CLICKS: An Effective Algorithm for Mining Subspace Clusters in Categorical Datasets

Mohammed J. Zaki *
Rensselaer Polytechnic Institute, Troy, NY
zaki@cs.rpi.edu

Markus Peters, Ira Assent, Thomas Seidl
RWTH University, Aachen, Germany
{peters,assent,seidl}@informatik.rwth-aachen.de

ABSTRACT

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

Categories and Subject Descriptors: H.2.8 [Database Management]: Database Applications - Data Mining

General Terms: Algorithms.

Keywords: Clustering, Categorical Data, K-partite Graph, Maximal Cliques, Data Mining

1. INTRODUCTION

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

In this paper, we present CLICKS(an anagram of the bold letters in **S**ubspace **CL**uster**I**ng of **C**ategorical data via maximal **K**-partite cliques), a novel algorithm for mining categorical (subspace) clusters. Our main contributions are: 1) We present a novel formalization of categorical clusters. We summarize the dataset as a k-partite graph, and mine maximal k-partite cliques, which after post-processing correspond to the clusters. The k-partite maximal clique mining method is interesting in its own right. 2) CLICKS uses a *selective vertical expansion* approach to guarantee complete search; no valid cluster is missed. It also merges overlapping cliques to report more meaningful clusters. 3) CLICKS addresses the main shortcomings of existing methods. Unlike many previous categorical clustering algorithms, CLICKS can mine subspace clusters. Furthermore, it imposes no domain constraints and is scalable to high dimensions. 4) CLICKS outperforms existing approaches by over an order of magnitude, especially for high-dimensional datasets. These results are demonstrated in a comprehensive performance study on real and synthetic datasets.

*This work was supported in part by NSF CAREER Award IIS-0092978, NSF grants EIA-0103708 & EMT-0432098, and DOE Career award DE-FG02-02ER25538.

KDD'05, August 21–24, 2005, Chicago, Illinois, USA.

2. PRELIMINARIES

Our definition of categorical clusters is based on ideas first proposed in [4]. Let $A_1, \ldots, A_n$ denote a set of *categorical attributes* and $D_1, \ldots, D_n$ a set of *domains*, where $D_i = \{v_{i_1}, \ldots, v_{i_m}\}$ is the domain for attribute A_i, and $D_i \cap D_j = \emptyset$ for $i \neq j$. A *dataset* is a subset of the cartesian product of the attribute domains, given as $\mathcal{D} \subseteq D_1 \times \ldots \times D_n$. The number n of attributes is also referred to as the *dimensionality* of the dataset. An element $\mathbf{r} = (r.A_1, \ldots, r.A_n) \in D$ is called a *record*, where $r.A_i \in D_i$ refers to the value for attribute A_i in $\mathbf{r}$. Each record also has a unique record id (rid), given as $\mathbf{r}.id$.

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

Let $S = S_1 \times \ldots \times S_k$ be a k-subspace, 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 $r.A_{i_j} \in S_j$ for all $j \in [1, k]$. The *support* of S in dataset $\mathcal{D}$ is defined as the number of records in the dataset that belong to it; it is given as $\sigma(S) = |\{\mathbf{r} \in \mathcal{D} : \mathbf{r} \in S\}|$. S is called a *frequent subspace* if $\sigma(S) \geq \sigma^{\min}$, where $\sigma^{\min}$ is some user-defined minimum support threshold. Under attribute independence, the *expected support* of S in $\mathcal{D}$ is given as $E[\sigma(S)] = |\mathcal{D}| \cdot \prod_{j=1}^{k} \frac{|S_j|}{|D_{i_j}|}$. One can incorporate other expected dataset distributions by modifying the definition for $E[\sigma(S)]$. Let $\alpha \in \mathbb{R}^+$. Define a *density indicator* function $\delta_\alpha(S)$ as follows: $\delta_\alpha(S) = 1$ iff $\sigma(S) \geq \alpha \cdot E[\sigma(S)]$, otherwise $\delta_\alpha(S) = 0$. S is called a *dense subspace* iff $\delta_\alpha(S) = 1$, that

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

Table 1: Sample Categorical Dataset

is, if its expected support exceeds its actual support by a user-defined factor α (also called *density threshold*).

