CLICK: Clustering Categorical Data using K-partite Maximal Cliques

Markus Peters and Mohammed J. Zaki
Computer Science Department
Rensselaer Polytechnic Institute
Troy NY 12180

Abstract

Clustering is one of the central data mining problems and numerous approaches have been proposed in this field. However, few of these methods focus on categorical data. The categorical techniques that do exist have significant shortcomings in terms of performance, the clusters they detect, and their ability to locate clusters in subspaces.

This work introduces a novel algorithm called CLICK, which finds clusters in categorical datasets based on a search method for k-partite maximal cliques. CLICK is able to detect subspace clusters, and outperforms previous approaches by a factor of two to three. It scales better than any of the existing method for high dimensional datasets. These results are demonstrated in a comprehensive performance study on synthetic and real data sets.

1 Introduction

Clustering is one of the central data mining problems [17, 18]. Generally, the goal in clustering is, given a dataset, to find "naturally" occurring groups within the dataset, or regions in the space generated by the dataset where the density of data points is higher than "normally" expected. Figure 1 shows an example of a two dimensional dataset. One can intuitively identify two dense regions in the figure that should make up clusters.

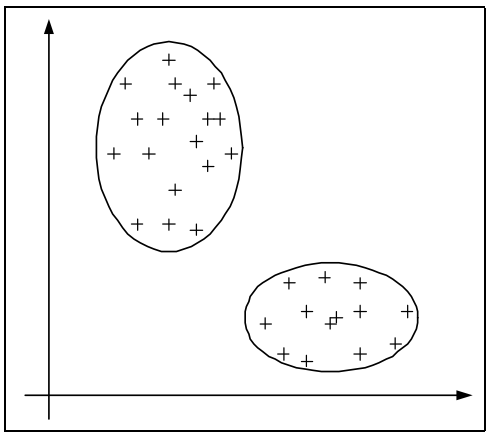

Figure 1: Clustering Example

It is natural to approach this problem by computing the similarities between individual data points in the dataset and partitioning the data points in a way that yields groups where the similarity of points within each partition is as high as possible, while points in different partitions are as dissimilar as possible.

Problems arise when it comes to clustering *categorical* data, i.e. data where the domains of the individual attributes are discrete valued and not naturally ordered. Roughly, the challenges in clustering categorical attributes can be grouped into three categories:

No Natural Order The lack of an inherent "natural" order on the individual domains. This property renders a large number of traditional similarity measures ineffective.

High Dimensionality Categorical datasets are frequently high dimensional.

Subspace Clusters Many categorical datasets do not exhibit clusters over the full set of dimensions.

The remainder of this chapter will give an introduction to the overall problem of clustering, the specifics of categorical clustering, and then continue to discuss each of the above problems in more detail.

1.1 Clustering in Data Mining

To formalize the task of clustering a dataset consider the following definition of a dataset.

Definition 1.1 (Dataset) *Let $A_1, \ldots, A_n$ be a set of attributes and $D_1, \ldots, D_n$ non-empty sets over these attributes where $D_i \cap D_j = \emptyset$ for $i \neq j$.*

A set $\mathcal{D} \subseteq D_1 \times \ldots \times D_n$ is called a dataset *over the* domains $D_1, \ldots, D_n$*. An element $(r_1, \ldots, r_n) \in \mathcal{D}$ is called a* record *or* feature vector*. Each $r_i, i \in \{1, \ldots, n\}$ is a* field or *attribute of its record. The notation $r.A_i$ is used to refer to the i-th component of the feature vector r. The number n of attributes is also referred to as the* dimensionality *of the dataset.*

The disjointness constraint in definition 1.1 is exclusively formal. Without loss of generality, disjointness can be ensured on an arbitrary dataset by mapping value v_j of attribute A_i to the unique value (A_i, v_j) of a surrogate attribute A'_i.

Note, that – other than non-emptiness – there is no restriction on the domains underlying the individual attributes. However, depending on the properties of a concrete domain, it is classified as either *categorical* or *non-categorical*, where non-categorical domains are inherently ordered, and categorical domains are not. Numerical attributes are typically of the non-categorical type. When a dataset consists solely of categorical attributes it is said to be categorical itself. This type of dataset will be the focus of the present work.

Given a dataset, clustering can be understood as the optimization problem of partitioning the dataset into groups, the elements of which are as similar as possible to elements of the same group and as different as possible from elements of other groups.

Definition 1.2 (Clustering) *Let $\mathcal{D}$ be a dataset and $d : \mathcal{D} \times \mathcal{D} \to \mathbb{R}^+$ be a measure of distance (or dissimilarity) between feature vectors.* Clustering *is the task of finding a partition $(C_1, \ldots, C_n)$ of $\mathcal{D}$ such that*

$$\forall i, j \in \{1, \ldots, n\}, j \neq i, \forall x \in C_i : d(x, M_i) \leq d(x, M_j)$$

where M_i is one cluster representative of *cluster C_i.*

This definition leaves some decisive points unclear, such as the distance measure and the cluster representative to use, or how to obtain the optimal number of clusters. Clearly, partitioning the dataset into clusters that contain one record each would solve the above problem in a trivial way. [18] addresses these issues in a general setting, while section 1.2 and chapter 3 more specifically address these problems for categorical clustering.

1.2 Categorical Clustering

The categorical clustering problem can be formulated as follows [12]: Let $\mathcal{D}$ be a dataset over $A_1, \ldots, A_n$, and $D_1, \ldots, D_n$ the associated categorical domains

Definition 1.3 (Interval) *A set $S_i \subseteq D_i$ is called an* interval *over attribute A_i. A k-*interval *is a set $S = S_{i_1} \times \ldots \times S_{i_k}$ over a subset of k attributes $A_{i_1}, \ldots, A_{i_k}$. If $k < n$, then S is also called a* subspace interval*, otherwise S is an interval over the whole space of n attributes.*

Clusters can informally be understood as especially dense interval regions within the dataset. To capture the density notion, the *support* of such a region needs to be defined. Especially dense regions can then be identified by capturing the expected support of a given interval region and comparing it to the support the region actually has within the dataset. If the actual support is higher than the expected support – possibly by a user defined factor – the region can be considered dense.

Definition 1.4 (Support) *Let S be a k-interval, with $k \leq n$. A record $\mathbf{r} = (r.A_1, \ldots, r.A_n) \in \mathcal{D}$,* belongs *to S, denoted $\mathbf{r} \in S$, iff [1] $r.A_{i_j} \in S_{i_j}$ for all $j \in \{1, \ldots, k\}$. The* support *of S in dataset $\mathcal{D}$ is given as*

$$\sigma(S) = |\{\mathbf{r} \in \mathcal{D} : \mathbf{r} \in S\}|$$

Assuming attribute independence, the expected support *of S in $\mathcal{D}$ is given as*

$$E[\sigma(S)] = |\mathcal{D}| \cdot \prod_{i=1}^{n} \frac{|S_i|}{|D_i|}$$

The expected support notion can be used to formulate a co-occurrence measure that denotes how strongly attribute values and interval regions interact with each other.

Definition 1.5 (Strongly Connected Intervals) *Let $S = S_{i_1} \times \ldots \times S_{i_k}$ be a k-interval. Define an indicator function $\sigma^*(S)$ as follows*

$$\sigma^*(S) = \begin{cases} 1 & \text{if } \sigma(S) > \alpha \cdot E[\sigma(S)] \\ 0 & \text{otherwise} \end{cases}$$

where $\alpha \in \mathbb{R}^{>0}$ is a user defined real number. For any pair of values $v_i \in S_i$ and $v_j \in S_j$, we say that v_i and v_j are strongly connected *iff $\sigma^*(\{v_i\} \times \{v_j\}) = 1$, i.e., the 2-interval $\{v_i\} \times \{v_j\}$ has support at least α times its expected support. We say that v_i is strongly connected to interval S_j iff $\sigma^*(\{v_i\} \times \{v_j\}) = 1$ for all $v_j \in S_j$, and we say that the intervals S_i and S_j are strongly connected iff $\sigma^*(\{v_i\} \times \{v_j\}) = 1$ for all $v_i \in S_i, v_j \in S_j$.*

Definition 1.6 (Cluster) *Let $C_i \subseteq D_i$ for $i \in \{1, \ldots, n\}$, and $\alpha > 0$. The k-interval $C = (C_{i_1} \times \ldots \times C_{i_k})$ is a* (subspace) cluster *over attributes $A_{i_1}, \ldots, A_{i_k}$ iff*

1. *$\sigma^*(C) = 1$, i.e., the support of the cluster is at least α times its expectation.*
2. *All pairs of intervals $C_i, C_j \in C$ $(i \neq j)$ are strongly connected.*
3. *$\nexists C_i' \supset C_i$, for all $i, j \in \{1, \ldots, n\}, i \neq j$, such that C_i and C_j are strongly connected, i.e. C_i and C_j are* maximal *strongly connected intervals.*

The interval C_i is also called the cluster projection *of C on attribute A_i. If $k < n$, then C is called a* subspace cluster *or a k-*cluster*, otherwise C is called a* cluster*.*

Given a dataset $\mathcal{D}$ and a user specified threshold α, the goal is to find all clusters and also all subspace clusters if desired.

[1] if and only if

1.3 Challenges in Categorical Clustering

Categorical datasets impose a number of challenges on clustering methods, the most significant of which is the lack of a natural order on the individual domains. This property effectively renders a large number of traditional similarity measures obsolete. Replacements have been proposed that do not take advantage of the order or, even more restrictive, numerical operations. Generally, these measures are based on co-occurrence of attribute values. They may require similarity to be defined even between attribute values that never occur together in one record. Examples include the Simple Matching Coefficient, and the Jaccard Coefficient [18].

Secondly, categorical datasets are frequently high dimensional. Clearly, high dimensionality is not an immediate consequence of these datasets. However, practical examples suggest that clustering approaches for categorical data should be highly scalable in terms of number of attributes. In high-dimensions, it can be shown that traditional distance measures become ineffective, a phenomenon known as the *curse of dimensionality* [6, 27].

Finally, many categorical datasets do not exhibit clusters over all dimensions. This is especially true for sparse spaces, e.g. in document clustering, where the dictionary can be very large but individual documents contain relatively few words. Thus it may be desirable to identify clusters in subspaces.

The above discussion entails a number of key characteristics for good categorical clustering algorithms, which should ideally

- not impose any constraints or assumptions on the underlying domain,
- scale well over the number of attributes, and
- detect clusters not only over all attributes, but also over subsets thereof.

Chapter 2 presents an analysis of existing methods, and highlights areas of improvement. Chapter 3 introduces CLICK [2], which finds clusters in categorical datasets based on a search method for k-partite maximal cliques. CLICK helps address the main shortcomings of existing approaches. It detects *subspace clusters*, and outperforms previous approaches by a factor of two to three. It scales better than the existing method for high dimensional datasets. The performance characteristics of this method are studied in chapter 4 in a comprehensive performance evaluation on real and synthetic datasets. Finally, chapter 5 contains a discussion, conclusions, and pointers to future work.

2 Related Work

Two areas of previous work are relevant in the context of categorical subspace clustering. Section 2.1 discusses earlier methods for clustering categorical data. As most of these techniques do not consider subspaces, section 2.2 presents an extended review of general subspace clustering techniques and their relevance with respect to categorical data.

2.1 Categorical Clustering Techniques

While a lot of work has focused on clustering of numeric data [17], only a limited number of studies have focused on categorical clustering; these include STIRR [13], ROCK [15], CACTUS [12], COOLCAT [5], k-modes [19], and others more [8, 37]. Other works have focused more narrowly on binary or transactional data [25, 32], on a framework to compress high dimensional categorical datasets [22], and on using hypergraph partitioning to cluster itemsets [16].

The COOLCAT algorithm introduced by Barbara et al. [5] is based on the idea of entropy reduction within the generated clusters. It first *bootstraps* itself using a sample of maximally dissimilar points from the dataset to create initial clusters. The remaining points are then added incrementally. Naturally, this approach is highly dependent on the order of selection. To mitigate this dependency, the authors propose to remove the "worst fitting" points at defined times during the execution and re-clustering them.

[2] **CL**uster**I**ng **C**ategorical data via maximal **K**-partite cliques

Cristofor et al.[8] present another approach based on cluster entropy measures for categorical attributes. Starting from a seed clustering, it uses genetic algorithms with crossover and mutation operators 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.

Huang introduces k-modes [19], an extension to the well-known k-means algorithm for clustering numerical data. By defining the mode notion for categorical clusters, and introducing an incremental update rule for cluster modes, the algorithm preserves the scaling properties of k-means. Naturally, it also inherits its disadvantages, such as dependence on the seed clusters, and the inability to automatically detect the number of clusters.

