mathscr{C}}$. Intuitively, the coverage weight is an approximation of the inclusion/exclusion computation for the supporting records. More formally, given any $X \in \mathscr{F}_{\mathscr{C}}$, where $X = \{1, \ldots, m\}$ is a set of clique ids (corresponding to cliques $\{C^1, \ldots, C^m\}$) that frequently occur together, its *coverage weight* is defined as $\omega(X) = (\sum_{i=1}^{m} \sigma_{\mathscr{D}}(C^i)) - (m-1) \times \sigma_{\mathscr{C}}(X)$, where $\sigma_{\mathscr{C}}(X)$ denotes X's support within $\mathscr{C}$. For merging decisions, all csets in $\mathscr{F}_{\mathscr{C}}$ are sorted in decreasing order of their coverage weight.

Fig. 6 shows the pseudo-code for the post-processing phase, including the merging steps. After validating the set of mined maximal cliques $\mathscr{L}$, by counting their support and computing their density (line 1), we call selective vertical expansion if needed (line 2). This is followed by transforming the dataset $\mathscr{D}$ into the cset database $\mathscr{C}$, which is mined at minimum support $\sigma^{\mathscr{C}}$ to obtain all maximal frequent csets $\mathscr{F}_{\mathscr{C}}$, using GenMax [23] (line 3). This set is sorted in decreasing order of coverage weight to obtain a total order (denoted by the relationship $>_\omega$) on all maximal csets (line 4).

We then process each set $X \in \mathscr{F}_{\mathscr{C}}$ in order of $>_\omega$ (line 6); X is added to $\mathscr{F}^P$ (line 7) the set of processed csets, that give the clique ids of cliques to be merged in the end. Since no clique can be merged twice, all clique ids that occur in X have to be removed from the not-yet-processed csets $Y >_\omega X$ (lines 8–9).

Finally, we create the final set of merged clusters $\mathscr{L}$ by iterating through each cset $Z \in \mathscr{F}^P$ (line 12), and merging the cliques accordingly (line 13). Before merging, we make a copy of $\mathscr{L}$ in $\mathscr{L}'$ (line 10), so that we can merge cliques identified by Z by looking at $\mathscr{L}'$ (line 13), whereas $\mathscr{L}$ contains only the final set of cliques. If a merged clique M (line 14) is potentially dense we add it to the final set of clusters (line 15).

4.4. Selective vertical expansion

As noted earlier, all maximal cliques $\mathscr{L}$ are reported by the clique detection phase, but some might be pruned out in the post-processing if they are non-dense; let $\mathscr{L}^P \subseteq \mathscr{L}$ denote all such pruned cliques. Sub-cliques of a pruned clique, however, might have required density. In such cases, to guarantee completeness

PostProcess($\mathcal{D}$, $\mathcal{L}$, α, $\sigma^{\mathcal{C}}$)
1. Scan $\mathcal{D}$ and check density of each $C \in \mathcal{L}$
2. Perform Selective Vertical Expansion if required
3. $\mathcal{F}_{\mathcal{C}}$ = Maximal Frequent Csets in $\mathcal{C}$ (using $\sigma^{\mathcal{C}}$)
4. Sort $\mathcal{F}_{\mathcal{C}}$ by decreasing coverage weight ($>_{\omega}$)
5. $\mathcal{F}^P = \emptyset$
6. **for all** $X \in \mathcal{F}_{\mathcal{C}}$, **such that** $X \neq \emptyset$ **do**
7. $\quad \mathcal{F}^P = \mathcal{F}^P \cup \{X\}$
8. $\quad$ **for all** $Y \in \mathcal{F}_{\mathcal{C}}$, **such that** $Y >_{\omega} X$ **do**
9. $\quad\quad Y = Y \backslash X$ //remove cliques to be merged
10. $\mathcal{L}' = \mathcal{L}$ // Save the original cliques
11. $\mathcal{L} = \emptyset$
12. **for all** $Z = (z_1, z_2, \ldots, z_m) \in \mathcal{F}^P$ **do**
13. $\quad M = \bigcup_{i=1}^{m} C^{z_i}$, where $C^{z_i} \in \mathcal{L}'$
14. $\quad$ **if** $\omega(M) \geq E[\omega(M)]$ **then**
15. $\quad\quad \mathcal{L} = \mathcal{L} \cup \{M\}$

Fig. 6. CLICKS post-processing.

CLICKS uses the *selective vertical expansion* approach to consider all sub-cliques for each $C \in \mathscr{L}^P$, and reports all the remaining maximal, dense cliques.

