MAXIMAL CLIQUES AND VERTICAL MINING FOR CLUSTERING CATEGORICAL DATA

By

Markus Peters

A Thesis Submitted to the Graduate

Faculty of the Aachen University of Technology

in Partial Fulfillment of the

Requirements for the Degree of

DIPLOM INFORMATIKER

Advisers

Dipl.-Inform. Ira Assent, Prof. Dr. Thomas Seidl
(Chair IX for Computer Science, RWTH Aachen)

Prof. Mohammed J. Zaki, Ph.D.
(Rensselaer Polytechnic Institute, Troy, NY)

RWTH Aachen

September 2004

CONTENTS

LIST OF TABLES

LIST OF FIGURES

ACKNOWLEDGEMENT

Ira Assent and Thomas Seidl provided advice for the work on the second edition of this thesis. I greatly appreciate their support.

I would like to thank Mohammed J. Zaki for his numerous suggestions and his efforts to critically review the first edition of this work.

Venkatesh Ganti kindly provided the source code for CACTUS and STIRR, and Eui-Hong "Sam" Han provided the ROCK implementation.

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 CLICKS, which finds clusters in categorical datasets based on a search method for k-partite maximal cliques. CLICKS 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 [26, 27]. 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 1.1 shows an example of a two dimensional dataset. One can intuitively identify two dense regions in the figure that should make up clusters.

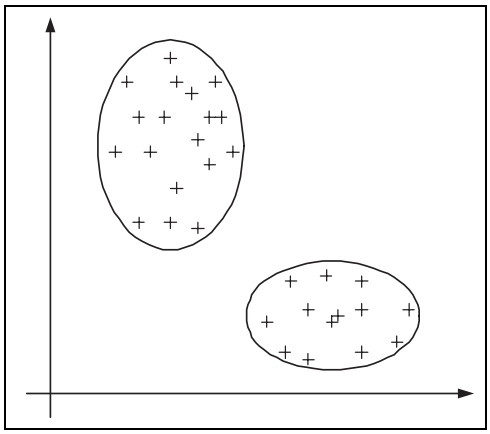

Figure 1.1: Clustering Example

It is natural to approach this problem by computing the similarities between individual data points in the dataset and partitioning the data points so that 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. As an example for a categorical attribute consider the color of a person's hair. Clearly, the number of possible attribute values is meaningfully chosen to be a set like $\{blond, red, brown, black\}$. But unless the concrete application suggests so, there is no natural order on these colors.

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 Datasets with categorical values are frequently high dimensional.

Subspace Clusters Many categorical datasets do not exhibit clusters over the full set of dimensions. This imposes a problem when "noise" in the remaining dimensions distorts distances between clustered points.

The remainder of this chapter will give an introduction to the overall problem of clustering, as well as the specifics of categorical clustering. It then continues 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 1.1.1 (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 1.1.1 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, an attribute 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. However, in certain cases, numerical attributes can meaningfully be treated as categorical. Example include tax classes, identifier numbers, and boolean attributes. 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 1.1.2 (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. Standard works, such as [27], address these issues in a general setting, while section 1.2 and chapter 3 present a co-occurrence based approach for categorical clustering.

1.2 Categorical Clustering

The categorical clustering problem can be formulated as follows [17]: Let $\mathcal{D}$ be a dataset over $A_1, \ldots, A_n$, and $D_1, \ldots, D_n$ the associated categorical domains.

Definition 1.2.1 (k-subspace) *A set $S_i \subseteq D_i$ is called an* interval *over attribute A_i. A k-*subspace *is a set $S = S_{i_1} \times \ldots \times S_{i_k}$ over some subset of k attributes $A_{i_1}, \ldots, A_{i_k}$. If $k = n$, then S is called a* full-space.

Clusters can informally be understood as especially *dense* k-subspaces 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 k-subspace 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.

In the absence of other information about the distribution of the dataset two assumptions are made.

Uniform distribution. The values of each attribute are assumed to be uniformly distributed. That is, for an attribute A_i with domain D_i, each attribute value $v_i \in D_i$ is expected to occur in $\mathcal{D} \times \frac{1}{|D_i|}$ records in $\mathcal{D}$.