STIRR was presented by Gibson et al. in [13]. The method encodes datasets into a weighted graph structure where the individual attribute values correspond to weighted vertices. STIRR iterates multiple instances (so-called *basins*) of these graphs using a user defined combination operator to eventually converge to a fix point. The authors argue that upon reaching this fix point, the weights of the *basins* can be used to partition the data points, yielding the final clusters. The dynamical systems approach underlying STIRR is problematic with regards to the type of detected clusters; the separation of attribute values by their weights is non-intuitive. Moreover, the number of basins required to attain a sufficiently large probability of convergence can be significant.

Zhang et al.[37] point out that the lack of a definite convergence is one of STIRR's shortcomings and propose a similar method that is guaranteed to converge. However, for both methods, the combination operator, as well as local modification operations are left to the user to find depending on the concrete data. Finally, the post-processing required to generate the actual clusters from the basin weights upon reaching the fix point is non-trivial and impacts the detected clusters. The clusters identified by STIRR were shown to be incomplete in cases of overlapping cluster projections [12].

Guha et al. present ROCK[15], a clustering algorithm based on the number of *links* between tuples. The number of links intuitively captures the number of records that two records are both sufficiently similar to. This approach yields satisfactory results with respect to comparing attribute values that never co-occur in a single tuple. ROCK heuristically optimizes a cluster quality function with respect to the number of links in an agglomerative hierarchical fashion. The base algorithm exhibits cubic complexity in the number of records, which makes it unsuitable for large datasets. Guha et al. propose a sampling approach to this end.

Ganti et al. introduce CACTUS [12], a combinatorial search based algorithm utilizing summary information of the dataset. Unlike earlier algorithms it characterizes the detected categorical clusters. The algorithm relies on inter- and intra-attribute summaries that are assumed to fit into main memory for most categorical datasets. CACTUS 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 that represents the minimum size of the *distinguishing sets* which are attribute value sets that uniquely occur within only one cluster. The distinguishing sets are then extended to cluster projections. Finally, cluster projections can be combined to clusters candidates over multiple attributes which are validated against the original dataset.

The distinguishing sets in CACTUS rely on the assumption that clusters are uniquely identified by a core set of attribute values that occur in no other cluster. While this assumption may hold true for many real-world datasets, it is unnatural and unnecessary for the clustering process, as shall be shown later. Moreover, it is desirable to choose κ as low as computationally possible in order to detect all clusters. A small κ, however, entails a large number of candidate cluster projections on the individual attributes that lead to a combinatorial explosion in the number of final clusters.

The cluster projections on single attributes that CACTUS generates are used in its *extension* phase to generate cluster candidates of higher dimensionality that are then validated on the actual dataset. The proposed approach to this end selects as initial one dimensional candidates C^1 all cluster projections c_1 on the first attribute. Candidates in subsequent C^{k+1} are generated by combining each $(c_1, \ldots, c_k) \in C^k$ with all cluster projections c_{k+1} on attribute A_{k+1}. If for all $1 \leq i \leq k$, (c_i, c_{k+1}) is a cluster projection on (A_i, A_{k+1}), $(c_1, \ldots, c_{k+1})$ is added to the candidate set C^{k+1}. Clearly, the candidates have to be validated by scanning the original dataset and counting the support of each candidate.

The available CACTUS implementation from its authors does not include this extension step, and it is

unclear whether or not the reported performance [12] accounts for extension and validation. A study of the extension and validation phase shows a significant performance impact (see figure 9). Finally, the proposed extension does not discover subspace clusters, other than those in the subspaces

$$(A_1, A_2), (A_1, A_2, A_3), \ldots, (A_1, \ldots, A_n)$$

The authors propose to apply the MDL pruning approach used in [3] for subspace clustering, but it was never implemented.

2.2 Subspace Clustering Work

Subspace clustering has been explored more extensively in the context of numerical data. Relevant approaches in this field include CLIQUE [3], MAFIA [23], PROCLUS [1], ORCLUS [2], and others more [21, 26, 35]. The following presentation is aimed at the subspace functionalities of the respective methods; strengths and weaknesses of the clustering methods itself are not discussed in detail. After a brief summary of the individual methods, their applicability to categorical subspaces is presented.

CLIQUE was proposed in [3]. It is a grid-based approach that works on ξ equal-width intervals in each dimension, where ξ is user defined. The cross-product of one of these intervals per dimension is referred to as a *unit* for a given set of dimensions. A unit is considered *dense* if its support is above a user specified level τ. Starting from one dimensional dense units D_1, CLIQUE generates the higher dimensional dense units (i.e. candidate clusters) in an a-priori manner [4]. To compute D_k, the algorithm self-joins D_{k-1} for units that share the first $k-2$ dimensions. Those elements of C_k that have a $k-1$ dimensional projection not included in C_{k-1} are subsequently pruned. To further reduce the computational cost of the candidate construction, the authors propose to prune candidates with low *coverage* where the cut-off point is determined by using the Minimum Description Length (MDL) principle. Clearly, completeness of the method is lost in this step. The final clusters are then generated by finding maximally connected sets of dense units, where connections refer to shared faces in hyperspace. DNF expressions with disjunctions of disjoint intervals on the same attribute and conjunctions over different attributes are used to describe the resulting clusters.

MAFIA [23] improves CLIQUE by introducing the notion of *adaptive grids*. Adaptive grids mitigate one of the core problems of grid-based approach: the trade-off between computationally intensive fine grids, and imprecise coarse grids. An initially fine-grained histogram is used to merge bins in regions that have a density below average. The results is a variable-size grid structure with finer resolution in regions with higher density, i.e. regions that are more interesting. The bottom-up approach used by CLIQUE for candidate generation is duplicated with the exception that *any* $k-2$ matching dimensions will suffice for two $k-1$ dimensional candidates to be joined into a k dimensional candidate, not necessarily the *first* $k-2$ dimensions. Note, that the cluster notion of both, CLIQUE and MAFIA, is different from that used by the algorithms discussed below: CLIQUE and MAFIA computer overlapping clusters while the remaining methods are targeted at computing a partition of the data set.

Aggarwal et al. introduce PROCLUS [2], a projection-based clustering algorithm. PROCLUS creates subspace clusters by considering for every cluster C a subspace which yields the "best" cluster for the associated axes-parallel projection of C. The cluster computation itself is based on a hill-climbing technique similar to the CLARANS local search approach [24]. An initial set of potential cluster medoids $\mathcal{M}$ is chosen based on a modified greedy approach. For every iteration the algorithm then determines the "best" dimensions of subspaces associated with each medoid in the set of current medoids $M \subset \mathcal{M}$. Given a maximal spherical neighborhood $\mathcal{L}_i$ of a medoid m_i that does not contain any other $m_j \in \mathcal{M}$, the average distance $X_{i,j}$ of a point in $\mathcal{L}_i$ to m_i along dimension j is determined. The "best" dimensions are selected as the smallest $X_{i,j}$, reflecting to the idea that along a relevant dimension of a medoid m_i points should be close to that medoid.

ORCLUS [2] improves PROCLUS by including non-axes-parallel projections. In an effort to decrease the *energy* (i.e. the sum of the error squares) of the cluster projections, the algorithm uses the eigenvectors e_i of the covariance matrix of points in a cluster C corresponding to the lowest eigenvalues λ_i. These eigenvalues λ_i correspond to to the variance along the direction e_i. In this sense, the ORCLUS projections can be

understood as an inverted Principal Component Analysis, where the goal is to find projection directions with minimum variance.

The RIS algorithm presented in [21] ranks "interesting" subspaces of a data set which can then be used to pre-process data for other clustering techniques. The method is based on the notions of density and core objects first defined in the context of the DBSCAN algorithm [11]. In essence, those subspaces are considered most interesting (have the highest *quality*) that contain the highest number of points in ϵ neighborhoods around core objects in the subspace. To this end, the algorithm first computes for every core object o those subspaces in which o can still be considered a core object in a bottom-up manner. A related monotonicity property is exploited in this step. The set of candidate subspaces is subsequently pruned where higher dimensional subspaces are of higher quality than their lower dimensional projections (*downward pruning*). Also, a heuristic approach is presented to prune those k dimensional subspaces that can be thought of as combination of a high-quality $k-1$ dimensional subspace and a low-quality one dimensional subspace (*upward pruning*). The intuition behind this latter reduction is, that a k dimensional subspace resulting from such a combination is not the best possible k dimensional subspace for the subsequent clustering, i.e. a higher quality k dimensional subspace exists.

The problems that the above methods face in the presence of categorical data can be grouped in the following categories.

Distance notions Distances between categorical data points are problematic, as illustrated in section 1.3. Categorical distance metrics, such as SMC, can be used but are likely not to produce equivalent result. In PROCLUS algorithm, for example, the dimensions of the best subspaces for each medoid are determined by the average distances between the medoids m_i and the surrounding data points x_k along every dimension. When using categorical distance measures, the distance along each dimension j degenerates to a boolean value (i.e. 0 if the value of x_k on attribute j is the same as the value of m_i on attribute j, and 1 otherwise). The resulting loss in resolution is likely to reduce the quality of the results. The same observation holds also true for the RIS algorithm, e.g. for the verification of the core object property in different subspaces.

Combinatorial effects The subspace extension approach of CACTUS and MAFIA could potentially be extended to capture categorical data. In fact, one of the central elements of the algorithms itself is the *discretization* of numerical data into a grid. However, not all numerical information is lost in this process. An important attribute of the units used in the subspace extension process is their order, i.e. the faces that they have in common. Clearly, such an order is not present in the case of categorical data. To make the extension technique applicable to categorical data, every combination of two dense cells would have to be considered dense, not only those that are neighbors. The effect would be a combinatorial explosion of the candidate set.

Numerical requirements Finally, methods such as ORCLUS require numerical computations (e.g. to compute the direction of lowest variance), which are not well suited for categorical data sets.

3 Improved Categorical Clustering

This chapter introduces Click, a novel clustering method that maps the categorical clustering problem to the problem of enumerating maximal k-partite cliques in a k-partite graph [31, 34]. Like the attribute summaries used by CACTUS, Click utilizes a compressed representation of the dataset that can be fit into main memory for many datasets. It can be thought of as an adjacency matrix for a k-partite graph.

Consider the sample dataset $\mathcal{D}$ given in table 1 with a total of three categorical attributes A_1, A_2, A_3 and six displayed records. A natural way to depict this dataset is as a simple undirected graph (V, E) where the vertices V represent distinct attribute values and the edges E indicate co-occurrence of these attribute values in some record in the database. Such a mapping is shown in figure 2.

The co-occurrence of a_1 and b_1 in the first record is represented by the edge connecting the vertices a_1 and b_1. However, the multiple co-occurrences of a_2 and b_3 in records 2, 3, and 5 result in only one

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
	$\vdots$		

Table 1: Sample categorical dataset

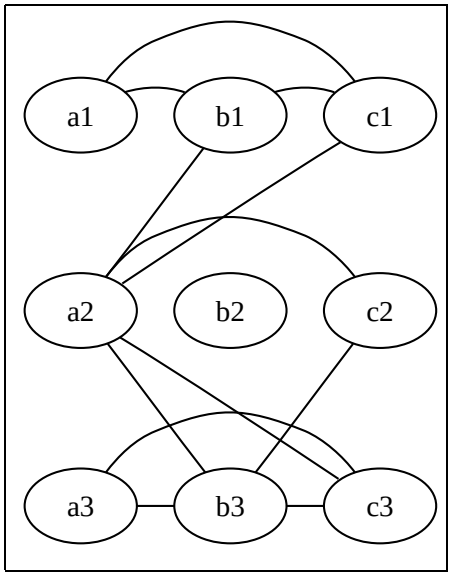

Figure 2: A sample graph encoding

edge connecting these vertices. Note, that the resulting graph is always k-partite, as values from the same attribute can never co-occur in the same transaction. In other words, it is natural to model a categorical dataset as a k-partite graph where the vertex set is partitioned into k disjoint sets (one per attribute) and edges exist only between vertices in different partitions.

Instead of connecting the vertices $v_1, v_2 \in V$ for any co-occurrence, a more general threshold α can be introduced, that quantifies the number of co-occurrences that an attribute value pair must have before the corresponding vertices are connected in the graph model. In particular, α can be chosen to correspond to the threshold that makes the two values strongly connected. Thus the dataset $\mathcal{D}$ can be represented as a k-partite graph $\Gamma(\mathcal{D})$ as follows.