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

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

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

EXAMPLE 1. *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. Here $D_1 = \{a_1, a_2, a_3\}, D_2 = \{b_1, b_2, b_3\}, D_3 = \{c_1, c_2, c_3\}$. Let $\alpha = 2.5$. Assuming that attributes and their values are independent the expected support for any pair of values, i.e, the 2-subspace $\{v_i\} \times \{v_j\}$ from different attributes A_i and A_j is $E[\sigma(\{v_i\} \times \{v_j\})] = 1/3 \times 1/3 \times 6 = 0.67$. Thus any pair of values that occurs at least 2 times (i.e, $\sigma(\{v_i\} \times \{v_j\}) = 2$) will be dense (since $2/0.67 = 2.98 > \alpha$). Thus for $\alpha = 2.5$ there is only one full-space cluster in this dataset $(\{a_2\} \times \{b_3\} \times \{c_3\})$, There is an additional subspace cluster: $(\{b_1\} \times \{c_1\})$.*

On the other hand, if we use $\alpha = 1.5$, then any pair of values that occurs once will be considered dense. Thus for $\alpha = 1.5$, there are 3 full-space clusters in this dataset: $(\{a_1, a_2\} \times \{b_1\} \times \{c_1\})$, $(\{a_2, a_3\} \times \{b_3\} \times \{c_3\})$, and $(\{a_2\} \times \{b_3\} \times \{c_2, c_3\})$. There are two additional subspace clusters: $(\{a_2\} \times \{b_1, b_3\})$, and $(\{a_2\} \times \{c_1, c_2, c_3\})$. Note that an interesting property of our approach is that the clusters found for higher α will always be contained in a lower α, which can allow the user to produce a cluster hierarchy.

3. RELATED WORK

Full-space clustering has been an active research topic for a long time; more recently, since CLIQUE [1] introduced the problem, many subspace clustering techniques have been also been proposed. We focus here only on the relevant research in categorical clustering.

K-modes [9] is an extension to the k-means numeric clustering algorithm. COOLCAT [3] is based on the idea of entropy reduction within the generated clusters. LIMBO [2] is a recent information theoretic clustering based on the information-bottleneck framework. STIRR [5] uses a non-linear dynamical systems approach to categorical clustering. It encodes the dataset into a weighted (with attribute-values as vertices) graph and iteratively propagates these weights until convergence to "basins". The main weakness of STIRR is that the separation of attribute values by their weights is non-intuitive and the post-processing required to extract the actual clusters from the basin weights is non-trivial. ROCK [7] uses an agglomerative hierarchical clustering approach, using the number of "links" (i.e., shared similar records) between two records as the similarity; it has $O(|\mathcal{D}|^3)$ complexity, making it unsuitable for large problems. CACTUS [4] first builds a summary information from the dataset and it then computes cluster projections onto the individual attributes and then extends them to find cluster candidates over multiple attributes. The extension step, though described in the paper, was not implemented by the authors, but our augmented implementation showed a severe performance impact over the cactus baseline version (see Section 5.1). Note also that with the exception of CACTUS, which mines only a limited class of subspace clusters, none of the previous methods can mine subspace clusters.

Other previous work has focused on binary or transactional data [13, 8]. Also relevant is the problem of biclustering [11], which aims at finding subspace clusters on both attributes and records. However, these methods cannot typically handle the kinds of clusters we propose here (e.g.,$(\{a_1, a_2\} \times \{b_1\} \times \{c_1\})$ as shown in Example 1), since by definition two values of the same attribute never co-occur in any transaction, ans as such they will not be part of the same cluster.

4. THE CLICKS APPROACH

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

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

LEMMA 2. *Given a categorical dataset $\mathcal{D}$ and a k-subspace $C = C_1 \times \ldots \times C_k$ with $C_j \subseteq D_{i_j}$ over attributes $A_{i_1}, \ldots, A_{i_k}$. C is a k-cluster in $\mathcal{D}$ if and only if C is a maximal, dense k-partite clique in $\Gamma_{\mathcal{D}}$.*

EXAMPLE 3. *Consider the example in Table 1. Let $\alpha = 2.5$, then any pair of values that occurs at least 2 times is dense in $\mathcal{D}$, and thus there is an edge between such vertices in $\Gamma_{\mathcal{D}}$. The corresponding k-partite graph of $\mathcal{D}$ is shown in Figure 1, using bold edges. It clearly has two clusters, one full-space and one sub-space. If $\alpha = 1.5$, then some other (thin) edges will be added to the graph. Mining this new graph will produce the larger set of clusters mentioned in Example 1. It should be clear that clusters mined at $\alpha = 2.5$ are contained in those at $\alpha = 1.5$, since a lower α only* adds *edges to $\Gamma_{\mathcal{D}}$.*