Attribute independence. The attribute distributions are pairwise independent, i.e. for $A_i, A_j, i \neq j$ and attribute values $v_i \in D_i, v_j \in D_j$ the two values are expected to occur together in $\mathcal{D} \times \frac{1}{|D_i|} \times \frac{1}{|D_j|}$ records in $\mathcal{D}$.

Note, that the expected support notion is easily modified to capture any existing a priori information about the distribution of the attributes of $\mathcal{D}$, and potential dependencies between them.

Definition 1.2.2 (Support) *Let S 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* [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*

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

[1] if and only if

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
7	a_2	b_1	c_1
8	a_1	b_1	c_1
9	a_3	b_2	c_1
10	a_2	b_1	c_1

Table 1.1: Sample Categorical Dataset

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

Definition 1.2.3 (Density indicator) *Let $S = S_{i_1} \times \ldots \times S_{i_k}$ be a k-subspace. Define a* density indicator *function $\sigma^*_\alpha(S)$ as follows*

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

where $\alpha \in \mathbb{R}^+$ is a user defined real number. For brevity, S is said to be dense *w.r.t α when $\sigma^*_\alpha(S) = 1$. The user threshold α is omitted where it is clear from the context.*

Example 1.2.4 *Consider the dataset shown in Table 1.1 with a total of three categorical attributes A_1, A_2, A_3 and ten records. One can infer that $D_1 = \{a_1, a_2, a_3\}$, $D_2 = \{b_1, b_2, b_3\}$, and $D_3 = \{c_1, c_2, c_3\}$.*

- *$S_1 = \{a_1, a_2\}$ is an* interval *over A_1.*
- *$S = \{a_1, a_2\} \times \{b_1\} \times \{c_1\}$ is a* 3-subspace *or* full-space *over $\{A_1, A_2, A_3\}$.*

- *The support $\sigma(S)$ is 5 (records 1, 4, 7, 8, 10).*
- *This is higher than the expected support*

$$E[\sigma(S)] = |\mathcal{D}| \cdot \prod_{i=1}^{n} \frac{|S_i|}{|D_i|} = 10 \times \frac{2}{3} \times \frac{1}{3} \times \frac{1}{3} = \frac{20}{27}$$

- *Hence, $\sigma^*_\alpha(S) = 1$ for all $\alpha < \frac{5 \times 27}{20} = 6.75$.*

Definition 1.2.5 (Strongly Connected Subspaces) *Let $S = S_{i_1} \times \ldots \times S_{i_k}$ be a k-subspace.*

- *For any pair of values $v_i \in S_i$ and $v_j \in S_j$, v_i and v_j are* strongly connected *iff $(\{v_i\} \times \{v_j\})$ is dense, i.e., the 2-subspace $\{v_i\} \times \{v_j\}$ has support at least α times its expected support.*
- *The attribute value v_i is strongly connected to interval S_j iff $(\{v_i\} \times \{v_j\})$ is dense for all $v_j \in S_j$.*
- *The intervals S_i and S_j are strongly connected iff $(\{v_i\} \times \{v_j\})$ is dense for all $v_i \in S_i, v_j \in S_j$.*

With the definition of strong connectivity on hands, a cluster is defined as a maximal dense set of attribute values that are pairwise strongly connected.

Definition 1.2.6 (Categorical Cluster) *Let $C_i \subseteq D_i$ for $i \in \{1, \ldots, n\}$, and $\alpha > 0$. The k-subspace $C = (C_{i_1} \times \ldots \times C_{i_k})$ is a* (subspace) cluster *over attributes $A_{i_1}, \ldots, A_{i_k}$ iff it satisfies:*

1. **Density:** *$(C_{i_1} \times \ldots \times C_{i_k})$ is dense.*
2. **Connectedness:** *All pairs of intervals $C_i, C_j \in C$ $(i \neq j)$ are strongly connected.*
3. **Maximality:** *For all $i, j \in \{1, \ldots, n\}, i \neq j$ there is no $C'_i \supset C_i$, 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.