For any vertex v in the k-partite graph $\Gamma_{\mathscr{D}}$ of a dataset $\mathscr{D}$, define its *ridset* to be the set of all record ids where v occurs, given as $\lambda(v) = \{\mathbf{r}.id : \mathbf{r} = (r.A_1, \ldots, r.A_n) \in \mathscr{D}, \text{ and } r.A_i = v\}$. Conceptually the ridset is a "vertical" encoding of the input dataset $\mathscr{D}$. Thus the name selective "vertical" expansion. The supporting ridset for any clique $C = C_1 \times \cdots \times C_k$, can then be defined as $\lambda(C) = \bigcap_{i \in [1,k]} \lambda(C_i)$, where $\lambda(C_i) = \bigcup_{v_j \in C_i} \lambda(v_j)$. For example, from the dataset $\mathscr{D}$ in Fig. 1, we have $\lambda(a_2) = \{2,3,4,5\}$. For $C = \{a_1, a_2\} \times \{b_1\} \times \{c_1, c_2\}$, we have $\lambda(C) = (\lambda(a_1) \cup \lambda(a_2)) \cap \lambda(b_1) \cup (\lambda(c_1) \cup \lambda(c_2)) = \{1,2,3,4,5\} \cap \{1,4\} \cap \{1,2,4\} = \{1,4\}$. Note that the cardinality of the ridset gives the support for the corresponding clique. For example, $\sigma_{\mathscr{D}}(\{a_2\}) = |\lambda(a_2)| = 4$ and $\sigma_{\mathscr{D}}(\{a_1, a_2\} \times \{b_1\} \times \{c_1, c_2\}) = 2$.

The selective vertical expansion step uses the ridset information to explore all sub-cliques for each $C \in \mathscr{L}^R$. Starting from the ridsets for the single values in C, we build larger sub-cliques using a series of union and intersection operations on the corresponding ridsets. The search stops when we have found all maximal dense sub-cliques contained within C. For each such sub-clique, if it is not contained within an already found maximal dense clique in $\mathscr{L} \setminus \mathscr{L}^P$, we output it as a true maximal, dense clique. As we shall see in our experiments (see Section 5.5), selective vertical expansion restores completeness for an extra, but reasonable cost.

Besides selective expansion, at least two other approaches for complete search can be explored: (1) count the support of each candidate clique in the dataset as it is being extended, or (2) use the ridsets to compute support figures incrementally during the clique mining phase (we call this the *full vertical expansion* approach). Clearly, option (1) is too expensive due to its numerous passes over the dataset.

For full expansion to be successful, the support information needs to be leveraged for pruning the search space depicted in Fig. 5 to offset the additional computation. Hence, the supporting ridsets need to be computed every time a clique is extended, and an appropriate pruning criterion has to be defined based on $\lambda(C)$ for the current clique C. Care must be taken not to cut potentially successful branches. Specifically, a branch cannot be pruned as soon as the support falls below α times the expected support. The reason is that neither density, nor clique support is monotonic along the search path. For example, in the graph in Fig. 2 and the associated dataset in Fig. 1, the clique $C' = \{a_2, c_3, b_3\}$ has support 2 ($\lambda(C') = \{3,5\}$). However, when extending the clique to the final clique $C = C' \cup \{a_3\} = \{a_2, c_3, b_3, a_3\}$ the support increases to 3 ($\lambda(C) = \{3,5,6\}$). The only observation that can be used for pruning is that there always exists a way of extending a clique candidate to a maximal clique such that the support along the way stays above zero. We have implemented the full expansion approach, but we found it to be too slow (by over an order of magnitude) to be competitive with selective expansion.

4.5. CLICKS complexity

Here we briefly look at the CPU and I/O complexity of CLICKS. In terms of I/O cost the base-line method makes only two full scans of database $\mathscr{D}$, once in the pre-processing step to build $\Gamma(\mathscr{D})$ and once in post-processing to validate the maximal cliques $\mathscr{L}$ against $\mathscr{D}$. For even large datasets $\Gamma(\mathscr{D})$ is expected to fit in memory, so the clique mining step incurs no I/O. While performing the second scan, we can also create the cset database $\mathscr{C}$ required for the merging step. Mining $\mathscr{C}$ requires one scan of $\mathscr{C}$, and the GenMax [23] uses efficient counting techniques to reduce further I/O. Finally, if the optional selective expansion step is called, then we have to make one scan of $\mathscr{D}$ to create the *ridsets* for each value. Of course, performing intersections and unions of ridsets to compute support of sub-cliques may incur additional I/O cost which is hard to characterize. Thus the CLICKS algorithm typically needs at least two (three for completeness) scans of the database $\mathscr{D}$ and one scan of $\mathscr{C}$.

The CPU cost of CLICKS is mainly contributed by the clique mining steps and the merging and selective expansion steps. Given n attributes with m values each, $\Gamma(\mathscr{D})$ can have at most (nm) vertices. In the worst case there can be $O(n^2m^2)$ edges, but such a dense graph is highly unlikely. $O(nm)$ edges is more realistic, since the k-partite summary graph is expected to be sparse. Mining all maximal cliques is clearly an exponential time algorithm in the worst case, since there can be an exponential number of cliques $|\mathscr{L}|$. Typically all maximal cliques can be found in time $O(nm|\mathscr{L}|)$. For selective expansion, in the worst-case, each clique $X \in \mathscr{L}$ might be non-dense, and we have to enumerate all of its sub-cliques. If the longest clique has l vertices in it, this step can take $O(2^l|\mathscr{L}|)$ time. Finally, the merging step is a function of the number of maximal csets, the number of records in $\mathscr{D}$ and the number of maximal cliques, giving a complexity of $O(n \times |\mathscr{L}| \times |\mathscr{F}_{\mathscr{C}}|)$.