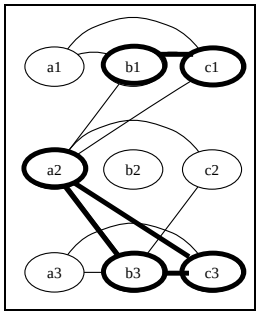

Figure 1: k-Partite Graph of $\mathcal{D}$

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

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

  PreProcess(D, α, Γ_D, R)
  DetectMaxCliques(Γ_D, L, R, C)
  PostProcess(D, L, α, σ^C)
  return L
```

Figure 2: The CLICKS Algorithm

The basic CLICKS approach consists of the three principal stages, shown in Figure 2, as follows: 1) *Pre-processing*: We create the k-partite graph from the input database $\mathcal{D}$. We also rank the attributes for efficiency reasons. 2) *Clique Detection*: We enumerate all the maximal k-partite cliques in the graph $\Gamma_\mathcal{D}$. 3) *Post-processing*: We verify the support of the candidate cliques within the original dataset to form the final clusters. If completeness is desired we perform selective sub-clique expansion of non-dense maximal cliques to find the true maximal, dense cliques. Moreover, the final clusters are optionally merged if they have significant overlap.

4.1 Pre-processing

In one scan of the dataset, CLICKS collects the support of all single and pairs of attribute values. From the pairs it computes the dense pairs $\{v_a\} \times \{v_b\}$, and add an edge (v_a, v_b) to $\Gamma_\mathcal{D}$, creating the full k-partite graph $\Gamma_\mathcal{D}$.

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

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

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

Intuitively, connectivity corresponds to the neighbors ($N(v_j)$) plus the remaining values of the attribute in question ($D_i \backslash v_j$). However, if a given value does not co-occur with values of other attributes it cannot be part of a k-partite clique; its connectivity should be zero. The connectivity of a clique C is given as follows: $\eta(C) = \bigcap_{v_j \in C} \eta(v_j)$, i.e, the connectivity common to all vertices in C. CLICKS ranks the set of all attribute values by decreasing connectivity for efficient clique enumeration. Given a seed clique, CLICKS adds a new vertex to the clique based on the next highest ranked value. This can significantly speed-up the search for maximal cliques.

4.2 Enumerating k-partite Maximal Cliques

The clique detection phase is based on a backtracking search, that at each step, adds only those vertices to a clique that are in the connectivity set of the clique. If more than one such vertex exists, attribute value ranking is used to break the tie. CLICKS uses a recursive algorithm that at each stage tries to expand the current clique to ensure maximality. It is similar in spirit to the Bron-Kerbosch (BK) algorithm [10], but whereas BK enumerates regular cliques, CLICKS is designed for k-partite cliques. The pseudo-code for the clique detection phase is shown in Figure 3.

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

Note that `foreach` statements process attribute value rankings (based on connectivity) in descending order. The predicate $\Phi(C)$ evaluates to *true* iff i) we want to mine subspace clusters, or ii) we want to mine full space clusters and C contains at least one attribute value for every attribute of the dataset. Otherwise $\Phi(C)$ is *false*. The set $\mathcal{R}^D$ 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(Graph Γ_D, CliqueList L,
        AttributeValueRanking R, Clique C)
1.  if (R = ∅) then
2.      if (η(C) = ∅ and Φ(C)) then L = L ∪ C
3.      return
4.  R^D = R^P = ∅
5.  foreach v in R − R^D − R^P do
6.      C' = C ∪ {v}; R' = ∅;
7.      R^D = R^D ∪ {v}
8.      foreach v' in R − R^D do
9.          if (v' ∈ η(v)) then R' = R' ∪ {v'}
10.     if (v is first value in R) then R^P = R'
11.     if (Φ(R' ∪ C')) then
12.         DetectMaxCliques(Γ_D, L, R', C')
```

Figure 3: The CLICKS Clique Detection

`DetectMaxCliques` starts by checking if the current clique C is maximal (lines 1-2). If $\mathcal{R} = \emptyset$ then there are no more elements to extend C, thus C is potentially maximal. If in ad-

dition $\eta(C) = \emptyset$ then C is a maximal clique, since an empty connectivity set means there are no additional vertices connected to all vertices in C. The only test that remains to be done is whether full/sub-space cliques are desired. For subspace clusters $\Phi(C)$ is always true, whereas for full-space clusters $\Phi(C)$ is true only if C contains at least one value from each attribute. Thus, C is added to the set of maximal cliques $\mathcal{L}$ iff $\Phi(C)$ is true (line 2), and the search is continued at the previous level (line 3).