Example 1.2.7 (Continued) *Consider again the dataset from Table 1.1 and the 3-subspace $S = \{a_1, a_2\} \times \{b_1\} \times \{c_1\}$.*

- *Say, $\alpha = 1.5$. Then all pairs of intervals in S are strongly connected. For example $(\{a_1\} \times \{b_1\})$ has support 2 where its expected support is*

$$10 \times \frac{1}{3} \times \frac{1}{3} = 1.1$$

Since $2 > \alpha \times 1.1$,

$$\sigma^*_\alpha(\{a_1\} \times \{b_1\}) = 1$$

(analogous for other combinations).

- *S is a categorical cluster over $\{A_1, A_2, A_3\}$: It is dense and all pairs of intervals are strongly connected. Also, no super-cluster S' with $S \subseteq S'$ exists.*

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.

1.3 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. An example of a categorical similarity measure is the Simple

Table 1.2: Sample Document Encoding

	d_1	d_2
Document	(Algorithm Complexity)	(State Transition)
Encoding	(1, 0, 1, 0, 0, 0, 0)	(0, 0, 0, 0, 1, 1, 0)

Matching Coefficient (SMC) [27]. For two records r_1 and r_2 from a d-dimensional categorical dataset $\mathcal{D}$ the SMC is defined as

$$SMC(r_1, r_2) = \frac{m(r_1, r_2)}{d}$$

where m indicates the number of matching attribute values, i.e. the number of index values $i \in \{1, \ldots, d\}$ where $r_1.A_i = r_2.A_i$.

Secondly, categorical datasets are frequently high dimensional. Clearly, high dimensionality is not an immediate consequence. However, practical examples suggest that clustering approaches for categorical data should be highly scalable in terms of number of attributes. Moreover, it can be shown that traditional distance measures become ineffective in the presence of high-dimensional datasets, a phenomenon known as the *curse of dimensionality* [7, 39].

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.

As an example, consider clustering a set of text documents. A natural categorical representation of a text document is a vector of bits that indicate the presence or absence of dictionary terms in a document. Suppose a collection of computer science documents is based on the dictionary

$$D = (\text{Algorithm,Cluster,Complexity,Mining,State,Transition,Turing})$$

Table 1.2 shows sample encodings for two documents. It is conceivable that two clusters "Data Mining papers" and "Computational Complexity papers" exist

on the attributes (Cluster,Mining) and (State,Transition,Turing), respectively. However, most likely no clusters will be found over the full dictionary as the common terms *Algorithm* and *Complexity* will periodically appear in documents from both clusters and distort the similarity information for documents in full-dimensional space.

If, for example, the SMC is used to measure document similarity to a prototypical cluster centroid c = (0,1,0,1,0,0,0) (Cluster, Mining), the document encoding e_1 = (1,1,1,1,0,0,0) (Algorithm, Cluster, Complexity, Mining) is less similar to c than e_2 = (0,1,0,1,0,0,0) (Cluster, Mining) even though the terms *Algorithm* and *Complexity* should not have an effect on the outcome of the clustering process. The values of these attributes are random with respect to the final clustering, and can hence be considered *noise*.

Document clustering is a prime example of a very high dimensional clustering task. Typical dictionaries over moderately sized text collections easily contain 5000 distinct entries or more.

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 CLICKS [2], which finds clusters in categorical datasets based on a search method for k-partite maximal cliques. CLICKS 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. An extension to CLICKS is presented in chapter 4. Here, a vertical encoding similar to the vertical encoding in frequent itemset mining is used to restore the completeness

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

of the algorithm. The performance characteristics of both, the vertical and the non-vertical method, are studied in chapter 5 in a comprehensive performance evaluation on real and synthetic datasets. Finally, chapter 6 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 [26], only a limited number of studies have focused on categorical clustering; these include STIRR [19], ROCK [23], CACTUS [17], COOLCAT [6], k-modes [29], and others more [11, 53]. Other works have focused more narrowly on binary or transactional data [37, 46], on a framework to compress high dimensional categorical datasets [33], and on using hypergraph partitioning to cluster itemsets [25].

The COOLCAT algorithm introduced by Barbara et al. [6] 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.

Cristofor et al.[11] present another approach based on cluster entropy measures for categorical attributes. Starting from a seed clustering, it uses genetic algorithms [20, 47] 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 [29], an extension to the well-known k-means

algorithm [34] 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 [19]. 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.[53] 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 affects the detected clusters. The clusters identified by STIRR were shown to be incomplete in cases of overlapping cluster projections [17].