Definition 3.1 (k-Partite Graph) *Let $\mathcal{D}$ be a categorical dataset over attributes $A_1, \ldots, A_n$ and $V = \bigcup_{i=1}^{n} D_i$. Let $\alpha \in \mathbb{R}^{>0}$. The undirected graph $\Gamma(\mathcal{D}) = (V, E)$ where*

$$(v_i, v_j) \in E \iff \sigma^*(\{v_i\}, \{v_j\}) = 1$$

is called the k-partite graph encoding of $\mathcal{D}$.

Definition 3.2 (k-partite (Maximal) Clique) *Let $\mathcal{D}$ be a dataset with $V = \bigcup_{i=1}^{n} D_i$, and let $\Gamma(D) = (V, E)$ be its k-partite graph. $C \subseteq V$ is a* k-partite clique *in $\Gamma(D)$ 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* $(v_i, v_j) \in E$ *in* $\Gamma(\mathcal{D})$*. If there is no* $C' \supset C$ *such that* C' *is a k-partite clique in* $\Gamma(D)$*,* C *is called a* maximal clique*.*

Given the k-partite graph $\Gamma(\mathcal{D})$, the k-partite maximal cliques of the graph correspond to clusters of the underlying dataset.

Theorem 3.3 (Cluster/Clique mapping) *Given a categorical dataset* $\mathcal{D}$ *and a k-interval* $C = C_1 \times \ldots \times C_k$ *with* $C_j \subseteq D_{i_j}$*.*

1. *If* C *is a cluster over attributes* $\{A_{i_1}, \ldots, A_{i_k}\}$*, then* C *is a maximal k-partite clique in* $\Gamma(\mathcal{D})$*.*
2. *If* C *is a maximal k-partite clique in* $\Gamma(\mathcal{D})$*, and* $\sigma^*(C) = 1$*, then* C *is a cluster over attributes* $\{A_{i_1}, \ldots, A_{i_k}\}$*.*

PROOF: $\Rightarrow$. For $i, j \in \{1, \ldots, m\}$, $i \neq j$, C_i and C_j are strongly connected in D, i.e. $\sigma^*(c_i, c_j) = 1$ for all $c_i \in C_i$ and $c_j \in C_j$. Hence, $(c_i, c_j) \in E$ and C is a k-partite clique in $\Gamma(\mathcal{D})$. Moreover, $\nexists C_i' \supset C_i$ s.t. C_i' and C_j are strongly connected, implies that C_i is maximal.

$\Leftarrow$. For $i, j \in \{1, \ldots, m\}$ and $c_i \in C_i, c_j \in C_j$, C is a clique means $(c_i, c_j) \in E$ holds and thus $\sigma^*(c_i, c_j) = 1$. C is a maximal k-partite clique implies that for every possible $C_i' \supset C_i$, C_i' is not strongly connected to at least one C_j. Hence, there is no proper superset of C that satisfies the cluster requirements. Finally, $\sigma^*(C) = 1$ implies $\sigma(C) \geq \alpha \times E[\sigma(C)]$. Thus C is a cluster over $\{A_{i_1}, \ldots, A_{i_m}\}$. □

3.1 The CLICK Algorithm

By theorem 3.3, to mine all the categorical clusters in $\mathcal{D}$ is equivalent to enumerating the set $\mathcal{C}$ of all the maximal k-partite cliques in $\Gamma(\mathcal{D})$, followed by a validation step that verifies whether $\sigma^*(C) = 1$ for all $C \in \mathcal{C}$. Note, that CLICK can mine maximal k-partite cliques for any $1 \leq k \leq n$. 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.

```
CLICK(Dataset D, α, minsup)
  AttributeValueRanking: R = ⋃_{i=1}^{n} D_i
  Clique C = ∅
  CliqueCollection C = ∅

  PreProcess(D, α, Γ(D), R)
  DetectMaxCliques(Γ(D), C, R, C)
  PostProcess(D, C, α, minsup)
  return C
```

Figure 3: The CLICK algorithm

The basic CLICK approach consists of the three principal stages, shown in figure 3, as follows:

- *Pre-Processing*: In this step, the k-partite graph is created from the input database $\mathcal{D}$, and the attributes are ranked for efficiency reasons.
- *Clique Detection*: Given $\Gamma(\mathcal{D})$, all the maximal k-partite cliques in the graph are enumerated.
- *Post-Processing*: the support of the candidate cliques within the original dataset is verified to form the final clusters. Moreover, the final clusters are optionally merged to partially relax the strict cluster conditions.

The details of each step appear below.

3.1.1 Pre-processing

Definition 3.4 (Neighbors) *Let $\mathcal{D}$ be a categorical dataset over attributes $A_1, \ldots, A_n$ and $V = \bigcup_{i=1}^{n} D_i$. Let $\alpha \in \mathbb{R}^{>0}$. The* neighbors *for an attribute value $v_j \in D_i$ are given by the function $N : V \rightarrow 2^V$, defined as*

$$N(v_j) = \left\{ v_k \in V : \sigma^*\big(\{v_j\} \times \{v_k\}\big) = 1 \right\}$$

Note, that the neighbors for an attribute value v_j are those other attribute values V_k that are strongly connected to it. Note also that, by definition, if $v_j, v_k \in D_i$ then $v_k \notin N(v_j)$, since if both v_j and v_k are values of the same attribute A_i they cannot co-occur in the same database record.

CLICK generalizes the clique enumeration technique presented in [20] to handle k-partite cliques. Many clique detection algorithms make use of a heuristic to guide the search for maximal cliques. The number of vertices in a graph that a given vertex is connected to is clearly a good choice when a next best attribute value to continue the search is selected. This heuristic is formalized as the connectivity defined below. Intuitively, it corresponds to number of neighbors ($|N|$) plus the remaining values of the attribute in question ($|D_i| - 1$). However, if a given value does not co-occur with values of other attributes it cannot be part of a k-partite clique. Therefore its connectivity should be zero.

Definition 3.5 (Connectivity) *Let $\mathcal{D}$ be a categorical dataset and $v_i \in D_i$. The* connectivity $\eta(v_i)$ *is defined as:*

$$\eta(v_i) = \begin{cases} |N(v_i)| + |D_i| - 1 & \text{if } |N(v_i)| > 0 \\ 0 & \text{otherwise} \end{cases}$$

Based on the connectivity, a preference for testing nodes to be added to a clique can be formalized. This is captured in the following definition.

Definition 3.6 (Attribute Value Ranking) *Let $\mathcal{D}$ be a dataset over attributes $A_1, \ldots, A_n$ and $V = \bigcup_{i=1}^{n} D_i$. Let $v_i \in V$ for all $i \in \{1, \ldots, m\}$. The total order $v_1, \ldots, v_m$ such that $\eta(v_i) \geq \eta(v_{i+1})$ for $1 \leq i \leq m-1$ is called an* attribute-value ranking *of V.*

The preprocessing step (**PreProcess**$(\mathcal{D}, \alpha, \Gamma(\mathcal{D}), \mathcal{R})$ in figure 3), takes as an input the categorical dataset $\mathcal{D}$ and the threshold α and computes all the strongly connected attribute values to create the k-partite graph $\Gamma(\mathcal{D})$. Also a ranking of the set of all attribute values $\mathcal{R}$ by connectivity is generated.

3.1.2 Enumerating K-partite Maximal Cliques

The clique detection phase is based on the idea that at each point in time only those vertices can be added to a clique that are strongly connected to all previous vertices. If more than one such vertex exists, the attribute value ranking is used to break the tie. It is a recursive algorithm that at each stage tries to expand the current clique in the above fashion to ensure maximality.

Initially the clique detection **DetectMaxCliques** is called with the empty clique C and the full ranked attribute value set $\mathcal{R}$ as list of possible vertices to be used for an extension. Upon return, the clique collection $\mathcal{C}$ 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 subspace clusters are to be mined, or if full space mining is desired (i.e., n-partite cliques) and C contains at least one attribute value for every attribute of the dataset. Otherwise $\Phi(C)$ is *false*. The set $\mathcal{R}^D$ contains all elements of $\mathcal{R}$ that have their *deleted* flag set. Similarly, $\mathcal{R}^P$ is the subset of $\mathcal{R}$ that contains all elements that have their *processed* flag set.

DetectMaxCliques starts by checking if the current clique C covers all relevant attributes and contains all possible attribute values, i.e., it is a maximal clique. $\Phi(C)$ is used to configure the algorithm for either full space clustering or subspace clustering. In the latter case, it always evaluates to true so that only maximality ($\mathcal{R} = \emptyset$) remains as a requirement. In the former case, the attribute values in C are checked for coverage of

```
DetectMaxCliques(Graph Γ(D)
                 CliqueList C,
                 AttributeValueRanking R,
                 Clique C)
  if(Φ(C) ∧ R = ∅)
    C = C ∪ C
    return

  R^D = R^P = ∅
  foreach v in R − R^D − R^P do
      C' = C ∪ {v}
      R' = ∅
      R^D = R^D ∪ {v}

      foreach v' in R − R^D do
        if (σ*(v, v') = 1)
          R' = R' ∪ {v'}
          R^P = R^P ∪ {v'}

      if(Φ(R' ∪ C'))
        DetectMaxCliques(Γ(D), C, R', C')
```

Figure 4: The CLICK Clique Detection

all dimensions of the dataset. If the current clique C satisfies these constraints, it is added to the list of all cliques $\mathcal{C}$ and the search is continued at the previous level.

If C does not fulfill the above requirements, the outer *foreach* loop attempts to add one attribute value v to C in an effort to create a yet larger clique. Note, that at any given point in time $\mathcal{R}$ contains only those attribute values that are strongly connected to all values in C. Hence, adding $v \in \mathcal{R}$ to C will yield another clique C'. v is marked as deleted ($\mathcal{R}^D = \mathcal{R}^D \cup \{v\}$), indicating that it was already considered in the clique construction.

To maintain the condition that all attribute values in $\mathcal{R}$ are strongly connected to all values in C, a $\mathcal{R}'$ matching C' needs to be constructed before recursing. To this end, the inner *foreach* loop scans all attribute values that were possible extensions to C and selects only those that are also strongly connected to the new attribute value v that was added to C. A list of these nodes is maintained in $\mathcal{R}^P$.

Finally, the algorithm recurses on the newly created clique C' with its matching attribute value ranking $\mathcal{R}'$. If only full dimensional clusters are to be detected, part of the search space can be pruned at this point: Only if the clique C' can be extended to the full space through values in $\mathcal{R}'$ (i.e., $\Phi(\mathcal{R}' \cup C')$ is true) does the algorithm have to recurse.

Both, $\mathcal{R}^D$ and $\mathcal{R}^P$, 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 ($\mathcal{R}'$) 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 are strongly connected and, hence, v_2 was part of the $\mathcal{R}'$ accompanying v_1. The set $\mathcal{R}^D$ prunes these cases by recording every value that has already been used to extend C.

Similarly, if v_2 was already part of the $\mathcal{R}'$ accompanying v_1, it need not be considered as an extension to C. This latter case is guarded against by the *processed* attribute values $\mathcal{R}^P$.

Consider the k-partite graph encoding $\Gamma(\mathcal{D})$ in figure 2, where edges denote strong connectivity. An attribute value ranking of V is

$$(a_2(7), b_1(6), c_1(6), b_3(6), c_3(5), a_1(4), c_2(4), a_3(4), b_2(0))$$

where the connectivities $\eta(v)$ are given in parentheses.

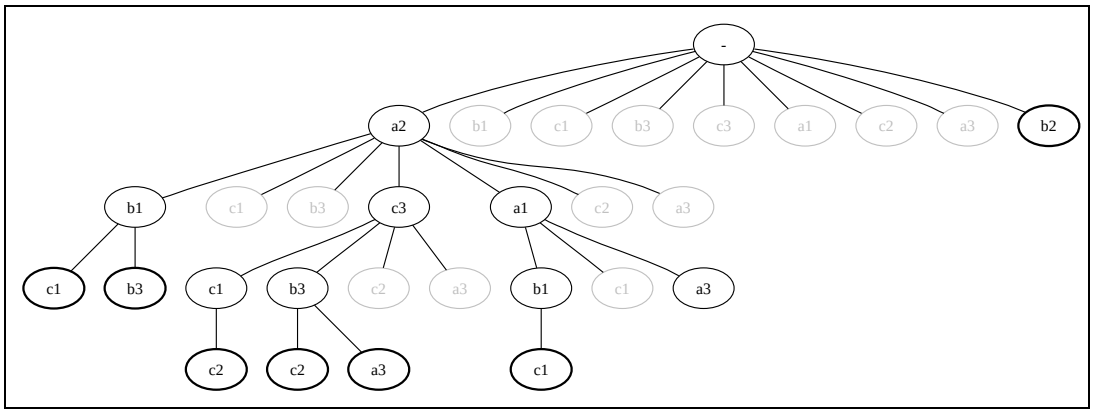

Figure 5: DetectMaxCliques example

Figure 5 shows a corresponding run of **DetectMaxCliques**. Vertices depicted in gray denote search paths that were pruned due to $\mathcal{R}^P$, whereas bold vertices indicate that a clique was found. By following the edges up to the root one can construct the corresponding cliques. The $\mathcal{R}'$ sets can be read from the figure by computing the union of all children of a node. For example, the $\mathcal{R}'$ that is created when **DetectMaxCliques** is called with the clique $\{a_2, b_1\}$ (in the leftmost path) is $\{c_1, b_3\}$. This example shows both full and subspace cliques. For example $\{a_2, b_1, c_1\}$ is a full space clique.