4.6. Parameter selection

As described above, CLICKS output depends on two main parameters: α, the density threshold, and $\sigma^{\mathscr{C}}$, the clique merging threshold. Here we give guidance on how to select these parameter values.

First, the cluster merging threshold $\sigma^{\mathscr{C}}$ is not too hard to set. If $\sigma^C = 1.0$ (or 100%) no cliques will be merged, and all (overlapping) mined clusters will be reported. Starting with a high value, the user can systematically reduce σ^C so that they get a desired number of clusters. This selection process is similar to selecting k, the number of clusters, in k-means clustering, or selecting at what level to stop a hierarchical clustering algorithm.

Setting the density threshold α is more involved, since it depends on the data distribution. One of the features of the k-partite graph $\Gamma_{\mathscr{D}}$ is that the graph obtained for a high value of α is always a sub-graph of the one obtained for a lower value of α. This means that the clusters mined at higher α are contained in those clusters mined at a lower α. On the other hand, there is no data independent criteria for setting α. One approach that is likely to work is to examine the density of the k-partite graph by trying several values of α, ranging from small to large. If the graph is too dense, α should be increased, since in the worst case, the whole graph may become a complete k-partite graph containing a single cluster, which is not very informative. On the other hand, if the graph is too sparse, then again the clusters found will not be very interesting; in this case α should be decreased. Meaningful clusters will be found for α values that yield a medium range of graph density.

5. Experimental study

This section presents a comparative study of CLICKS vs. CACTUS [20] and other methods like ROCK [21] and STIRR [19]. All testing was done on a hyper-threaded Intel Xeon 2.8 GHz with 2GB of RAM, running the 2.4.22 SMP Linux kernel. All datasets were stored on an NFS mounted network drive on the same local 100MBit network. The code for CACTUS was obtained from its authors.

All synthetic datasets used in our experiments were created using the generation method proposed in [19]. The generator creates a user specified number of records that are uniformly distributed over the entire data space. It allows for specification of the number of attributes and the domain size on each attribute. The generator then injects a user specified number of additional records in designated cluster regions, thus increasing the support of these regions above their expected support.

In the performance studies below we use three attributes with domain size of 100 (unless specified otherwise), and we embed two clusters, located on the attribute values [0,9] and [10,19] for every attribute. Each cluster was created by adding an additional 5% of the original number of records in this subspace region. In all performance tests, $\kappa = 3$ (distinguishing number) and $\alpha = 3$ were chosen for CACTUS as suggested by Ganti et al. CLICKS was also configured to use $\alpha = 3$. We will show that compared to previous methods, CLICKS is orders of magnitude faster and/or delivers more intuitive and better clustering results.

5.1. CLICKS vs. ROCK

We first compare CLICKS with ROCK [21]. ROCK does not lend itself well to a direct comparison with CLICKS. Whereas CLICKS uses about 90% of its execution time for building in-memory representations of the attribute connectivities, the ROCK data format assumes that the similarities between data points are given. Despite this seeming advantage, the test series depicted in Fig. 7 shows that CLICKS still outperforms ROCK by orders of magnitude. Note that these are very small datasets with a maximum of 5500 tuples, thus it appears that the practical application of ROCK is limited to datasets of well below 10,000 records whereas CLICKS scales into the million record range. Since ROCK is too slow, we only compare CLICKS with STIRR and CACTUS below.

5.2. CACTUS extension

The available CACTUS implementation stops at the stage where it finds the potential cluster projections but does not extend these to produce the final (full- or sub-space) clusters. Note that the reported performance in [20] focuses only on I/O cost, and does not account for CPU cost of extension and validation, since they were mainly interested in showing the effectiveness of the small data summaries, even for very large datasets. To study the impact of these additional steps, we augmented the CACTUS implementation with the cluster extension and validation steps.

Fig. 8 shows the running time of CACTUS with and without the additional steps, on datasets with up to 500,000 tuples. We see that CACTUS with extensions is about 3 times slower than the base-line version, and the gap is increasing. This impact is largely due to the excessive number of projections that CACTUS generates. In the remaining performance studies only the base-line CACTUS version is used, since the version with extensions is too slow to be run on larger datasets.

5.3. CLICKS vs. CACTUS and STIRR: cluster quality

To evaluate the quality of the clusters, three basic scenarios were tested on synthetic datasets, with $\alpha = 3$, and with post-processing turned off, in order to verify the actual reported cliques before merging. The datasets, as shown in Fig. 9, contained 105,000 records in scenarios one and two. In Scenario 3, 110,000 records were used, reflecting additional 5000 records in the third cluster. In all scenarios, attributes have 100 values, but

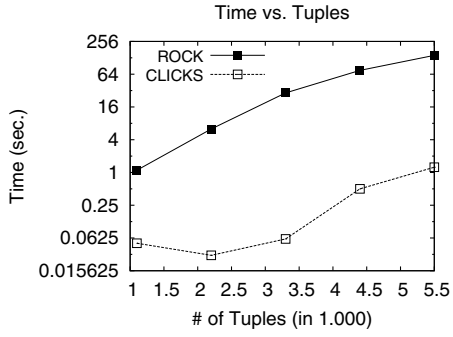


Fig. 7. CLICKS vs. ROCK.