Guha et al. present ROCK[23], 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 when 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 [17], 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 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 which 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 their support.

The available CACTUS implementation from its authors does not include this extension step, and it is unclear whether or not the reported performance [17] accounts for extension and validation. A study of the extension and validation

phase shows a significant performance impact (see Figure 5.1). 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. A detailed account of the MDL principle is presented in the context of the CLIQUE algorithm in the following section.

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 [35], PROCLUS [1], ORCLUS [2], and others more [32, 38, 50]. 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 discussed.

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 D_k that have a $k-1$ dimensional projection not included in D_{k-1} are subsequently pruned. To further reduce the computational cost of the candidate construction, the authors propose to prune candidate subspaces with low *coverage* where the cut-off point is determined by using a Minimum Description Length (MDL) technique. Clearly, completeness of the method is lost in this step. Suppose that candidate subspaces $S_1, \ldots, S_n$

were detected. The *coverage* x_{S_i} of S_i is defined as the number of points that fall into dense units u_j in S_i

$$x_{S_i} = \sum_{u_j \in S_i} \sigma(u_j)$$

Highly relevant subspaces (i.e. subspaces with high coverage) should be retained, while low coverage subspaces should be pruned. To this end, the subspaces are sorted by descending coverage and for each index $i \in \{1, \ldots, n\}$ in the resulting sequence, a potential split into a set of retained subspaces $I = S_1, \ldots, S_i$ and a set of pruned subspaces $P = S_{i+1}, \ldots, S_n$ is considered. Let $\mu_I(i)$ and $\mu_P(i)$ be the average coverage of the two sets when split at index i. $\mu(i)$ can be binary encoded in $log_2(\mu(i))$ bits space. If the coverage information about $S_1, \ldots, S_i, S_{i+1}, \ldots, S_n$ is to be encoded, storing the differences to the respective means is sufficient. Overall, a *code length* $CL(i)$ of

$$log_2(\mu_I(i)) + \sum_{1 \leq j \leq i} log_2(|x_{S_i} - \mu_I(i)|) + log_2(\mu_P(i)) + \sum_{i+1 \leq j \leq n} log_2(|x_{S_j} - \mu_P(i)|)$$

bits is needed to capture the coverage information when splitting at index i. The authors propose using the index i with the minimal $CL(i)$ as optimal cut-off point when selecting appropriate subspaces. Finally, the output clusters are 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 [35] improves over 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 [36]. 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 the idea that along a relevant dimension of a medoid m_i points should be close to that medoid.

ORCLUS [2] improves over 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 the variance along direction e_i. In this sense, the ORCLUS projections can be understood as an inverted Principal Component Analysis [9, 24], where the goal is to find projection directions with minimum variance.

The RIS algorithm presented in [32] 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 [15]. 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 suitable for categorical data sets.

3. Improved Categorical Clustering

This chapter introduces CLICKS, a novel clustering method that maps the categorical clustering problem to the problem of enumerating maximal k-partite cliques in a k-partite graph [44, 45, 49]. Like the attribute summaries used by CACTUS, CLICKS 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.

3.1 Graph Fomalization for Categorical Datasets

Consider the sample dataset $\mathcal{D}$ given in Table 3.1 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 3.1.

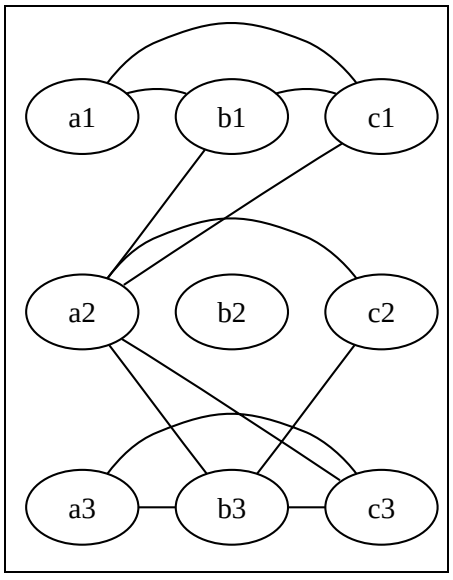

Figure 3.1: A Sample Graph Encoding

The co-occurrence of a_1 and b_1 in the first record is represented by the edge

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 3.1: Sample Categorical Dataset

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 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 τ 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, τ can be chosen to correspond to the threshold that makes the two values strongly connected with respect to the user parameter α. Thus the dataset $\mathcal{D}$ can be represented as a k-partite graph $\Gamma(\mathcal{D})$ as follows.