If $\mathcal{R} \neq \emptyset$ then C can potentially be extended with vertices in $\mathcal{R}$. The outer loop (line 5) attempts to add a value v to C in an effort to create a yet larger clique C' (line 6). Note also that at any given point in time $\mathcal{R}$ contains only those attribute values that are connected to C. Hence, adding $v \in \mathcal{R}$ to C will yield another clique C'. We mark v as deleted ($\mathcal{R}^D = \mathcal{R}^D \cup \{v\}$), indicating that it was already considered in the clique construction (line 7).

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

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 we can prune part of the search space at this point; we can stop the recursion if the new clique C' cannot be extended to cover at least one value from all attributes, i.e, we recurse only if $\Phi(\mathcal{R}' \cup C')$ is true (lines 11-12).

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 form a dense 2-subspace 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 $\mathcal{R}^P$.

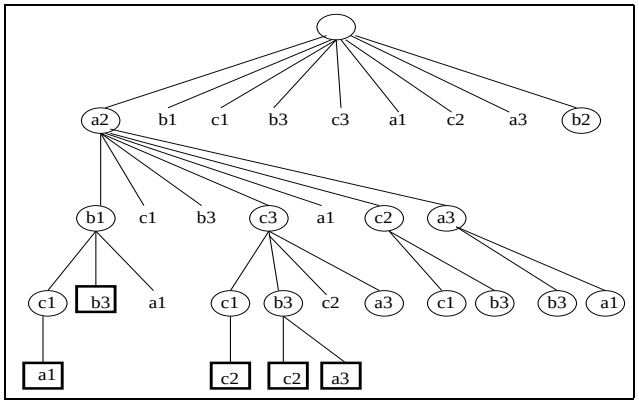

Figure 4: Clique Finding

EXAMPLE 4. *Consider the k-partite graph encoding $\Gamma_{\mathcal{D}}$ in Figure 1. An attribute value ranking of V is as follows: $a_2(7)$, $b_1(6)$, $c_1(6)$, $b_3(6)$, $c_3(5)$, $a_1(4)$, $c_2(4)$, $a_3(4)$, $b_2(0)$, where the connectivity cardinalities $|\eta(v)|$ are given in parentheses. Figure 4 shows a run of* `DetectMaxCliques` *on this example. Vertices depicted without circles denote search paths that were pruned due to $\mathcal{R}^P$, whereas bold squares indicate that a maximal clique was found. By following the edges up to the root we can construct the corresponding cliques. The $\mathcal{R}'$ sets can be read from the figure by computing the union of all children of a node. For example, $\mathcal{R}'$ for clique $\{a_2, b_1\}$ (in the leftmost path) is $\{c_1, b_3, a_1\}$. This example shows all five full and subspace maximal cliques. For example $\{a_2, b_1, c_1, a_1\}$ is a full space clique, whereas $\{a_2, b_1, b_3\}$ is a subspace clique.*

4.3 Post-Processing

Once the set of all the maximal k-partite (or n-partite) cliques $\mathcal{L}$ have been mined, the post-processing phase involves a single scan of the dataset to count, for each candidate clique $C \in \mathcal{L}$, the number of transactions in the dataset that support it. If $\delta_\alpha(C) = 1$, i.e, the support of C is at least α times its expected support, then C is a valid clique, and CLICKS outputs it as a cluster. However, there are two challenges that remain: 1) a maximal clique may fail the density test, whereas one of its sub-cliques may be dense. To guarantee completeness, CLICKS allows an optional *selective vertical expansion* approach to explore the sub-cliques induced by a non-dense maximal clique; we give more details of this step in the next section. 2) There may be many overlapping cliques in $\mathcal{L}$. In this case, it is desirable to merge those cliques that have significant overlap into large cliques; we give details of this step below.

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

The enhanced post-processing step in CLICKS implements a novel method for merging a set of discovered maximal cliques based on their common *coverage*, i.e., the number of records that are common to that set of cliques. Let $\mathcal{L}$ be the set of maximal cliques mined from $\Gamma_{\mathcal{D}}$. For every clique $C^i \in \mathcal{L}$ let i denote its unique clique *id*. We define the term *cset* to denote any set of clique ids. Let $\mathcal{C}$ denote the database of csets obtained by replacing each record $\mathbf{r} \in \mathcal{D}$ with its cset, the set of clique ids that the record belongs to, given as $cset(\mathbf{r}) = \{i : \mathbf{r} \in C^i\}$. We can then mine the cset database $\mathcal{C}$ to obtain all the maximal frequent csets, denoted as $\mathcal{F}_{\mathcal{C}}$, that are above a minimum frequency threshold $\sigma^{\mathcal{C}}$, i.e., those csets that are co-supported by a minimum number of records. Note that $\mathcal{F}_{\mathcal{C}}$ can be efficiently mined using any maximal itemset mining method (such as GenMax[6]). For example, consider Table 1. Let $C^1 = \{a_2\} \times \{b_3\}$ and $C^2 = \{b_3\} \times \{c_3\}$ be the only maximal cliques in $\Gamma_{\mathcal{D}}$. Then the cset database $\mathcal{C}$ is given as: $\mathcal{C} = \Big\{\{\}, \{1\}, \{1,2\}, \{\}, \{1,2\}, \{2\}, \ldots\Big\}$. Mining $\mathcal{C}$ with minimum support $\sigma^{\mathcal{C}} = 2$ yields the maximal cset $\{1,2\}$ suggesting that cliques C^1 and C^2 should be merged into one clique: $\{a_2\} \times \{b_3\} \times \{c_3\}$.