3.1.3 Post-processing

Once all the maximal k-partite (or n-partite) cliques $\mathcal{C}$ have been mined, the post-processing phase (**PostProcess**($\mathcal{D}$, $\mathcal{C}$, α, *minsup*)) involves a single scan of the dataset to count – for each candidate clique $C \in \mathcal{C}$ – the number of transaction in the dataset that support it. If $\sigma^*(C) = 1$, i.e., the support of C is at least α times its expected support, then C is a valid clique.

Note, that the support counting compromises the completeness of the CLICK output. While all maximal cliques are reported by the clique detection phase, some might be pruned out based on the support criteria. Subcliques of a pruned clique, however, might have had proper support. An exploration of the subcliques induced by every rejected clique will restore the completeness of CLICK. The vertical approach presented in section 3.3 could potentially be used to this end.

The method above works well, but for some datasets it may output too many overlapping clique. This is mainly because of the strict notion of strongly connected vertices. For instance, consider a clique $C = C_{i_1} \times \ldots \times C_{i_k}$, and consider a vertex v_m such that v_m is strongly connected to all intervals except for one, say $C_{i_j} = \{v_1, \ldots, v_l\}$. Assume that v_m is strongly connected to all vertices in C_{i_j} except for v_a. In this case v_m cannot belong to maximal C, but it may belong to another maximal clique C' that has a high degree of overlapping intervals with C. Thus it may be appropriate to partially relax the strict clique notion to generate more meaningful clusters.

As an example, consider again the dataset in table 1. Assume that the clique detection algorithm had reported two candidate cliques $C^1 = \{a_2\} \times \{b_3\}$, and $C^2 = \{b_3\} \times \{c_3\}$. Even though a_2 and c_3 may not be be strongly connected (say, strong connectivity would require 3 co-occurrences in the example), it might be reasonable to merge them into the full dimensional cluster $C = \{a_2\} \times \{b_3\} \times \{c_3\}$, since a_2 and c_3 can be deemed *almost* strongly connected.

The enhanced post-processing step in CLICK implements a novel coverage method based on maximal frequent itemsets that merges the final clusters according to a user specified threshold. By building for each transaction in $\mathcal{D}$ the set of cliques that are supported by it, a maximal frequent set mining problem is formulated, the solution of which points to cliques that are often co-supported by the same transaction and should thus be merged. For the example in table 1 (assuming C^1 and C^2 are the only maximal cliques), the

frequent itemset problem would be to find maximally frequent sets in

$$D_M = \Big\{\{\}, \{1\}, \{1,2\}, \{\}, \{1,2\}, \{2\}, \dots\Big\}$$

where the sets contain the respective clique indices. Mining at $minsup = 2$ yields the maximal frequent set $\{1,2\}$ suggesting that C^1 and C^2 be merged.

The problem can intuitively be understood as a coverage problem, where one tries to cover the maximal amount of tuples with the minimal number of cliques, according to the user defined *minsup* parameter. Clearly, a range of options exists for merging the candidate cliques based on the result of the above mining problem. The algorithm hence needs to assign priorities to the maximal frequent itemsets. A good choice is the number of records that would fall into the interval that results from merging the element cliques of such a maximal frequent itemset, i.e. its coverage. A low-complexity approximation, the *coverage weight*, of this priority measure is defined below.

The following definitions formalize the maximal frequent set mining problem associated with a set of candidate cliques and provide the necessary tools to formulate the merging algorithm.

Definition 3.7 (C Set) *Let $\mathcal{D}$ be a dataset, $\Gamma(\mathcal{D}) = (V, E)$ its corresponding graph, and $\mathcal{C}$ the maximal k-partite cliques in $\Gamma(D)$. For every clique $C^i \in \mathcal{C}$ let i denotes its unique clique* id. *The* C Set *of $\Gamma(\mathcal{D})$ is a function C_{Set} that maps every record in $\mathcal{D}$ to the set of clique ids that the record supports.*

$$C_{Set}(t) = \{i : t \in C^i\}$$

Definition 3.8 (Maximal Frequent Clique Set) *Given the C_{Set} of $\mathcal{D}$, the* maximal frequent clique problem *is defined as finding all maximal frequent itemsets at a given minsup level within the problem dataset*

$$D_M = \Big\{C_{Set}(t) : t \in \mathcal{D}\Big\}$$

The solution to the maximal frequent clique problem is called the maximal frequent clique set $\mathcal{F}_{\mathcal{D}}$

The total number of records that support a maximal frequent clique can be approximated by adding up the transactions that support each individual element clique and correcting the $(m-1)$ double countings due to records supporting more than one clique. This approximation can be done with information from the validation and maximal frequent clique mining stages, whereas a precise computation would require a full inclusion/exclusion approach, entailing numerous passes over the dataset.

Definition 3.9 (Coverage Weight) *For a frequent clique set $\mathcal{F}_D$, the* coverage weight $\omega : \mathcal{F}_D \to N$ *is defined as*

$$\omega(X) = \sum_{i=1}^{m} \Big[\sigma_{\mathcal{D}}(C^i)\Big] - (m-1) * \sigma_{D_M}(X)$$

where $X = \{1, \cdots, m\} \in \mathcal{F}_D$ is a set of clique ids (corresponding to cliques $\{C^1, \dots, C^m\}$) that frequently occur together and $\sigma_{D_M}(X)$ denotes X's support within the problem dataset D_M from definition 3.8.

After validating candidate cliques, the enhanced post processing computes the maximal frequent clique set $\mathcal{F}$ of $\mathcal{D}$. The implementation uses an existing GenMax implementation [14] to this end.

the individual maximal frequent sets are then processed in order of descending coverage weight. Each element $\mathcal{F}_{\mathcal{D}}[i]$ is added to $\mathcal{F}^P$, which contains sets of clique ids to be merged in the end. Since no clique can be merged twice, all clique ids that occur in $\mathcal{F}_{\mathcal{D}}[i]$ have to be removed from the not-yet-processed $\mathcal{F}_{\mathcal{D}}[j], j > i$.

Finally, the new clique set $\mathcal{C}$ is created by iterating through the sets of clique ids and merging the cliques accordingly. Note, that while $\mathcal{C}$ contains actual cliques, $\mathcal{F}^P$ contains only the clique ids used in the maximal frequent set mining. Hence, a copy $\mathcal{C}'$ of the original cliques needs to be retained, that can be referenced in the merging process.

```
PostProcess(D, C, α, minsup)
  Scan D and check support of each C ∈ C
  F_D = Maximal Frequent Clique Set of D
  Sort F_D by coverage weights ω

  F^P = ∅
  for i = 1 to |F_D| do
    if F_D[i] ≠ ∅
      F^P = F^P ∪ {F_D[i]}
      F_D[j] = F_D[j] − F_D[i] for all i < j ≤ |F_D|

  C' = C // Save the original cliques
  C = ∅
  for i = 1 to |F^P| do
    if ω(F^P[i]) ≥ E[ω(F^P[i])
      // Add union of cliques with indices in F^P[i] to C
      C = C ∪ C'[F^P[i]]
```

Figure 6: CLICK Post Processing for Merging Clusters

3.2 Merging Characterization

Clusters computed by the clique generation phase of CLICK are (after support checking) exactly those required by the cluster definition as stated in theorem 3.3. However, in favor of a more meaningful output, the merging phase of the post-processor combines the strict original clusters to final clusters that "almost" comply with definition 1.6. The present section characterizes these modified clusters based on the merging phase algorithm.

Consider a set of *merged clusters* $\{C_1^M, \ldots, C_m^M\}$ stemming from applying the CLICK algorithm to dataset $\mathcal{D}$ with parameters α and *minsup*. For every C_i^M there exists a set of *original clusters* $\{C_i^1, \ldots, C_i^{n(i)}\}$ such that

$$C_i^M = \bigcup_{j=1,\ldots,n(i)} C_i^j$$

The individual C_i^M (or more precisely the indices of the C_i^j in the set of original cliques) are either elements of the maximal frequent clique set $\mathcal{F}_\mathcal{D}$ from definition 3.8 or subsets thereof. A C_i^M is a subset of a maximal frequent itemset in $\mathcal{F}_\mathcal{D}$ if the corresponding maximal frequent itemset contained cliques that were already merged with other cliques before the post-processor considered C_i^M. Due to this possibility, the individual C_i^M are only guaranteed to be frequent, the maximality is lost.

A frequent C_i^M entails, that there are at least $r \times minsup$ transactions that support all the component cliques C_i^j ,where r is the number of records in $\mathcal{D}$ that support *any* clique. In other words, the merged cliques are supported by a set of *core* transactions of size at least $r \times minsup$.

Secondly, among all possibilities to form clusters with sufficiently large cores, the CLICK merging phase prefers those that yield larger clusters. To this end, CLICK applies a greedy strategy combined with a heuristic for the resulting clique size. The *coverage weight* introduced in definition 3.9 approximates the number of transactions that will support a given C_i^M and the merging phase processes the C_i^M in order of descending coverage weight.

In summary, the CLICK merging phase produces from the set of originally detected cliques a merged result with the following properties.

- Final cliques C_i^M contain a set of core transactions that support every component clique C_i^j.
- Where more than one way of combining the original clique in that manner exists, CLICK heuristically chooses the solution that will yield the largest cliques.

3.3 Vertical Mining Extension

The CLICK algorithm as outlined above separates the maximal clique mining step from the validation step, where the actual support of each clique in the dataset is computed.

A possible alternative formulation for creating the k-partite graph $\Gamma(\mathcal{D})$ of a dataset $\mathcal{D}$ is a *vertical extension* that allows direct computation of the support of each clique as it is being mined. One way to integrate information about the source transactions would be to introduce a new dimension A_o, where the domain D_o of attribute A_o is the set of all transaction ids in the dataset. A new k-partite graph $G = (V, E)$ can then be constructed as follows: $V = \bigcup_{i=0}^{n} D_i$, and $(v_i, v_j) \in E$ if $v_i \in D_{i>0}, v_j \in D_{j>0}$ and $\sigma^*(v_i, v_j) = 1$, or $v_i \in D_o, v_j \in D_{j>0}$ and v_j occurs in transaction with id v_i.

Figure 7 shows an example where the attribute value a_1, b_1, and c_1 are not only forming a clique among themselves. The edges to the transaction nodes 1 and 2 also indicate the transactions that these specific values occur in.

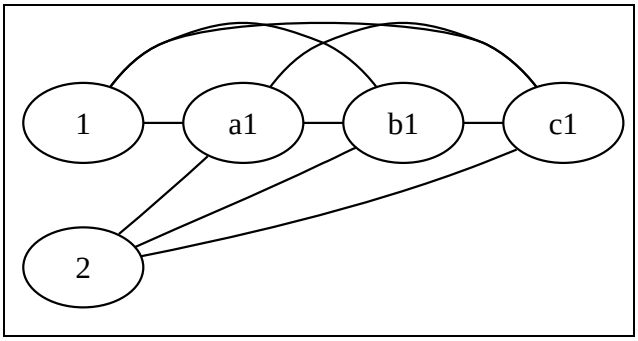

Figure 7: Vertical Representation Example

A maximal clique $C = C_o \cup C_{i_1} \cup \ldots \cup C_{i_k}$ in the $k+1$ dimensional space $D_0, D_{i_1}, \ldots, D_{i_k}$ has support $\sigma(C) = |C_o|$. If the support is at least α times its expected value, then $C_{i_1} \times \ldots \times C_{i_k}$ is a cluster. Alas, this approach is very expensive, since edges have to be added from every transaction id to the set of attribute values in that transaction. The resulting graph would be too large to fit in main memory.