Definition 3.1.1 (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 $\alpha \in \mathbb{R}^+$. The undirected graph $\Gamma(\mathcal{D}) = (V, E)$ where*

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

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

Definition 3.1.2 (k-partite (Maximal) Clique) *Let $\mathcal{D}$ be a dataset with $V = \bigcup_{i=1}^{n} D_i$, and let $\Gamma(\mathcal{D}) = (V, E)$ be its k-partite graph. $C \subseteq V$ 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 $(v_i, v_j) \in E$ in $\Gamma(\mathcal{D})$. If there is no $C' \supset C$ such that C' is a k-partite clique in $\Gamma(\mathcal{D})$, C is called a* k-partite 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 3.1.3 (Cluster/Clique mapping) *Given a categorical dataset $\mathcal{D}$ and a k-subspace $C = C_1 \times \ldots \times C_k$ with $C_j \subseteq D_{i_j}$.*

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 $\sigma^*_\alpha(C) = 1$, then C is a cluster over attributes $\{A_{i_1}, \ldots, A_{i_k}\}$.*

PROOF: $\underline{\Rightarrow}$. For $i, j \in \{1, \ldots, k\}$, $i \neq j$, C_i and C_j are strongly connected in D, i.e. $\sigma^*_\alpha(c_i, c_j) = 1$ 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 every C_i is maximal. Hence C is maximal.

$\underline{\Leftarrow}$. For $i, j \in \{1, \ldots, k\}$, $i \neq j$ 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 $\sigma^*_\alpha(c_i, c_j) = 1$. 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, $\sigma^*_\alpha(C) = 1$ implies $\sigma(C) \geq \alpha \times E[\sigma(C)]$, i.e. C is dense. Thus C is a cluster over $\{A_{i_1}, \ldots, A_{i_k}\}$. □

Example 3.1.4 *For the k-partite graph $\Gamma(\mathcal{D})$ depicted in Figure 3.1 a total of five k-partite maximal cliques can be found. The cliques on $(\{a_1, a_2\} \times \{b_1\} \times \{c_1\})$, $(\{a_2, a_3\} \times \{b_3\} \times \{c_3\})$, and $(\{a_2\} \times \{b_3\} \times \{c_2, c_3\})$ are full dimensional.*

Additionally, the two subspace cliques $(\{a_2\} \times \{b_1, b_3\})$ *and* $(\{a_2\} \times \{c_1, c_2, c_3\})$ *exist.*

The first clique $C_1 = (\{a_1, a_2\} \times \{b_1\} \times \{c_1\})$*, for example, is dense w.r.t an* α *value of* 2*, i.e.* $\sigma_2^*(C_1) = 1$ *. To show this, consider the expected support under the attribute independence assumption.*

Assume that the dataset from which $\Gamma(\mathcal{D})$ *was generated contains ten records (the depicted records from Table 3.1 plus four additional records indicated by the ellipsis). By Figure 3.1, the individual domains are* $D_1 = \{a_1, a_2, a_3\}, D_2 = \{b_1, b_2, b_3\}, D_3 = \{c_1, c_2, c_3\}$*. Hence, every value should occur in* $1/3$ *of the transactions under independence assumption. For* C_1 *this yields an expected support of*

$$E[\sigma(C_1)] = 10 \times 2/3 \times 1/3 \times 1/3 = 0.74$$

C_1 *receives an actual support* $\sigma(C_1)$ *of at least* 2 *from transactions* 1 *and* 4 *plus potential other supporters in transactions* 7 - 10*. Since*

$$2 = \sigma(C_1) \geq \alpha \times E[\sigma(C_1)] = 2 \times 0.74 = 1.48$$

the clique C_1 *is dense. By theorem 3.1.3,* C_1 *is also a cluster over* $\{A_1, A_2, A_3\}$*.*

3.2 The CLICKS Algorithm

By theorem 3.1.3, 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 $\sigma_\alpha^*(C) = 1$ for $C \in \mathcal{C}$. 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.