Every cset $X \in \mathcal{F}_{\mathcal{C}}$ is a potential set of cliques that can be merged. However $\mathcal{F}_{\mathcal{C}}$ may itself have overlapping maximal csets, and of various sizes. Clearly we need a ranking of csets so that merging can be done in a systematic manner. A good ranking measure is the coverage, i.e., the number of records in $\mathcal{D}$, that belong to the clique obtained after merging all cliques ids in a given cset. Unfortunately, computing the coverage for each $X \in \mathcal{F}_{\mathcal{C}}$ can be very expensive, since it would require multiple scans of the original database $\mathcal{D}$. Instead, we introduce an approximate measure

of coverage, called *coverage weight*, that does not need to access the original database $\mathcal{D}$; it uses the clique support already computed from $\mathcal{D}$ in the validation step, and cset support computed while mining $\mathcal{F}_\mathcal{C}$. Intuitively, the coverage weight is an approximation of the inclusion/exclusion computation for the supporting records. More formally, given any $X \in \mathcal{F}_\mathcal{C}$, where $X = \{1, \cdots, m\}$ is a set of clique ids (corresponding to cliques $\{C^1, \ldots, C^m\}$) that frequently occur together, its *coverage weight* is defined as $\omega(X) = \left(\sum_{i=1}^{m} \sigma_\mathcal{D}(C^i)\right) - (m-1) \times \sigma_\mathcal{C}(X)$, where $\sigma_\mathcal{C}(X)$ denotes X's support within $\mathcal{C}$. For merging decisions, all csets in $\mathcal{F}_\mathcal{C}$ are sorted in decreasing order of their coverage weight.

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

Figure 5: CLICKS **Post Processing**

Figure 5 shows the pseudo-code for the post-processing phase, including the merging steps. After validating the set of mined maximal cliques $\mathcal{L}$, by counting their support and computing their density (line 1), we call selective vertical expansion if needed (line 2). This is followed by transforming the dataset $\mathcal{D}$ into the cset database $\mathcal{C}$, which is mined at minimum support $\sigma^\mathcal{C}$ to obtain all maximal frequent csets $\mathcal{F}_\mathcal{C}$, using GenMax [6] (line 3). This set is sorted in decreasing order of coverage weight to obtain a total order (denoted by the relationship $>_\omega$) on all maximal csets (line 4).

We then process each set $X \in \mathcal{F}_\mathcal{C}$ in order of $>_\omega$ (line 6); X is added to $\mathcal{F}^P$ (line 7) the set of processed csets, that give the clique ids of cliques to be merged in the end. Since no clique can be merged twice, all clique ids that occur in X have to be removed from the not-yet-processed csets $Y >_\omega X$ (lines 8-9). Finally, we create the final set of merged clusters $\mathcal{L}$ by iterating through each cset $Z \in \mathcal{F}^P$ (line 12), and and merging the cliques accordingly (line 13). Before merging, we make a copy of $\mathcal{L}$ in $\mathcal{L}'$ (line 10), so that we can merge cliques identified by Z by looking at $\mathcal{L}'$ (line 13), whereas $\mathcal{L}$ contains only the final set of cliques. If a merged clique M (line 14) is potentially dense we add it to the final set of clusters (line 15).

4.4 Selective Vertical Expansion

All maximal cliques $\mathcal{L}$ are reported by the clique detection phase, but some might be pruned out in post-processing if they are non-dense; let $\mathcal{L}^P \subseteq \mathcal{L}$ denote all such pruned cliques. Sub-cliques of a pruned clique, however, might have required density. In such cases, to guarantee completeness CLICKS uses the *selective vertical expansion* approach to consider all sub-cliques for each $C \in \mathcal{L}^P$, and reports all the remaining maximal, dense cliques.