Another alternative is to annotate the vertices of the k-partite graph $\Gamma(\mathcal{D})$ with the transaction ids that the vertices are supported by. Consider once again the example dataset $\mathcal{D}$ in table 1 and its associated graph $\Gamma(\mathcal{D})$ in figure 2. To indicate that the attribute values b_1 occurs in transactions 1 and 4, the vertex corresponding to b_1 could be labeled with $\{1, 4\}$. More formally, a labeling function $\lambda_{\Gamma(\mathcal{D})}$ can be introduced that assigns to each vertex of the graph the transactions that support the vertex in the underlying dataset.

Definition 3.10 (Vertical Labeling) *Let $\mathcal{D}$ be a categorical dataset with attributes $A_1, \ldots, A_n$. The* vertical labeling *function on its k-partite graph is defined as follows.*

$$\lambda_{\Gamma(\mathcal{D})}(v_i) = \{t \in \mathcal{D} : t.A_i = v_i\}^3$$

λ can be extended to capture the supporting transactions of a clique $C = C_1 \cup \ldots \cup C_n$ according to $\lambda(C) = \bigcap_{i \in \{1,\ldots,n\}} \lambda(C_i)$ with $\lambda(C_i) = \bigcup_{v_i \in C_i} \lambda(v_i)$.

This corresponds to the intuition that for a transaction to support a clique, its attribute value for *each* attribute must match *one of* the values of the clique for that attribute. The conjunction entailed by *each* is reflected in the intersection operator between the attributes, while the disjunction *one of* is contained in the union over all attribute values.

Clearly, for vertical mining to be successful, the transaction information needs to be leveraged for pruning the search space depicted in figure 5 to offset the additional computation. Hence, the supporting transactions need to be computed every time a clique is extended, and an appropriate pruning criteria has to be defined.

[3] λ is written in lieu of $\lambda_{\Gamma(\mathcal{D})}$ where it is clear from the context.

Consider a clique $C = C_{i_1} \cup \ldots \cup C_{i_j}$ and an attribute value $v_i \in D_i$ that is not yet in C but strongly connected to all elements of C. To compute the transactions supporting the new clique $C' = C \cup \{v_i\}$ proceed as follows.

- $C \cap D_i = \emptyset$. If v_i is the first value of attribute A_i that is added to C, the new transaction set is the intersection $\lambda(C') = \lambda(C) \cap \lambda(v_i)$ as all supporting transactions t must now meet the additional constraint of having $t.A_i = v_i$.
- $C \cap D_i = R \neq \emptyset$. If v_i is not the first value, the new transaction set can be computed as $\lambda(C') = \lambda(C) \cup \lambda((C - R) \cup v_i)$. In other words all supporting transactions can either come from the supporters of the original clique or from those transactions that fitted into the old clique except for their A_i value v_i.

The latter case can be computationally expensive as $\lambda((C - R) \cup v_i)$ is not necessarily derivable from previous transaction set computations. An appropriate caching strategy needs to be in place, so that the transactions need not be computed from scratch. One can, for example, store all unions for every attribute that have been computed so far. The stored unions can then be used as building blocks for the new transactions set to be computed by performing relatively cheap intersections. Clearly, any caching strategy is expensive in the presence of large datasets. Complementary techniques such as Diffsets [36] can be applied to reduce memory consumption.

With transaction information available at every point of the search tree, a pruning criteria can be defined based on the support of the clique constructed up to that point. Care must be taken not to cut potentially successful branched. Specifically, a branch can not be pruned as soon as the support falls below the *minsup* value (in this case α times the expected support), as it can be in the context of itemset mining [9, 30, 36]. The reason for this is, that the support is not monotonous along the search path. In the example graph in figure 2 and the associate dataset in table 1, the clique $C' = \{a_3, b_3, c_3\}$ has support 1 ($\lambda(C) = \{6\}$). However, when extending the clique to the final clique $C = C \cup \{a_2\}$ the support increases to 3 ($\lambda(C) = \{3, 5, 6\}$).

The underlying question is if, given a frequent clique $C = C_{i_1} \cup \ldots \cup C_{i_j}$, $|\lambda(C)| \geq minsup$, a lower bound $\underline{\sigma}$ can be obtained for the support along a given search path to C. Clearly, such a lower bound must be of the form $\underline{\sigma} = |\lambda(v_{i_1}) \cap \ldots \cap \lambda(v_{i_j})|$ where $v_{i_k} \in C_{i_k}$. This choice is intuitive as it maximizes the number of intersections (by choosing at least one value for every attribute), and minimizes the number of unions (by choosing at most one value for every attribute). It is straightforward to show that no other type of clique $C'' \subseteq C$ can have a lower support.

Let $C' = \{v_{i_1}, \ldots, v_{i_j}\}$ be the intermediate clique as chosen above. C' can be altered in two ways.

- $C'' = C' \cup \{v'_{i_k}\}$ for a $k \in \{1, \ldots, j\}$. Then $\lambda(C'') = \lambda(C') \cup \lambda((C' - \{v_{i_k}\}) \cup \{v'_{i_k}\}) \supseteq \lambda(C')$
- $C'' = C' - \{v_{i_k}\}$ for a $k \in \{1, \ldots, j\}$. Then $\lambda(C'') = \lambda(v_{i_1}) \cap \ldots \cap \lambda(v_{i_{k-1}}) \cap \lambda(v_{i_{k+1}}) \cap \ldots \cap \lambda(v_{i_j}) \supseteq \lambda(v_{i_1}) \cap \ldots \cap \lambda(v_{i_j}) = \lambda(C')$.

In a worst-case scenario even an intermediate clique with zero support can be on a specific search path to a frequent clique, thus negating all pruning strategies for that branch.

However, multiple paths exist to construct a clique. Continuing the example above, the final clique $C = \{a_2, a_3, b_3, c_3\}$ can alternatively be constructed by extending the intermediate clique $C'' = \{a_2, b_3, c_3\}$ with a_3. In this case the support bottleneck along the search path is the intermediate clique C'' with a support of two. Hence, when pruning at a support of two, the second extension strategy is successful while the strategy above fails to detect C. In effect, the *maximum* of the $\underline{\sigma}_P$ for all possible search paths P defines the admissible cut-off point for pruning.

Consider again a frequent clique $C = C_{i_1} \cup \ldots \cup C_{i_j}$. The maximum of the $\underline{\sigma}_P$ is obtained by observing an extension strategy that first extends to an intermediate clique $\{v_{i_1}, \ldots, v_{i_j}\}$ covering every dimension with one value as above. However, for each v_{i_k} one chooses the value from C_{i_k} that yields the highest support for the final clique C. Again, by further extending this clique the support can only grow.

It is straightforward to show that for any frequent clique C and any dimension $C_{i_k} \subseteq C$ there is always an attribute value v_{i_k} that occurs in at least

$$\left\lceil \frac{\sigma(C)}{|C_{i_k}|} \right\rceil$$

transactions that support the final clique.

In the example above, from the total support three of C, one transaction involves a_3 and two transactions involve a_2. Consequentially, a_2 is chosen when constructing the maximum $\underline{\sigma}_P$. Moreover, the equation above states that there is an attribute value that occurs in at least two transactions that support the final clique. Given a final clique supported by three transactions and one dimension with two values, either all transactions involve one of the values in that dimension, or they are distributed 2:1.

In summary, if all possible search paths to a clique are explored, pruning those where the support of any intermediate clique drops below $\left\lceil \frac{\sigma(C)}{|C_{i_k}|} \right\rceil$ will preserve the completeness of the search strategy. Alas, the value for $|C_{i_k}|$ is not known in advance, since the clique has not been completely constructed. The only bound that can be exploited is that for any clique C with $\sigma(C) \geq minsup > 0$ there is always a search path where the support for any intermediate clique is always above zero. The discussion is summarized in the following observation.

Observation 3.1 (Vertical Pruning Bound) *Let $\mathcal{D}$ be a dataset, $\Gamma(\mathcal{D})$ its k-partite graph, and C a maximal k-partite clique in $\Gamma(\mathcal{D})$ with $\sigma_{\mathcal{D}}(C) > 0$. Then there exists a sequence of sub-cliques*

$$\emptyset = C_o \subseteq C_1 \subseteq \ldots ... \ldots \subseteq C_n = C$$

with $C_i = C_{i-1} \cup \{v_i\}, v_i \in V(\Gamma(\mathcal{D}))$ for $i \in \{1, \ldots, n\}$ such that $\sigma_{\mathcal{D}}(C_i) > 0$.

Observation 3.1 can be used to extend the original CLICK algorithm as shown in figure 8. This algorithm integrates the pruning ideas discussed above combined with a simple decision criteria for backtracking: If a certain way of expanding the current clique leads to no branches in the subtree that actually expand to cover all dimensions, then other expansions must be explored. Clearly, this criteria works only for full-dimensional clustering. Other criteria will have to be developed for the subspace case.

The procedure has been modified to return whether or not there were branches in the current subtree that could be extended to cover all dimensions. This information is later on used to decide whether a node potentially threatens the completeness of the search through pruning.

The pruning decision itself is based on the vertical information computed in $\lambda_{Gamma(\mathcal{D})}(C')$ for the current extension candidate C'. If the support for the current clique falls to 0, observation 3.1 permits pruning the current branch at the expense of trying other ways to construct potentially pruned maximal cliques. This is achieved by resetting the markings $\mathcal{R}^D$ and $\mathcal{R}^P$ to their state before the current extension was attempted.

Moreover, multiple levels of backtracking need to be permitted. Consider, a parent clique C_p which is extended to cliques $C_1, \ldots, C_n$. All of the child cliques may still have non-zero zupport. Yet, the way that C_p is extended may lead to zero support cases further down the subtree, where the actually problematic extension was the one of C_p itself. By verifying that at least one of the **DetectMaxCliquesVertical** cases returns true, recovery is provided from this problem.

The vertical technique obliviates the need for support checking in the post-processing algorithm shown in figure 6. However, performance studies indicate that the vertical approach does not yield better performance than the regular approach.

4 Experimental Study

This chapter presents an extensive performance study of CLICK versus CACTUS and other methods. All testing was done on a hyper-threaded quad-processor Intel Xeon 2.8GHz with 6 gigabytes of RAM, running the 2.4.22 SMP Linux kernel. All datasets were stored on an NFS mounted network drive on the same local

```
DetectMaxCliquesVertical(Graph Γ(D)
                         CliqueList C,
                         AttributeValueRanking R,
                         Clique C)
  if(Φ(C) ∧ R = ∅)
    C = C ∪ C
    return true

  R^D = R^P = ∅
  foreach v in R − R^D − R^P do
      C' = C ∪ {v}
      R' = ∅
      R^D = R^D ∪ {v}

      P = ∅
      foreach v' in R − R^D do
        if (σ*(v, v') = 1)
          R' = R' ∪ {v'}
          R^P = R^P ∪ {v'}
          P = P ∪ {v'}

      if(Φ(R' ∪ C'))
        if(|λ_Gamma(D)(C')| > 0)
          if(DetectMaxCliquesVertical(Γ(D), C, R', C') = false)
            R^D = R^D − {v}
            R^P = R^P − P
        else
          // Prune, but other search paths must be explored
          R^D = R^D − {v}
          R^P = R^P − P

  if(No DetectMaxCliquesVertical call returned true)
    return false
  else
    return true
```

Figure 8: The Vertical CLICK Clique Detection

100MBit network. The code for CACTUS was obtained directly from its authors. The experiments will show that CLICK detects the same clusters as CACTUS while directly computing higher dimensional clusters, and significantly outperforming it.

All synthetic datasets used in these tests were created using the generation method proposed in [13]; the code was provided by Ganti et al. The generator creates a user specified number of tuples 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 tuples in designated cluster regions, thus increasing the support of these regions above their expected support.

In the performance studies the clusters were located on the attribute values $[0, 9]$ and $[10, 19]$ for every attribute unless otherwise specified. Each cluster was created by adding 5% of the original number of tuples in this interval region. In all performance tests, $\kappa = 3$ and $\alpha = 3$ were chosen for CACTUS as suggested by Ganti et al. CLICK was also configured to use $\alpha = 3$.