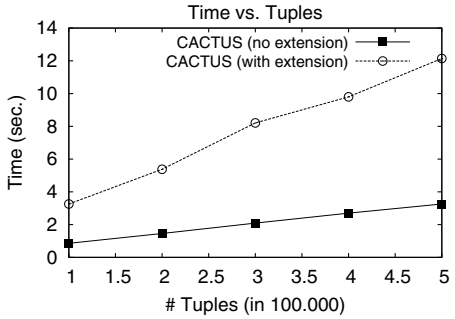


Fig. 8. Performance impact of extensions.

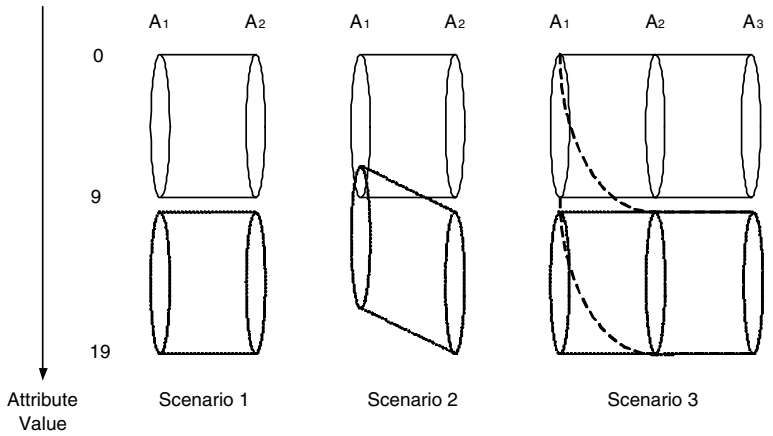


Fig. 9. Cluster quality comparison.

clusters are embedded only on the ranges drawn. For example, Scenario 1 has two attributes with two well separated clusters, one on attribute values [0–9] and another on [10–19] on A_1 and A_2; there are no clusters on attribute values [20–99] even though there are points in the dataset, chosen uniformly at random, in that range.

Scenario 1. Used a dataset with clear separation of the two clusters on ranges [0–9] and [10–19]. CLICKS detected both the clusters on the appropriate attribute values. The CACTUS implementation reported 240 cluster projections per attribute. These represented all subsets of size 3 of $\{0,\ldots,9\}$ and $\{10,\ldots,19\}$. They are part of the cluster projection but do not satisfy the maximality condition. Our CACTUS extension connected all subsets of $\{0,\ldots,9\}$ on the first attribute with the corresponding subsets on the second attribute. Similarly, all subsets of $\{10,\ldots,19\}$ were connected on both attributes, yielding 115,200 clusters (reflecting the lack of maximality of the projections). The STIRR algorithm reported weights of about 0.15 for the attribute values $[0,19]$ on both attributes, while the weights of the attribute values in $[20,99]$ were computed to be about 0.08. According to the interpretation in [19] this corresponds to a single cluster on $[0,19]\times[0,19]$, confirming the lack of separation found in [20].

Scenario 2. Used a dataset with a slight overlap between two clusters on one of the two attributes. CLICKS detected three initial cliques, two of which represented the original clusters and an additional clique on $[7,9]\times[0,19]$. Note that the third clique is correct according to Definition 1. However, the merge step in the post-processing step could optionally merge this third clique with one of the two primary cliques. CACTUS, on the other hand, reported 480 cluster projections, which were subsets of the three clusters that CLICKS reported. STIRR reported different weights for attribute values (i) outside the clusters, (ii) inside one cluster,

and (iii) inside both clusters. A non-trivial post-processing step external to STIRR could perhaps separate the attribute values based on these weights.

Scenario 3. Used a dataset with two clearly separated clusters and a third cluster that fully overlaps with the first cluster on attribute A_1, and with the second cluster on attributes A_2 and A_3. CLICKS reported two initial cliques on $[0,19]\times[10,19]\times[10,19]$ and $[0,9]\times[0,9]\times[0,9]$, respectively. These cliques were also the final clusters generated by CLICKS. This behavior is correct w.r.t. the cluster Definition 1, as $[10,19]\times[10,19]\times[10,19]$ is not maximal. CACTUS reported non-maximal subsets that yield about 312 million possible combinations. Verification of their correctness was not possible on our current machine due to the complexity of the extension operation. As in Scenario 1, STIRR reported weights of about 0.15 where a single cluster is present, 0.21 where clusters overlap, and 0.08 on all other attribute values. Again, it is not obvious how to extract actual clusters from these weights.

These results confirm that CLICKS is superior to both CACTUS and STIRR in detecting even the simplest of clusters!

5.4. CLICKS vs. CACTUS: performance

Three tests on synthetic datasets were performed to compare the performance of CLICKS and CACTUS w.r.t. number of records, number of attributes, and domain size, as shown in Fig. 10.