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

The selective vertical expansion step uses ridsets to explore all sub-cliques for each $C \in \mathcal{L}^R$. Starting from the ridsets for the single values in C, we build larger sub-cliques using a series of union and intersection operations on the corresponding ridsets. The search stops when we have found all maximal dense sub-cliques contained within C. For each such sub-clique, if it is not contained within an already found maximal dense clique in $\mathcal{L} \backslash \mathcal{L}^P$, we output it as a true maximal, dense clique.

5. EXPERIMENTAL STUDY

This section presents a comparative study of CLICKS versus CACTUS [4] and other methods like ROCK [7] and STIRR [5]. All testing was done on a hyper-threaded Intel Xeon 2.8GHz with 6GB of RAM, running Linux. The code for CACTUS was obtained from its authors.

All synthetic datasets used in our experiments were created using the generation method proposed in [5]. The generator creates a user specified number of records that are uniformly distributed over the entire data space. It allows for specification of the number of attributes and the domain size on each attribute. The generator then injects a user specified number of additional records in designated cluster regions, thus increasing the support of these regions above their expected support.

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

5.1 CLICKS vs. ROCK and CACTUS

We first compare CLICKS with ROCK [7]. Even though ROCK assumes that inter-point similarities are given, Figure 6 shows that CLICKS still outperforms ROCK by orders of magnitude. Since ROCK is too slow, we only compare CLICKS with STIRR and CACTUS below.

We next compare with CACTUS. As noted earlier, the available CACTUS implementation stops at the cluster projection step, but does not extend these to produce the final clusters. Note that the reported performance in [4] focuses only on I/O cost, and does not account for CPU cost of extension and validation, since they were mainly interested in showing the effectiveness of the small data summaries, even for very large datasets. To study the impact of these additional steps, we augmented CACTUS with the cluster extension and validation steps.

Figure 7 shows the running time of CACTUS with and without the additional steps, on datasets with up to 500,000 tuples. We see that CACTUS with extensions is about 3 times slower than the base-line version, and the gap is increasing. This impact is largely due to the excessive number of projections that CACTUS generates. In the remaining

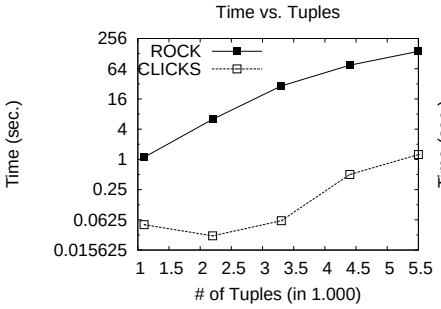

Figure 6: CLICKS **vs. ROCK**

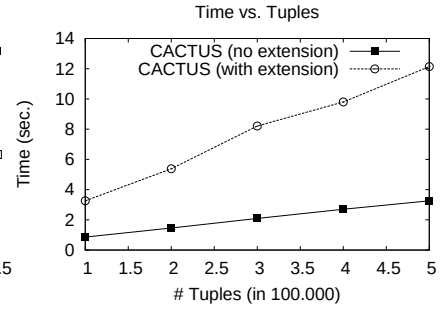

Figure 7: CACTUS Extension

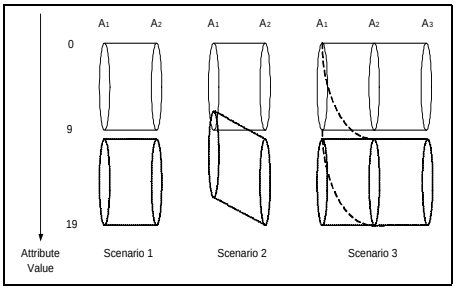

Figure 8: Cluster Quality Data

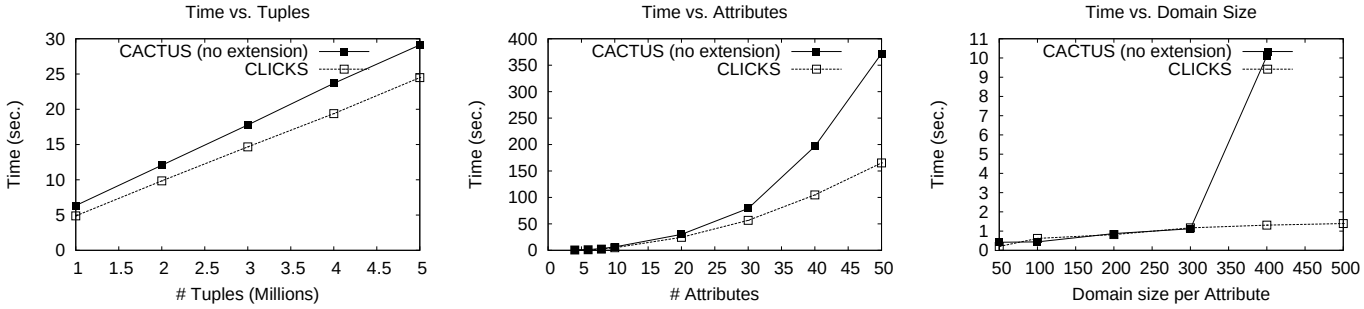

Figure 9: CLICKS **vs. CACTUS (no extensions)**

performance studies only the base-line CACTUS version is used, since the version with extensions is too slow to be run on larger datasets.