4.1 CACTUS Extension

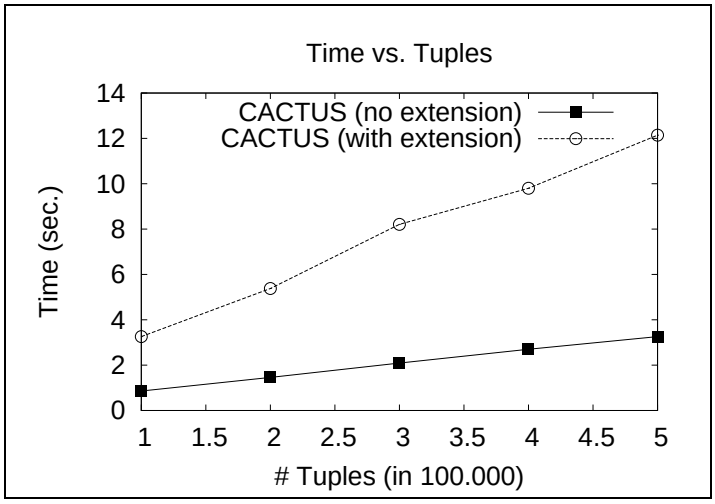

Figure 9: Performance Impact of Extensions

As mentioned in section 2, the available CACTUS implementation stops at the stage where it finds the potential cluster projections on each attribute, and it does not extend these to produce the final n dimensional clusters or subspace clusters. It is thus unclear whether or not the reported performance in [12] accounts for extension and validation. To study the impact of these additional steps, the CACTUS implementation was augmented with the cluster extension and validation steps. Figure 9 shows the running time of CACTUS with and without the additional steps.

It is obvious that the remaining steps are expensive. 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 experiments with some of the synthetic datasets and settings used in [12], CACTUS reported between 10 and 100 cluster projections per dimension. The combination of these projections resulted in enormous candidate sets. As a trade-off, the validation step could be performed interleaved with the candidate generation to eliminate false candidates as early as possible and thus prevent combinatorial explosion. However, this would impact the scalability over large datasets as one pass of the dataset is required for each validation. The effect of the extension procedure on the overall performance grows with both, the number of attribute values in the cluster, and the dimensionality of the dataset. In the remaining performance studies only the base-line CACTUS version is used, since the version with extensions is too slow.

4.2 STIRR and ROCK

The STIRR [13] 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. Ganti et al. report in [12] that CACTUS outperforms STIRR by a factor 3 - 10. These performance gains could not be reproduced in the present tests, even though the original source code was used. However, it seems clear that the final cluster extraction step in STIRR would cost at least as much as the extension step in CACTUS.

Like STIRR, the ROCK algorithm does not lend itself well to a direct comparison with CLICK. While CLICK 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 figure 10 shows that CLICK still outperforms ROCK by orders

of magnitude. The practical application of ROCK is thus limited to datasets of well below 10000 records where CLICK scales into the million record range.

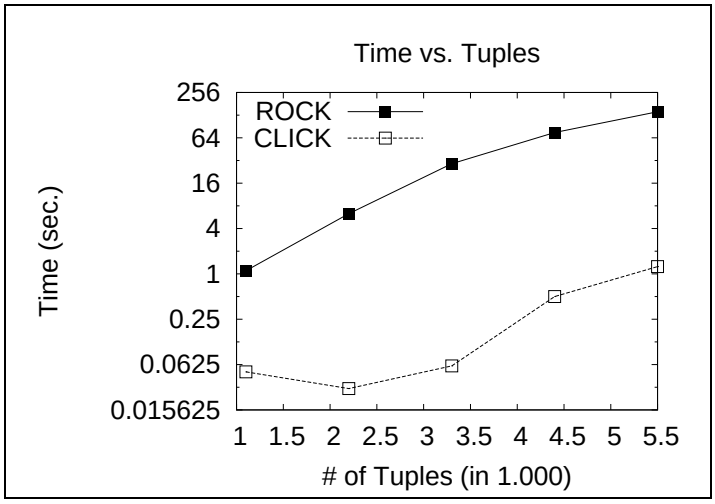

Figure 10: CLICK vs. ROCK

The remainder of the performance study hence only compares CLICK with CACTUS.

4.3 Performance Comparison

Three test series on synthetic datasets were performed to compare the time performance between CLICK and CACTUS: Performance over tuples, attributes, and domain sizes.

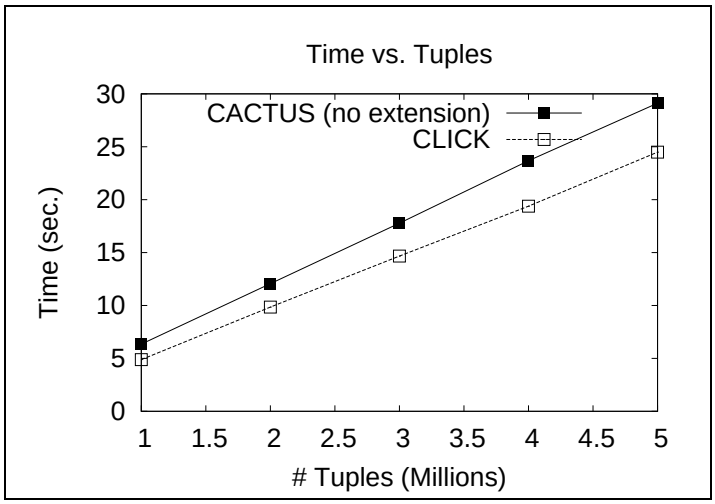

Figure 11: CLICK vs. CACTUS (Tuples)

Performance vs. Dataset Size: Synthetic datasets with 10 attributes, and 100 attribute values per dimension were used for this test, while the total number of tuples was varied from one to five million. Both methods scale linearly over the number of tuples in the source dataset, as can be seen in figure 11. CLICK outperforms CACTUS in this category by an average of 20%.

Performance vs. Domain size: Datasets with one million tuples and four attributes were used to measure the performance in relation to the domain size. The number of attribute values per attribute were varied

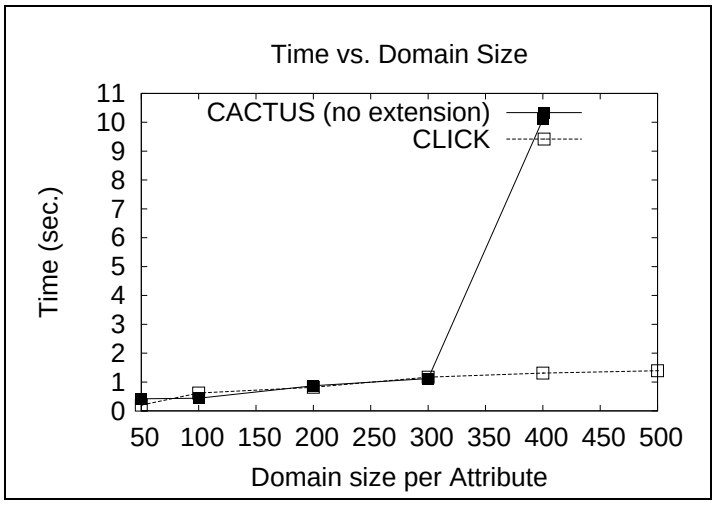

Figure 12: CLICK vs. CACTUS (Domain Size)

from 50 to 500. Figure 12 shows that 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. Ganti et al. use a "multi layered approach" in their own experiments to compress the memory consumption of their approach. CLICK scales well beyond this point without a need for additional memory compression.

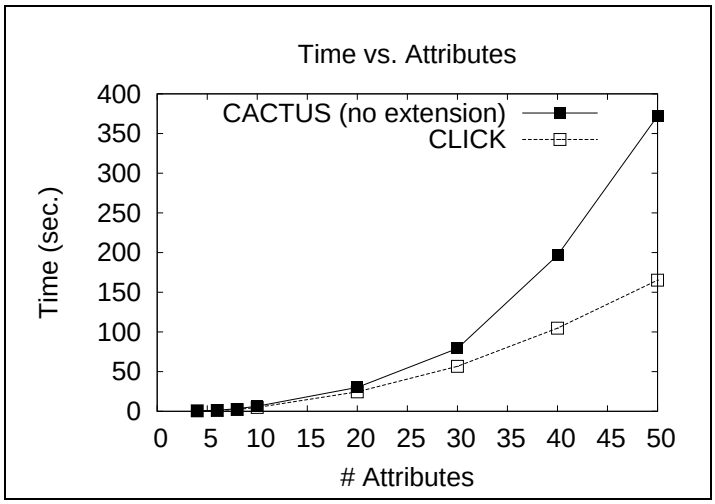

Figure 13: CLICK vs. CACTUS (Attributes)

Performance vs. Dimensionality: CLICK is especially scalable with regards to higher dimensional data. On a dataset with 1 million tuples and 100 attribute values per dimension, CLICK outperforms CACTUS by a factor 2 - 3 when varying the number of attributes from 10 to 50 as shown in figure 13.

4.4 Vertical Mining Performance

Section 3.3 presents a vertical mining extension to the standard CLICK algorithm that mitigates the need for a second pass over the dataset. Instead, the vertical CLICK method requires additional memory and additional CPU time for set computations.

The vertical extension was implemented in CLICK and benchmarked against the CLICK baseline version. As in the previous performance study, the number of tuples, attributes, and attribute values were varied to study the effect of these parameters.

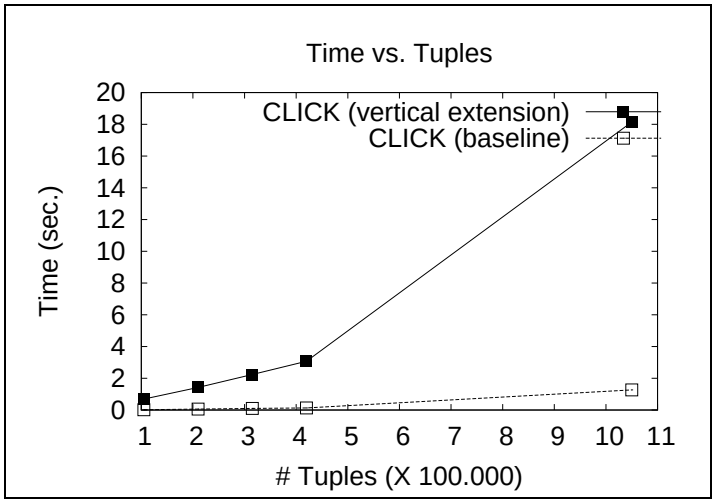

Figure 14: CLICK Vertical Performance (Tuples)

Performance vs. Dataset size: Figure 14 shows the execution time of both versions versus the dataset size. Clearly, the vertical extension introduces a large performance penalty. Harddisk operations would have to be more expensive by an order of magnitude to justify the additional computation. The pruning during the clique generation phase cannot offset this additional cost.

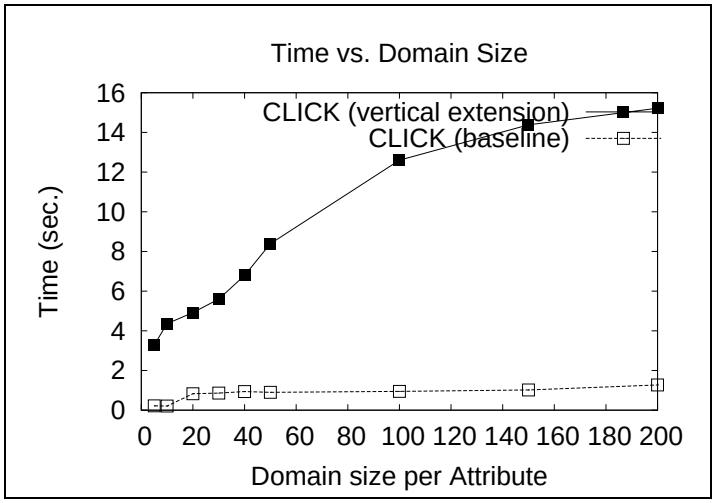

Figure 15: CLICK Vertical Performance (Domain Size)

Performance vs. Domain size: As in the previous case, the CLICK baseline version yields better results than the vertical implementation (figure 15).
Performance vs. Dimensionality: Finally, the execution time was measured versus against the number of attributes in the dataset (figure 16). Again, the vertical extension was proven to be computationally more intense than the baseline version.