5.4.1. Dataset size

Synthetic datasets with 10 attributes, and 100 values per attribute were used, while the total number of records was varied from one to five million. Both methods scale linearly over the number of records in the dataset, but CLICKS outperforms CACTUS (base-line) by an average of 20%. If we take CACTUS with extensions into account (see Fig. 8) CLICKS is at least 3–4 times faster.

5.4.2. Dimensionality

CLICKS is especially scalable with regards to higher dimensional data. On a dataset with 1 million records and 100 attribute values per dimension, CLICKS outperforms CACTUS (base-line) by a factor 2–3 (and thus CACTUS (extension) by at least a factor of 6–9), when varying the number of attributes from 10 to 50, and the gap is increasing.

5.4.3. Domain size

Datasets with one million records and four attributes were used to measure the performance w.r.t. domain size. The number of attribute values per attribute were varied from 50 to 500. Both methods perform equally well for less than 400 attribute values per domain. At this point, the runtime of CACTUS dramatically increases, most likely due to memory shortage. For large domains, CLICKS is thus over an order of magnitude faster than CACTUS.

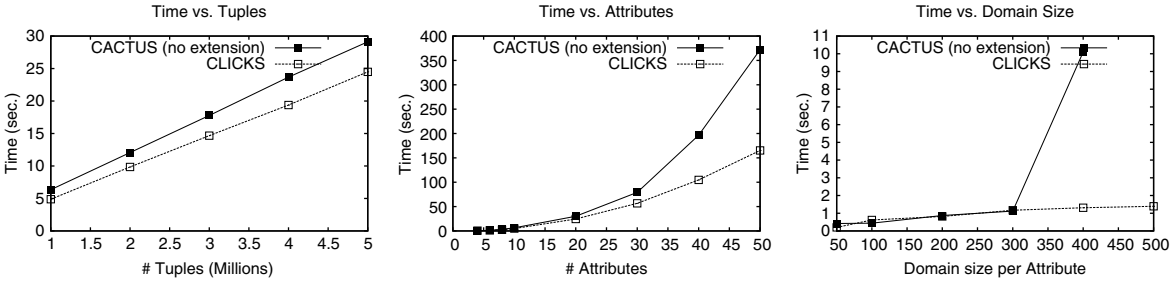


Fig. 10. CLICKS vs. CACTUS (no extensions).

The STIRR [19] algorithm, as implemented by Ganti et al. was also benchmarked. STIRR outputs the non-principal basins, i.e., weighted vertices, that identify the cluster projection on each attribute. As in the case of CACTUS, no clusters are actually output. However, it seems clear that the final cluster extraction step in STIRR would cost at least as much as the extension step in CACTUS.

5.5. Selective vertical expansion

CLICKS uses an optional selective vertical expansion method to mine the complete set of maximal, dense cliques. Clearly, this method cannot be faster than the baseline CLICKS version, as it adds additional post-processing. The question is whether the overhead is acceptable.

In Fig. 11 we plot the final number of reported clusters (after merging of overlapping clusters) in the selective vertical expansion and the base-line non-vertical case. We also plot the number of sub-cliques of non-dense cliques that are both maximal and dense, which we call "recovered cliques". Obviously, the number of recovered cliques are given before any merging is applied.

We ran CLICKS on synthetic datasets with 120,000 records, 4 attributes, and domain sizes varying from 100 to 180 values. Three fixed clusters (10,000 records each) were injected on attribute values $[0,4]$, $[3,7]$, and $[5,9]$. In all runs we used $\alpha = 1.6$, and $\sigma^{\mathscr{C}} = 0.005$ (threshold for merging cliques).

The first trend that can be observed in Fig. 11 is that there is little difference between the number of final clusters reported, after the merging step, for the complete selective expansion and the base-line CLICKS method. Only for large domain size (180) is there some difference. We also observe that whereas more clusters are recovered for sparser datasets (i.e., for larger domain sizes), a large fraction of those are merged into a larger cluster in post-processing. These results confirm that we get a good approximation of almost all clusters even without the selective vertical expansion.

In Fig. 12 we compare the selective vertical with baseline CLICKS in terms of running time for several varying parameters, namely number of records, dimensionality, and domain size. The figure shows that w.r.t. number of records, the baseline version starts about two times faster than the selective expansion method, but as

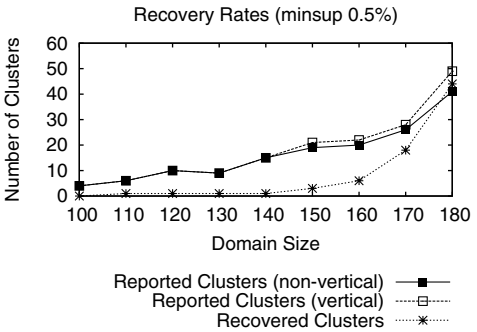


Fig. 11. Cluster recovery rates.