The basic CLICKS approach consists of the three principal stages, shown in Figure 3.2, as follows:

- *Pre-Processing*: In this step, the k-partite graph $\Gamma(\mathcal{D})$ is created from the input database $\mathcal{D}$, and the attributes are ranked for efficiency reasons.

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

  PreProcess$(\mathcal{D}, \alpha, \Gamma(\mathcal{D}), \mathcal{R})$
  DetectMaxCliques$(\Gamma(\mathcal{D}), \mathcal{C}, \mathcal{R}, C)$
  PostProcess($\mathcal{D}$, $\mathcal{C}$, α, $minsup$)
  return $\mathcal{C}$

Figure 3.2: The CLICKS algorithm

- *Clique Detection*: Given $\Gamma(\mathcal{D})$, all 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.

3.2.1 Pre-processing

CLICKS generalizes the clique enumeration technique presented in [31] 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 neighbors plus the remaining values of the attribute in question ($D_i - \{v_i\}$). 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 the empty set.

Definition 3.2.1 (Neighbors) *Let $\mathcal{D}$ be a categorical dataset over attributes $A_1, \ldots, A_n$ and $V = \bigcup_{i=1}^{n} D_i$. Let $\alpha \in \mathbb{R}^+$. The* neighbors *of an attribute value $v_i \in D_i$ are given by the function $N : V \to 2^V$, defined as*

$$N(v_i) = \Big\{ v_j \in V : \sigma_\alpha^*\big(\{v_i\} \times \{v_j\}\big) = 1 \Big\}$$

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

Definition 3.2.2 (Connectivity) *Let $\mathcal{D}$ be a categorical dataset and $v_i \in D_i$. The* connectivity $\eta(v_i)$ *is defined as:*

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

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 3.2.3 (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 3.2), 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.

3.2.2 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 constructed 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 starts with one vertex and 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(Graph Γ(D)
                 CliqueList C,
                 AttributeValueRanking R,
                 Clique C)
1.   if(Φ(C) ∧ R = ∅) then
2.     C = C ∪ {C }
3.     return

4.   R^D = R^P = ∅
5.   foreach v in R − R^D − R^P do
6.       C' = C ∪ {v}
7.       R' = ∅
8.       R^D = R^D ∪ {v}

9.       foreach v' in R − R^D do
10.          if (v' ∈ η(v)) then R' = R' ∪ {v'}