Overall, the vertical extension increases the computational cost of the algorithm instead of reducing it. The overhead of the additional initialization and interleaved set computations could be acceptable if the

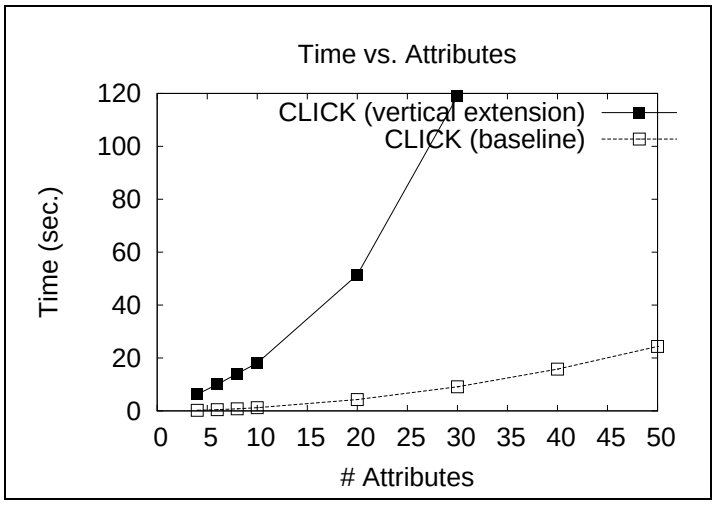

Figure 16: CLICK Vertical Performance (Attributes)

algorithm gained advantages with the number of tuples, as expected. However, the cost of hard disk access would have to be one order of magnitude higher for this effect to present itself. As mentioned above, the test machine read the datasets from a network drive already. A local hard drive would further improve the relative performance of the CLICK baseline version.

4.5 Clustering Quality

To evaluate the quality of the clusters generated by CLICK, three basic scenarios were tested on synthetic datasets and compared to the output of other categorical clustering algorithms. For the following experiments, α was set to 3 and the post-processing was turned off in order to verify the actual reported cliques before merging.

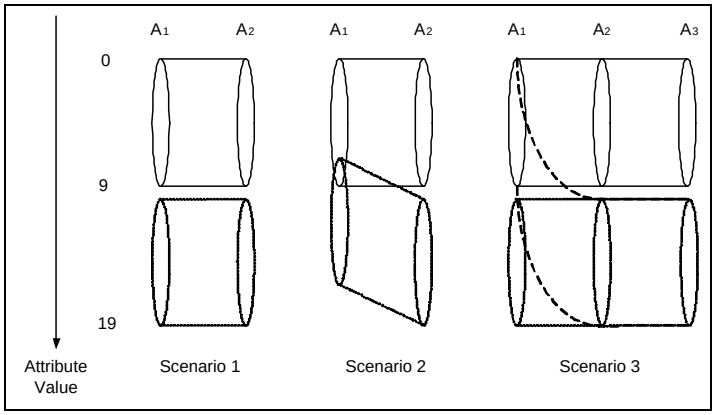

Figure 17: Cluster Quality Comparison

Scenario 1 A dataset with clear separation of two clusters. CLICK detected both individual clusters on the appropriate attribute values.

The available CACTUS implementation reported a total of 480 cluster projections, 240 per attribute. These represented all subsets of size 3 of the sets $\{0, \ldots, 9\}$ and $\{10, \ldots, 19\}$. Clearly, these subsets are part

of the cluster projection. However, they do not satisfy the maximality condition of the final clusters. The extension implementation then 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. Overall, the extension reported 115.200 clusters, reflecting the lack of maximality of the cluster 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 [13] this corresponds to a single cluster on $[0, 19] \times [0, 19]$, confirming the lack of separation found in [12].

Scenario 2 A dataset with a slight overlap between two clusters on one attribute. CLICK detected three initial cliques, two of which represented the original clusters and an additional clique on $[7, 9] \times [0, 19]$. The post processing step could optionally merge this third clique with one of the two primary cliques. Note, that the third clique is nevertheless correctly reported according to the cluster definition 1.6.

CACTUS, again, reported 480 cluster projection. In this scenario, these were all subsets of size 3 of $\{0, \ldots, 9\}$ and $\{7, \ldots, 16\}$. The extension procedure was then used to confirm that all extended cluster projections were indeed subsets of the three clusters that CLICK reported.

STIRR reported weights of about 0.15 for the attribute values $[0, 6]$ and $[10, 16]$ on the first attribute, and for the values $[0, 19]$ on the second attribute. The overlap in clusters was reflected in weights of about 0.21 for the values $[7, 9]$ on the first attribute. All other attribute values were reported to be about 0.08. A non-trivial post-processing step external to STIRR could be able to locate this overlap.

Scenario 3 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 the remaining attributes. CLICK 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 CLICK. This behavior is correct with respect to the cluster definition 1.6, as $[10, 19] \times [10, 19] \times [10, 19]$ is not maximal.

CACTUS reported all subsets of size 3 of $\{0, \ldots, 20\}$ on attributes one and three, and all subsets of size 3 of $\{0, \ldots, 9\}$ and $\{10, \ldots, 19\}$ on attribute two, yielding a total of 312 million extension candidates. Validation of this candidate set was not possible on the available machine. However, the reported subsets for attribute two are likely erroneous. In fact, they should be the same as for attribute three. If this correction was made, an extension should theoretically yield the same final clusters that CLICK reports.

As in scenario 2, 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 value.

4.6 Real Data Clustering

Four datasets were studied in an effort to judge the quality of the CLICK results on real data. These datasets were

Mushroom Dataset The Mushroom dataset is part of the UCI Machine Learning Repository [4] and contains 8124 records and 22 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). All 22 attributes are categorical.

Congressional Votes Dataset The Congressional Votes dataset is also part of the UCI repository. It contains 435 records indicating the voting behavior of Congressmen in terms of their votes in 16 different polls in 1984. Each record is labeled to be either Republican (168 records) or Democratic (267 records). The individual attributes are boolean valued (yes or no vote).

Bibliographic Dataset The bibliography dataset used in [12] contains 7766 records of database publications [33], and 30919 records of papers on theoretical Computer Science [29]. Each records has four

[4] http://www.ics.uci.edu/~ mlearn

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

Table 2: Confusion Matrix Mushrooms (Full Space)

	None	C_1	C_2	C_3	C_4	C_5	C_6	C_7	C_8	C_9
P	0.0 %	0.0%	21.3%	0.0%	0.0%	3.5%	0.0%	0.0%	0.0%	0.0%
E	0.0 %	2.4%	0.0%	0.8%	6.3%	0.0%	9.5%	0.6%	1.6%	1.8%
	C_{10}	C_{11}	C_{12}	C_{13}	C_{14}	C_{15}	C_{16}	C_{17}	C_{18}	C_{19}
P	0.0%	0.0%	0.1%	0.0%	0.5%	0.4%	2.8%	0.0%	0.1%	19.5%
E	17.9%	0.1%	0.3%	0.6%	0.5%	0.0%	0.0%	0.0%	9.4%	0.0%

Table 3: Confusion Matrix Mushrooms (Subspace)

attributes, namely the first and second author, the conference, and the year it was published. Where a paper was written by a single author, the name is replicated to the second attribute. The challenge of this dataset lies within the large domain sizes of over 10.000 values for each of the authors, over 2.300 values for the conference, and 52 values for the year attribute.

Reuters 21578 Dataset The Reuters dataset is a standard benchmarking dataset for text categorization and is also included in the UCI repository. It contains 21578 articles from the Reuters news agency as well as a number of meta information, such as places and people the article talks about, as SGML markup. To make the data accessible to CLICK the articles were pre-processed using a morphology function [10] to collapse different verb forms and tenses (e.g. *am*, *was*, *is* are all mapped to *be*). A frequency dictionary was built and pruned to the 193 most frequent words. Finally, the first 10000 articles were encoded according to the scheme presented in section 1.3. The result was a boolean dataset with 193 attributes and 10000 records on which CLICK could be used.

Full dimensional clustering as well as subspace clustering were applied to each of the datasets.

The Mushroom Dataset The mushroom dataset is rather sparse with 22 attributes and typically 6-10 values per attribute and only about 8000 records. Assuming an average of 8 values per attribute, one obtains $8^22 \approx 74 \times 10^18$ possible tuples. For that reason, CLICK was configured to run with a low α value of 0.4. Not surprisingly, many of the candidate clusters were overlapping. By assigning each tuple to the first cluster that contains it, tables 2 and 3 were generated for full dimensional and subspace clustering, respectively. The two rows represent the two original classes (poisonous and edible) while the columns represent the clusters that CLICK generated.

Full dimensional clustering (table 2) initially yielded 256 candidate clusters which were then reduced to 213 clusters using a *minsup* value of 0.5% for the post processing step. While about 9% of the tuples could not be clustered (column *None*), the remaining clusters exhibit perfect purity with respect to the original labeling.

The subspace clustering produced 596 initial clusters. This figure was reduced to 553 by merging with a *minsup* setting of 5%. As in the full dimensional case, a large number of clusters overlapped. By assigning each tuple to the first cluster that contains it, table 3 was obtained. The subspace procedure clearly improved the result with respect to the unclustered tuples (0% down from 9% in the full dimensional case). Alas, some of the generated clusters now show impurities with respect to the original labeling. Since this figure is below

	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%

Table 4: Confusion Matrix Votes (Full Space)

	None	C_1	C_2	C_3	C_4	C_5	C_6	C_7	C_8	C_9
R	0.5%	0.0%	0.2%	0.0%	36.5%	0.0%	0.0%	0.0%	0.0%	0.5%
D	0.4%	4.4%	0.5%	0.7%	3.2%	4.4%	45.6%	1.8%	0.2%	0.2%
	C_{10}	C_{11}	C_{12}							
R	0.5%	0.2%	0.2%							
D	0.0%	0.0%	0.0%							

Table 5: Confusion Matrix Votes (Subspace)

1% it is potentially acceptable. By using all 553 clusters a perfectly pure clustering is obtained. However, this level of granularity will be inappropriate for most applications.

The Congressional Votes Dataset As with the Mushroom dataset, the Congressional Votes dataset is relatively sparse with a total of $2^{16} = 65536$ potential tuples, while the dataset contains only 435 records. An α value of 0.1 was used in this experiment.

Table 4 shows the results for a full dimensional clustering with CLICK. The rows indicate the two original classes (Republican or Democrat). The post processing step proved to be especially useful in this case, as it reduced the original 51 cluster candidates down to 13 at a *minsup* level of 5%. Of these 13 clusters, the first 8 contained almost 98% of all tuples. Only 2.5% of the voting behaviors could not be clustered using this approach.

Using subspace clustering, the rate of unclustered tuples was reduced to about 1% while increasing the number of relevant clusters to 12 (table 5). The actual number of detected clusters was 30, down from 68 candidates before the merging procedure. Interestingly, the subspace clustering preserves all full dimensional clusters (C_1 - C_8) and adds four 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. By not considering these issues (i.e. by leaving out these dimensions) the algorithm can capture "non-standard" voting behaviors. This effect is emphasized by a number of missing values in the voting behavior records.

The Bibliographic Dataset The bibliographic dataset did not lend itself well to clustering. In a number of experiments a maximum of 12% of the publications could be clustered. However, those clusters that were detected provided insightful summaries on groups of authors that were active at certain conferences at a certain time. Table 6 shows an excerpt from one of the eight clusters that were detected using $\alpha = 65$ and $minsup = 0.005$ when performing full-dimensional clustering. This particular cluster captured a total of 260 publications in theoretical Computer Science by authors that co-published or appeared at the same time at the same conferences.

The Reuters 21578 Dataset Due to the specific classification of articles in the document base (e.g. places and people that articles report about), no confusion matrix could meaningfully be generated. Instead, the reported clusters were verified manually. However, by manually adjusting the words in the frequency dictionary CLICK could be used to cluster according to such specific dimensions.

CLICK was configured to run a subspace clustering with α set to 0.5 and *minsup* set to 1%. The result were 7 clusters as shown in table 7. Clearly, the clustering found by the algorithm gives a meaningful picture

Dimension	Values
Author 1	{Abadi Hromkovic Goldberg Papadimitriou ... }
Author 2	{ Seymour Yung Watanabe Schneider Gunopulos Motwani ... }
Conference	{ALGORITHMS DMATH ESA EUROCRYPT IEEETC ... }
Year	{1991 1988 1989 1992 1990 1993 1995 1996 1994 }

Table 6: Sample clique for the bibliographic dataset

Cluster	Support	Words
1	119	net, tax, share, revenue, profit, ...
2	73	low, fund, firm, credit, decline, american, commission, ...
3	168	say, agree, product, business, financial, investment, ...
4	199	reuter, record, ...
5	230	new, york
6	58	reuter, say, capital
7	200	march, company, exchange, ...

Table 7: Clusters for Reuters 21578

of the articles within the collection. The second most frequent cluster 7 can, for example, be interpreted as a large number of articles reporting on the situation of publicly listed companies in the month of march, or in other words the stock exchange impact of their Q1 results. Similar meaningful interpretations exist for the other clusters.