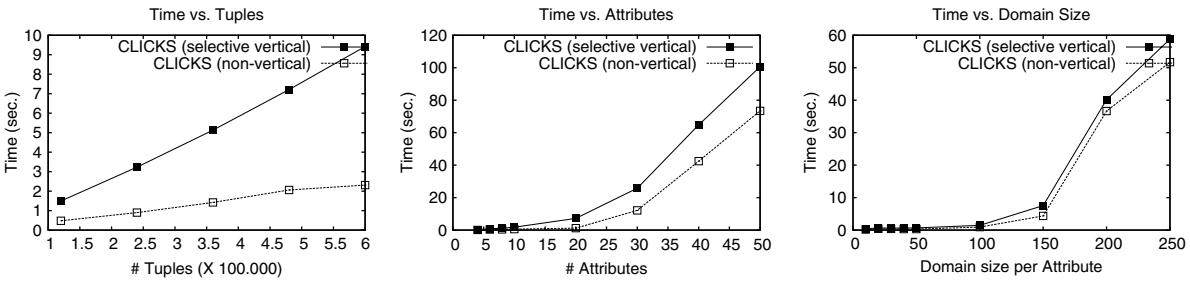


Fig. 12. Selective vertical mining.

the number of tuples increase six-fold, the gap increases to about 5 times (a factor of 2 increase). This overhead is mainly due to the building of ridsets in selective expansion step. The figure also shows that the selective vertical approach is quite resilient w.r.t. number of dimensions and domain size.

Overall, the selective vertical mining technique proves to be an efficient supplement to CLICKS. For scenarios where completeness of the clustering results is desired, it can find the erroneously pruned clusters at an acceptable cost (that is much lower than in full vertical mining; results not shown due to lack of space). The added overhead of selective mining is at most linear (in the number of records) w.r.t. the base-line, making it a computationally viable option even for large datasets. Most importantly, even *with* vertical mining enabled CLICKS is faster than other (incomplete!) methods on many datasets.

5.6. Post-processing performance

The CLICKS post-processing allows for merging "almost dense" cliques based on a user defined similarity threshold. Instead of looking at the merging behavior for synthetic datasets, the effectiveness and performance of the merging step were evaluated on the bibliographic dataset used in [20], which has 38,685 bibliographic records, with four attributes (1st author, 2nd author, publication venue, year). Because of the low number of attributes and the high density of the dataset, CLICKS was configured to use an α value of 75 yielding 258 initial cluster candidates.

Fig. 13 shows that when $\sigma^{\%}$ is increased from 0.1% to 30%, the number of final clusters also increases. This is intuitive, since the lower the merging threshold $\sigma^{\%}$ the more the clusters that are merged, resulting in fewer reported clusters. We also found that the merging time (results not shown) is not affected by the chosen $\sigma^{\%}$ value: on a moderately sized dataset such as the bibliographic dataset used for this experiment, only about 2% of the total execution time is spent validating and merging clusters.

5.7. Real datasets

Having established the superiority of CLICKS over previous methods like ROCK, STIRR and CACTUS in terms of time and/or quality, we now apply CLICKS on two real datasets.

5.7.1. Mushroom dataset

Part of the UCI Machine Learning Repository (http://www.ics.uci.edu/mlearn), it contains 8124 records and 22 categorical attributes. Each record describes one Mushroom specimen in terms of 22 physical properties (e.g., color, odor, and shape) and contains a label designating the specimen as either poisonous (3916 records) or edible (4208 records).

CLICKS was configured to run with a low α value of 0.4 as the dataset is very sparse. Not surprisingly, many of the candidate clusters were overlapping. By assigning each record to the first cluster that contains it, the confusion matrices shown in Figs. 14 and 15 were generated for full dimensional and subspace clustering, respectively. The two rows represent the two original classes, poisonous (P) and edible (E), while the columns represent the clusters that CLICKS generated. Each cell records the percentage of points that belong to that cell

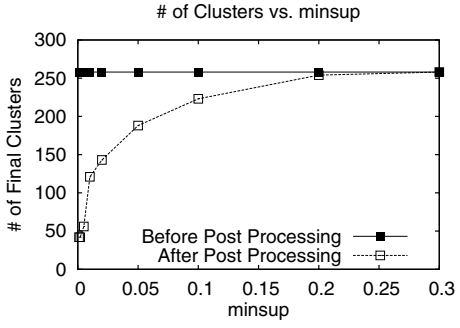


Fig. 13. Post-processing results.

	None	C_1	C_2	C_3	C_4	C_5
P	5.1	0.0	21.3	0.0	0.0	3.5
E	3.8	2.4	0.0	0.8	6.3	0.0
	C_6	C_7	C_8	C_9	C_{10}	C_{11}
P	0.0	0.0	0.0	0.0	0.0	2.4
E	9.5	0.6	1.6	1.8	17.9	0.0
	C_{12}	C_{13}	C_{14}	Others		
P	0.0	0.0	16.0	0.0		
E	2.4	0.6	0.0	4.0		

Fig. 14. Mushroom: confusion matrix (full space).

	None	C_1	C_2	C_3	C_4	C_5	C_6
P	0.0	0.0	21.3	0.0	0.0	3.5	0.0
E	0.0	2.4	0.0	0.8	6.3	0.0	9.5
	C_7	C_8	C_9	C_{10}	C_{11}	C_{12}	C_{13}
P	0.0	0.0	0.0	0.0	2.4	0.0	0.0
E	0.6	1.6	1.8	17.9	0.0	2.4	0.6
	C_{14}	C_{15}	C_{16}	C_{17}	C_{18}	C_{19}	
P	16.0	0.4	2.6	0.3	0.1	1.7	
E	0.0	0.0	0.0	0.2	7.6	0.0	

Fig. 15. Mushroom: confusion matrix (subspace).