Three tests on synthetic datasets were performed to compare the performance of CLICKS and CACTUS w.r.t. number of records, number of attributes, and domain size, as shown in Figure 9: 1) **Dataset Size:** Synthetic datasets with 10 attributes, and 100 values per attribute were used, while the total number of records was varied from one to five million. Both methods scale linearly over the number of records in the dataset, but CLICKS outperforms CACTUS (base-line) by an average of 20%. If we take CACTUS with extensions into account CLICKS is at least 3-4 times faster. 2) **Dimensionality:** CLICKS is especially scalable w.r.t. dimensions. On a dataset with 1 million records and 100 attribute values per dimension, CLICKS outperforms CACTUS (baseline) by a factor 2 - 3 (and thus CACTUS (extension) by at least a factor of 6-9), when varying the number of attributes from 10 to 50, and the gap is increasing. 3) **Domain size:** Datasets with one million records and four attributes were used to measure the performance w.r.t domain size. The number of attribute values per attribute were varied from 50 to 500. Both methods perform equally well for less than 400 attribute values per domain. At this point, the runtime of CACTUS dramatically increases, most likely due to memory shortage. For large domains, CLICKS is thus over an order of magnitude faster than CACTUS.

The STIRR [5] algorithm, as implemented by Ganti et al. was also benchmarked. STIRR outputs the non-principal basins, i.e, weighted vertices, that identify the cluster projection on each attribute. As in the case of CACTUS, no clusters are actually output. However, it seems clear that the final cluster extraction step in STIRR would cost at least as much as the extension step in CACTUS.

5.2 Cluster Quality: Synthetic Data

To evaluate the quality of the clusters, three basic scenarios were tested on synthetic datasets, with $\alpha = 3$, and with post-processing turned off, in order to verify the actual reported cliques before merging. The datasets, as shown in 8, contained $105,000$ records in scenarios one and two. In scenario 3, $110,000$ records were used, reflecting additional 5000 records in the third cluster. In all scenarios, attributes have 100 values, but clusters are embedded only on the ranges drawn. For example, scenario 1 has two attributes with two well separated clusters, one on attribute values $[0-9]$ and another on $[10-19]$ on A_1 and A_2; there are no clusters on attribute values $[20-99]$ even though there are points in the dataset, chosen uniformly at random, in that range.

Scenario 1: used a dataset with clear separation of the two clusters on ranges $[0-9]$ and $[10-19]$. CLICKS detected both the clusters on the appropriate attribute values. The CACTUS implementation reported 240 cluster projections per attribute. These represented all subsets of size 3 of $\{0,\ldots,9\}$ and $\{10,\ldots,19\}$. They are part of the cluster projection but do not satisfy the maximality condition. Our CACTUS extension connected all subsets of $\{0,\ldots,9\}$ on the first attribute with the corresponding subsets on the second attribute. Similarly, all subsets of $\{10,\ldots,19\}$ were connected on both attributes, yielding $115,200$ clusters (reflecting the lack of maximality of the projections). The STIRR algorithm reported weights of about 0.15 for the attribute values $[0,19]$ on both attributes, while the weights of the attribute values in $[20,99]$ were computed to be about 0.08. According to the interpretation in [5] this corresponds to a single cluster on $[0,19] \times [0,19]$, confirming the lack of separation found in [4].

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

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

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

5.3 Post-Processing and Selective Expansion

We also assessed the effectiveness of the post-processing and selective vertical expansion steps. Due to space limitations, we only give the conclusions. In the post-processing step, we found that the merging time was not affected by the chosen $\sigma^{\mathcal{C}}$ value, since only a small fraction of the total execution time is spent validating and merging clusters. We also found that selective expansion can be between 2-5 times slower than the baseline method, mainly due to the building of ridsets. The added overhead of selective mining was at most linear (in the number of records) w.r.t. the base-line, making it a computationally viable option even for large datasets. Most importantly, even *with* vertical mining enabled CLICKS is faster than other methods on many datasets.