4.7 Post-processor Performance

The CLICK post-processing allows for merging "almost strongly connected" cliques based on a user defined similarity threshold. The effectiveness and performance of this merging step were evaluated on the bibliographic dataset used in [12]. Because of the low number of attributes (4) and the high density within the dataset, CLICK was configured to use an α value of 75 yielding 258 initial cluster candidates.

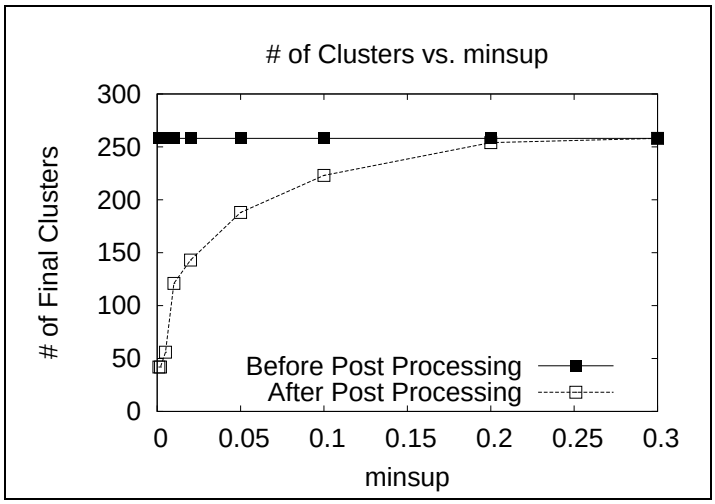

Figure 18: Post-processor Reduction

While varying the *minsup* value from 0.1% to 30%, the duration of the post-processing phase and

the number of final clusters were recorded. The cluster reduction shown in figure 18 demonstrates the effectiveness of the merging procedure in reducing the number of output clusters, as well as its responsiveness to the user determined *minsup* threshold.

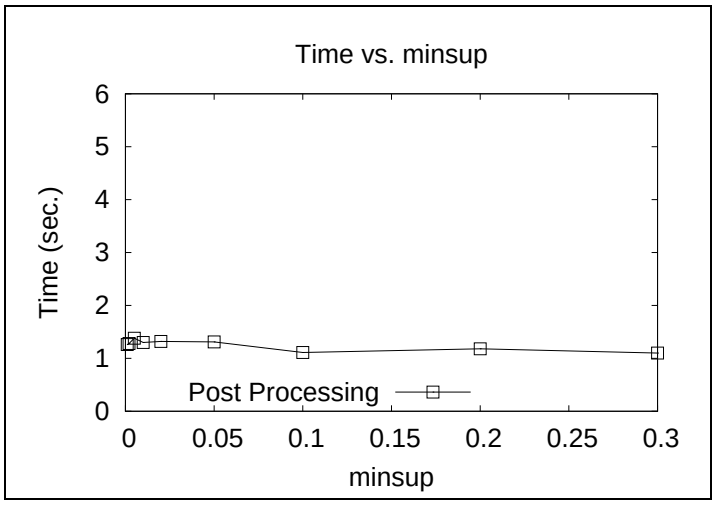

Figure 19: Post-processor Performance

Moreover, the merging performance is not affected by the chosen *minsup* value, as can be seen in figure 19. On a moderately sized dataset such as the bibliographic dataset ($\approx$ 38000 records) used for this experiment, only between 1 and 2 percent of the total execution time are spent validating and merging clusters.

5 Discussion and Conclusions

This work introduced CLICK, a novel clustering algorithm for categorical data based on enumerating maximal cliques in k-partite graphs. A new merging procedure was introduced as part of the work on CLICK. By leveraging existing maximal frequent set algorithms, the post-processing phase of the algorithm relaxes the strict cluster definition based on a user defined parameter. The algorithm performance and cluster quality were evaluated. The experiments indicate substantial performance gains of factor two to three over previous approaches, especially on high dimensional datasets.

An extension to CLICK was studied that uses a vertical representation of the original dataset. But while vertical technique have been successfully applied in association rule mining, the vertical approach delivers unsatisfactory results in the context of clique-based clustering. The theory behind this phenomenon was developed in this text.

In summary, the following contributions were made in this work.

Clique-based categorical clustering . The CLICK algorithm is the first categorical clustering algorithm to be based on an intuitive mapping between the categorical clustering problem and a well-known graph problem. It was formally proved that this mapping retains soundness and completeness of clustering techniques in the new space with respect to the cluster definition. This alternative representation opens a new perspective for future work in this field.

Direct subspace clustering . CLICK naturally integrates subspace clustering capabilities into the clustering process. This is significant step forward from previous methods that first clustered data in low dimensional subspaces and then combined low dimensional clusters into higher dimensional ones. It thus mitigates the need for expensive combinatorial post-processing steps.

A novel cluster merging approach . A new technique for merging clusters that are "almost" strongly connected was introduced. The merging step generates alternative output clusters that have a user defined set of common transactions and a number of non-core transactions that prevented the clusters from being merged in the first place. Experiments on various real datasets demonstrated that this technique can generate cluster outputs that are small in the number of clusters but still retain good separation characteristics.

Results in vertical clustering . The vertical mining approach known from association rule mining was explored. It was shown that the bar for a successful vertical approach is relatively high in clique-based clustering as the few additional pruning possibilities are easily outweighed by the additional search paths. This effect is directly related to a non-monotonicity property shown in this text.

A number of possible directions exist for future research. Firstly, alternative clique extension heuristics can be explored, especially with respect to vertical techniques. Even if the pruning capabilities of such techniques are inherently limited, an extension heuristic that guides more clique extensions to an immediate success could mitigate some of these drawbacks.

Secondly, vertical techniques can potentially be used to extend CLICK to detecting all maximal frequent cliques. Since the algorithm detects for each maximal frequent clique at least a super clique, the search space for this post-processing step would be limited in a way that makes vertical approaches feasible.

Thirdly, the merging phase could be improved by allowing the user to specify the number of clusters desired in the final output, as opposed to the rather cryptic *minsup* specification.

And finally, other graph-based techniques should be explored toward improved categorical clustering. The mapping between the two domains developed in this text can serve as a basis for future research in this direction.

Acknowledgement

Prof. Thomas Seidl contributed to this work through a number of helpful suggestions. Venkatesh Ganti kindly p rovided the source code for CACTUS and STIRR, and Eui-Hong "Sam" Han provided the ROCK implementation.

References

[1] C. C. Aggarwal, C. M. Procopiuc, J. L. Wolf, P. S. Yu, and J.S. Park. Fast algorithms for projected clustering. In *Proceedings of the SIGMOD*, pages 61–72, 1999.

[2] C. C. Aggarwal and P. S. Yu. Finding generalized projected clusters in high dimensional spaces. In *Proceedings of the SIGMOD*, pages 70–81, 2000.

[3] R. Agrawal, J. Gehrke, D. Gunopulos, and P. Raghavan. Automatic subspace clustering of high dimensional data for data mining applications. In *ACM SIGMOD Conf. Management of Data*, June 1998.

[4] R. Agrawal and R. Srikant. Fast algorithms for mining association rules. In *20th VLDB Conference*, September 1994.

[5] D. Barbara, Y. Li, and J. Couto. Coolcat: an entropy-based algorithm for categorical clustering. In *ACM Press*, pages 582–589, 2002.

[6] Kevin Beyer, Jonathan Goldstein, Raghu Ramakrishnan, and Uri Shaft. When is "nearest neighbor" meaningful? *Lecture Notes in Computer Science*, 1540:217–235, 1999.

[7] T.H. Cormen, C.E. Leiserson, and R.L. Rivest. *Introduction to Algorithms, Second Edition*. MIT Press, 2001.

[8] D. Cristofor and D. Simovici. An information-theoretical approach to clustering categorical databases using genetic algorithms. In *2nd SIAM ICDM, Workshop on clustering high dimensional data*, 2002.

[9] B. Dunkel and N. Soparkar. Data organization and access for efficient data mining. In *15th IEEE Intl. Conf. on Data Engineering*, March 1999.

[10] C. Fellbaum (Ed.). *WordNet, An Electronic Lexical Database*. Bradford Books, 1998.

[11] M. Ester, H.P. Kriegel, J. Sander, and X. Xu. A density-based algorithm for discovering clusters in large spatial databases with noise. In *Proceedings of the Second International Conference on Knowledge Discovery and Data Mining*, pages 291–316, 1996.

[12] V. Ganti, J. Gehrke, and R. Ramakrishnan. CACTUS: Clustering categorical data using summaries. In *ACM SIGKDD Int'l Conference on Knowledge discovery in Databases*, 1999.

[13] D. Gibson, J. Kleinberg, and P. Raghavan. Clustering categorical data: An approach based on dynamical systems. In *24th Int'l Conference on Very Large Databases*, August 1998.

[14] K. Gouda and M. J. Zaki. Efficiently mining maximal frequent itemsets. In *1st IEEE Int'l Conf. on Data Mining*, November 2001.

[15] S. Guha, R. Rastogi, and K. Shim. Rock: A robust clustering algorithm for categorical attributes. In *International Conference on Data Engineering*, 1999.

[16] E. Han, G. Karypis, V. Kumar, and B. Mobasher. Clustering based on association rule hypergraphs. In *Research Issues on Data Mining and Knowledge Discovery*, 1997.

[17] J. Han and M. Kamber. *Data Mining: Concepts and Techniques*. Morgan Kaufmann Publishers, San Francisco, CA, 2001.

[18] D. Hand, H. Mannila, and P. Smyth. *Principles of Data Mining*. MIT Press, Cambridge, MA, 2001.

[19] Z. Huang. A fast clustering algorithm to cluster very large categorical data sets in data mining. In *SIGMOD Workshop on Research Issues on Data Mining and Knowledge Discovery*, Tucson, AZ, 1997.

[20] H.C. Johnston. Cliques of a graph - variations on the bron-kerbosch algorithm. *International Journal of Computer and Information Sciences*, 5(3), 1976.

[21] K. Kailing, H.P. Kriegel, Peer Kroeger, and S. Wanka. Ranking interesting subspaces for clustering high dimensional data. In *PKDD*, pages 241–252, 2003.

[22] M. Koyutürk and A. Grama. PROXIMUS: A framework for analyzing very high-dimensional discrete attributed datasets. In *Proceedings of the Ninth ACM SIGKDD Intl. Conf. on Knowledge Discovery and Data Mining (KDD 2003)*, pages 147–156, 2003.

[23] H. Nagesh, S. Goil, and A. Choudhary. Mafia: Efficient and scalable subspace clustering for very large data sets, 1999.

[24] R.T. Ng and J. Han. Efficient and effective clustering methods for spatial data mining. In *Proceedings of the 20th International Conference on VLDB*, pages 144–155, 1994.

[25] C. Ordonez. Clustering binary data streams with k-means. In *Proceedings of DMKD*, pages 12–19, 2003.

[26] C. M. Procopiuc, M. Jones, P. K. Aggarwal, and T. M. Murali. A monte carlo algorithm for fast projective clustering. In *SIGMOD*, pages 418–427, 2002.

[27] D.W. Scott. *Multivariate Density Estimation*. Wiley, 1992.

[28] R. Sedgewick. *Algorithms in C++*. Addison Wesley, 1992.

[29] J. Seiferas. Bibliography on theory of computer science. In *http://liinwww.ira.uka.de/bibliography/Theory/Seiferas*, 2004.

[30] P. Shenoy, J.R. Haritsa, S. Sudarshan, G. Bhalotia, M. Bawa, and D. Shah. Turbo-charging vertical mining of large databases. In *ACM SIGMOD Intl. Conf. Management of Data*, May 2000.

[31] K. Thulasiraman and M. N. S. Swamy. *Graphs: Theory and Algorithms*. Wiley, 1992.

[32] K. Wang, C. Xu, and B. Liu. Clustering transactions using large items. In *CIKM*, pages 483–490, 1999.

[33] G. Wiederhold. Bibliography on database systems. In *http://liinwww.ira.uka.de/bibliography/Database/Wiederhold*, 2004.

[34] R. J. Wilson. *Introduction to Graph Theory, Fourth Edition*. Addison-Wesley, 1996.

[35] J. Yang, W. Wang, H. Wang, and P.S. Yu. Delta-clusters: Capturing subspace correlation in a large data set. In *Proceedings of the ICDE*, pages 517–528, 2002.

[36] M. J. Zaki and K. Gouda. Fast vertical mining using Diffsets. In *9th ACM SIGKDD Int'l Conf. Knowledge Discovery and Data Mining*, August 2003.

[37] Y. Zhang, A. Fu, C. Cai, and P. Heng. Clustering categorical data. In *Proceedings of the ICDE*, page 305, 2000.