(e.g., in Fig. 14, 21.3% of all points belong to cluster C_2 and have the label 'P'). Note, that the class attribute was not used in the clustering.

Full-dimensional clustering initially yielded 256 candidate clusters which were then reduced to 213 clusters using a $\sigma^{\mathscr{C}}$ value of 0.5% for the post-processing step. Out of the 213 total clusters, 14 of them account for 87.1% of all points; these clusters with highest support values are shown explicitly (C_1 to C_{14}) in Fig. 14, while the other smaller clusters are grouped under the column *Others*. Note that these smaller clusters account for only 4% of all points, and thus one can safely discard these clusters to yield 14 useful full-dimensional clusters which show perfect purity w.r.t. the class label. About 9% of the records could not be grouped (column *None*) in any cluster.

For the subspace case CLICKS produced 596 initial clusters (including full-space clusters). This number was reduced to 553 by merging with a $\sigma^{\mathscr{C}}$ setting of 5%. As in the full dimensional case, a large number of clusters overlapped. By assigning each record to the first cluster that contains it, Fig. 15 was obtained. All points can be covered by only 19 clusters, of which C_1–C_{14} are full-dimensional (same as before), and there are five new subspace clusters C_{15}–C_{19}. The subspace clusters clearly improved the result w.r.t. the unclustered records, since all records are now covered by some cluster (the *None* column has 0% of points, as opposed to 8.9%

	None	C_1	C_2	C_3	C_4	C_5	C_6	C_7	C_8
R	1.9%	0.0%	0.2%	0.0%	36.5%	0.0%	0.0%	0.0%	0.0%
D	0.6%	4.4%	0.5%	0.7%	3.2%	4.4%	45.6%	1.8%	0.2%

Fig. 16. Confusion matrix votes (full Space).

	None	C_1	C_2	C_3	C_4	C_5	C_6	C_7	C_8
R	0.5%	0.0%	0.2%	0.0%	36.5%	0.0%	0.0%	0.0%	0.0%
D	0.4%	4.4%	0.5%	0.7%	3.2%	4.4%	45.6%	1.8%	0.2%

	C_9	C_{10}	C_{11}	C_{12}
R	0.5%	0.5%	0.2%	0.2%
D	0.2%	0.0%	0.0%	0.0%

Fig. 17. Confusion matrix votes (subspace).

in the full dimensional case). Cluster C_{17} and C_{18} show minor impurities (<1%) w.r.t. the class label. By using all 553 clusters a perfectly pure clustering is obtained. However, this level of granularity will be inappropriate for most applications.

5.7.2. Congressional votes dataset

Also part of the UCI repository, it contains 435 records indicating the votes of Congressmen in 16 different polls in 1984. Each record is labeled as Republican (168 records) or Democratic (267 records). The individual attributes are Boolean valued (yes or no vote). The dataset is relatively sparse; an α value of 0.1 was used.

Fig. 16 shows the results for a full dimensional clustering with CLICKS. The rows indicate the two original classes (Republican or Democrat). For full dimensional clustering, the post-processing step proved to be especially useful, as it reduced the original 51 candidate clusters down to 13 at a $\sigma^{\mathscr{C}}$ level of 5%. Of these, 8 final clusters contained almost 98% of all records. Only 2.5% of the voting behaviors could not be clustered using this approach.

With subspace clustering, of the 68 candidates, 38 were obtained after merging. The rate of unclustered records was reduced to about 1% while increasing the number of relevant clusters to 12 (Fig. 17). Interestingly, as in the case of mushroom, the subspace clustering preserves all full dimensional clusters and adds four new subspace clusters that capture previously unclustered voting behavior. This intuitively models the fact that there are strong Democratic and Republican positions but that some Congressmen may not be in line with the party's overall policy on individual issues. Thus the algorithm can capture "non-standard" voting behaviors.

6. Conclusions and future work

CLICKS uses a novel formulation of categorical subspace clusters, based on the notion of mining cliques in a k-partite graph. It implements an efficient algorithm to mine k-partite maximal cliques, which correspond to the clusters. Using a novel vertical encoding we can guarantee the completeness of the results at a reasonable additional cost without sacrificing scalability. CLICKS imposes no domain constraint, is scalable to high dimensions, mines subspace and full-dimensional clusters, and outperforms existing approaches by over an order of

magnitude. In the future, the k-partite model will serve as a foundation for applying other graph based methods to data mining problems on categorical datasets.

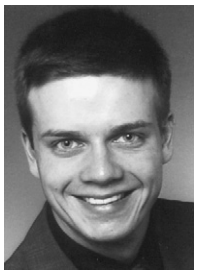
Acknowledgment

We would like to thank Venkatesh Ganti for providing the source code for CACTUS and STIRR, and Eui-Hong "Sam" Han for the ROCK implementation. We would also like to thank Xiuhong "Cheryl" Hu, who worked on an early implementation of CLICKS.