5.4 Real Dataset

We applied CLICKS on several real datasets, but here we present results only on Mushroom; see [12] for more details.

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

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

Table 2: Mushroom: Confusion Matrix (Full Space)

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

Table 3: Mushroom: Confusion Matrix (Subspace)

CLICKS was configured to run with a low α value of 0.4 as the dataset is very sparse. Not surprisingly, many of the candidate clusters were overlapping. By assigning each record to the first cluster that contains it, the confusion matrices shown in Tables 2 and 3 were generated for full dimensional and subspace clustering, respectively. The two rows represent the two original classes, poisonous(P) and edible (E), while the columns represent the clusters that CLICKS generated. Each cell records the percentage of points that belong to that cell (e.g., in Table 2, 21.3% of all points belong to cluster C_2 and have the label 'P'). Note, that the class attribute was not used in the clustering.

Full-dimensional clustering initially yielded 256 candidate clusters which were then reduced to 213 clusters using a $\sigma^{\mathcal{C}}$ value of 0.5% for the post-processing step. Out of the 213 total clusters, 14 of them account for 87.1% of all points; these clusters with highest support values are shown explicitly (C_1 to C_{14}) in Table 2, while the other smaller clusters are grouped under the column *Others*. Note that these smaller clusters account for only 4% of all points, and thus one can safely discard these clusters to yield 14 useful full-dimensional clusters which show perfect purity w.r.t. the class label. About 9% of the records could not be grouped (column *None*) in any cluster.

For the subspace case CLICKS produced 596 initial clusters (including full-space clusters). This number was reduced to 553 by merging with a $\sigma^{\mathcal{C}}$ setting of 5%. As in the full dimensional case, a large number of clusters overlapped. By assigning each record to the first cluster that contains it, Table 3 was obtained. All points can be covered by only 19 clusters, of which $C_1 - C_{14}$ are full-dimensional (same as before), and there are five new subspace clusters $C_{15} - C_{19}$. The subspace clusters clearly improved the result w.r.t. the unclustered records, since all records are now covered by some cluster (the *None* column has 0% of points, as opposed to 8.9% in the full dimensional case). Cluster C_{17} and C_{18} show minor impurities (less than 1%) w.r.t. the class label. By using all 553 clusters a perfectly pure clustering is obtained. However, this level of granularity will be inappropriate for most applications.

Acknowledgment: We would like to thank Venkatesh Ganti for providing the source code for CACTUS and STIRR, and Eui-Hong "Sam" Han for the ROCK implementation.

6. REFERENCES

[1] R. Agrawal, J. Gehrke, D. Gunopulos, and P. Raghavan. Automatic subspace clustering of high dimensional data for data mining applications. In *SIGMOD Conf.*, 1997.

[2] P. Andritsos, P. Tsaparas, R. J. Miller, and K. C. Sevcik. LIMBO: Scalable clustering of categorical data. In *9th Int'l Conf. on Extending DataBase Technology*, March 2004.

[3] D. Barbara, Y. Li, and J. Couto. Coolcat: an entropy-based algorithm for categorical clustering. In *CIKM Conf.*, 2002.

[4] V. Ganti, J. Gehrke, and R. Ramakrishnan. CACTUS: Clustering categorical data using summaries. *SIGKDD*, 1999.

[5] D. Gibson, J. Kleinberg, and P. Raghavan. Clustering categorical data: An approach based on dynamical systems. In *VLDB Conf.*, August 1998.

[6] K. Gouda and M. J. Zaki. Efficiently mining maximal frequent itemsets. In *ICDM Conf.*, November 2001.

[7] S. Guha, R. Rastogi, and K. Shim. Rock: A robust clustering algorithm for categorical attributes. In *ICDE Conf.*, 1999.

[8] E. Han, G. Karypis, V. Kumar, and B. Mobasher. Clustering based on association rule hypergraphs. In *SIGMOD DMKD Workshop*, 1997.

[9] Z. Huang. Extensions to the k-means algorithm for clustering large data sets with categorical values. *Data Mining and Knowledge Discovery*, 2(3), 1998.

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

[11] S. C. Madeira and A. L. Oliveira. Biclustering algorithms for biological data analysis: a survey. *IEEE/ACM Trans. on Computational Biology and Bioinformatics*, 1(1):24–45, 2004.

[12] M. Peters and M. J. Zaki. CLICK: Clustering categorical data using k-partite maximal cliques. CS TR 04-11, RPI, 2004.

[13] K. Wang, C. Xu, and B. Liu. Clustering transactions using large items. In *CIKM Conf.*, 1999.