References

[1] J. Han, M. Kamber, Data Mining: Concepts and Techniques, Morgan Kaufmann Publishers, San Francisco, CA, 2001.
[2] S.C. Madeira, A.L. Oliveira, Biclustering algorithms for biological data analysis: a survey, IEEE/ACM Transactions on Computational Biology and Bioinformatics 1 (1) (2004) 24–45.
[3] R. Agrawal, J. Gehrke, D. Gunopulos, P. Raghavan, Automatic subspace clustering of high dimensional data for data mining applications, in: SIGMOD Conf., 1997.
[4] H. Nagesh, S. Goil, A. Choudhary, Adaptive grids for clustering massive data sets, in: SIAM Data Mining Conf., 2002.
[5] K. Sequeira, M. Zaki, SCHISM: a new approach for interesting subspace mining, in: IEEE Int'l Conf. on Data Mining, 2004.
[6] C.M. Procopiuc, M. Jones, P.K. Aggarwal, T.M. Murali, A Monte Carlo algorithm for fast projective clustering, in: SIGMOD Conf., 2002.
[7] C.C. Aggarwal, P.S. Yu, Finding generalized projected clusters in high dimensional spaces, in: SIGMOD Conf., 2000.
[8] K. Kailing, H. Kriegel, P. Kroeger, S. Wanka, Ranking interesting subspaces for clustering high dimensional data, in: PKDD Conf., 2003.
[9] Y. Cheng, G. Chruch, Biclustering of expression data, in: ISMB Conf., 2000.
[10] M. Ester, H. Kriegel, J. Sander, X. Xu, A density-based algorithm for discovering clusters in large spatial databases with noise, in: SIGKDD Conf., 1996.
[11] C. Ordonez, Clustering binary data streams with k-means, in: SIGMOD DMKD Workshop, 2003.
[12] K. Wang, C. Xu, B. Liu, Clustering transactions using large items, in: CIKM Conf., 1999.
[13] M. Koyutürk, A. Grama, PROXIMUS: a framework for analyzing very high-dimensional discrete attributed datasets, in: SIGKDD Conf., 2003.
[14] E. Han, G. Karypis, V. Kumar, B. Mobasher, Clustering based on association rule hypergraphs, in: SIGMOD DMKD Workshop, 1997.
[15] Z. Huang, A fast clustering algorithm to cluster very large categorical data sets in data mining, in: SIGMOD DMKD Workshop, 1997.
[16] D. Barbara, Y. Li, J. Couto, Coolcat: an entropy-based algorithm for categorical clustering, in: CIKM Conf., 2002.
[17] D. Cristofor, D. Simovici, An information-theoretical approach to clustering categorical databases using genetic algorithms, in: SIAM Workshop on clustering high dimensional data, 2002.
[18] P. Andritsos, P. Tsaparas, R.J. Miller, K.C. Sevcik, LIMBO: scalable clustering of categorical data, in: 9th Int'l Conf. on Extending DataBase Technology, 2004.
[19] D. Gibson, J. Kleinberg, P. Raghavan, Clustering categorical data: an approach based on dynamical systems, in: VLDB Conf., 1998.
[20] V. Ganti, J. Gehrke, R. Ramakrishnan, CACTUS: clustering categorical data using summaries, in: SIGKDD Conf., 1999.
[21] S. Guha, R. Rastogi, K. Shim, Rock: a robust clustering algorithm for categorical attributes, in: ICDE Conf., 1999.
[22] H. Johnston, Cliques of a graph—variations on the Bron–Kerbosch algorithm, International Journal of Computer and Information Sciences 5 (3) (1976) 209–238.
[23] K. Gouda, M.J. Zaki, Efficiently mining maximal frequent itemsets, in: ICDM Conf., 2001.

Mohammed J. Zaki is an Associate Professor of Computer Science at RPI. He received his Ph.D. degree in computer science from the University of Rochester in 1998. His research interests focus on developing novel data mining techniques for bioinformatics, and other applications. He has published over 100 papers on data mining and co-edited 11 books. He is currently an associate editor for IEEE Transactions on Knowledge and Data Engineering, action editor for Data Mining and Knowledge Discovery, and on the editorial board of Int'l Journal of Data Warehousing and Mining, Int'l Journal of Data Mining and Bioinformatics, Scientific Programming and the ACM SIGMOD Digital Symposium Collection. He received the NSF Career Award in 2001 and the DOE Career Award in 2002. He also received a recognition of service award from ACM in 2003, and a certificate of appreciation from IEEE in 2005.



Markus Peters is a consultant for Deloitte & Touche's Business Intelligence service line, Germany. He specializes in the design and implementation of enterprise-scale BI applications with a focus on reporting and multi-dimensional analysis. Markus received a MS degree in Information Technology from Rensselaer Polytechnic Institute and MS degrees in Computer Science and Business from RWTH Aachen, Germany in 2004.

Ira Assent is currently a research associate at RWTH Aachen. She received her Dipl.-Inform. from RWTH. Her research interests lie in the area of efficient similarity search in large multimedia databases.

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 Universitat Munchen, Seidl received his Ph.D. in 1997 and his venia legendi in 2001 from the University of Munich, Germany.