11.      if (v is first value in R) then R^P = R'
12.      if(Φ(R' ∪ C')) then
13.        DetectMaxCliques(Γ(D), C, R', C')
```

Figure 3.3: The CLICKS Clique Detection

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 (line 1). $\Phi(C)$ 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 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 (line 5) 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 (line 9) 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. If v is the first value in $\mathcal{R}$, the associated attribute value ranking $\mathcal{R}'$ can be added to $\mathcal{R}^P$ without sacrificing completeness (see [31]).

Finally, the algorithm recurses on the newly created clique C' with its matching attribute value ranking $\mathcal{R}'$ (line 13). 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 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 the first v_1, it need not be considered as an extension to C. This latter case is guarded against by the *processed* attribute values $\mathcal{R}^P$.

Example 3.2.4 *Consider the k-partite graph encoding* $\Gamma(\mathcal{D})$ *in Figure 3.1, 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.*

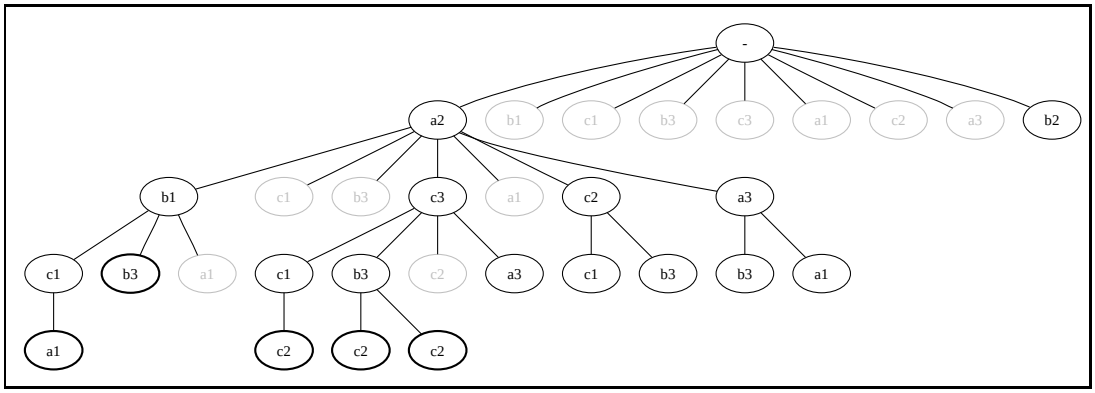

Figure 3.4: DetectMaxCliques Example

Figure 3.4 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* $\{c_1, b_3, a_1\}$*. This example shows both full and subspace cliques. For example* $\{a_1, a_2, b_1, c_1\}$ *is a full space clique.*

3.2.3 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*)) invokes 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 $\sigma^*_\alpha(C) = 1$, 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 CLICKS output. While all maximal cliques are reported by the clique detection phase, some might be pruned out based on the support criterion. 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 CLICKS. Section 4.2 presents a vertical method for exploring induced subcliques.

The method above works well, but for some datasets it may output too many overlapping cliques. 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 the 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.

Example 3.2.5 *Consider again the dataset in Table 3.1. 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 CLICKS 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 3.1 (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 subspace 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 3.2.6 (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 denote 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 3.2.7 (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. [3]

Definition 3.2.8 (Coverage Weight) *For a frequent clique set $\mathcal{F}_D$, the* coverage weight $\omega : \mathcal{F}_D \rightarrow \mathbb{N}$ *is defined as*

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

where $X = \{1, \cdots, m\} \in \mathcal{F}_D$ is a set of clique ids (corresponding to cliques $\{C^1, \ldots, C^m\}$) that frequently occur together and $\sigma_{D_M}(X)$ denotes X's support within the problem dataset D_M from definition 3.2.7.

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 [21] 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

[3] Alternatively, sampling could be used to increase the accuracy of the guiding heuristic at a reasonable cost.

PostProcess($\mathcal{D}$, $\mathcal{C}$, α, *minsup*)
1. Scan $\mathcal{D}$ and check support of each $C \in \mathcal{C}$
2. $\mathcal{F}_\mathcal{D}$ = Maximal Frequent Clique Set of $\mathcal{D}$
3. Sort $\mathcal{F}_\mathcal{D}$ by coverage weights ω

4. $\mathcal{F}^P = \emptyset$
5. **for** $i = 1$ **to** $|\mathcal{F}_\mathcal{D}|$ **do**
6. **if** $\mathcal{F}_\mathcal{D}[i] \neq \emptyset$ **then**
7. $\mathcal{F}^P = \mathcal{F}^P \cup \{\mathcal{F}_\mathcal{D}[i]\}$
8. $\mathcal{F}_\mathcal{D}[j] = \mathcal{F}_\mathcal{D}[j] - \mathcal{F}_\mathcal{D}[i]$ for all $i < j \leq |\mathcal{F}_\mathcal{D}|$

9. $\mathcal{C}' = \mathcal{C}$ // Save the original cliques
10. $\mathcal{C} = \emptyset$
11. **for** $i = 1$ **to** $|\mathcal{F}^P|$ **do**
12. **if** $\omega(\mathcal{F}^P[i]) \geq E[\omega(\mathcal{F}^P[i])$ **then**
13. // Add union of cliques with indices in $\mathcal{F}^P[i]$ to $\mathcal{C}$
14. $\mathcal{C} = \mathcal{C} \cup \mathcal{C}'[\mathcal{F}^P[i]]$

Figure 3.5: CLICKS **Post Processing for Merging Clusters**

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.

3.3 Merging Characterization

Clusters computed by the clique generation phase of CLICKS are (after support checking) exactly those required by the cluster definition as stated in theorem 3.1.3. 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 1.2.6. 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 CLICKS 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 3.2.7 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 CLICKS merging phase prefers those that yield larger clusters. To this end, CLICKS applies a greedy strategy combined with a heuristic for the resulting clique size. The *coverage weight* introduced in definition 3.2.8 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 CLICKS 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, CLICKS heuristically chooses the solution that will yield the largest cliques.

3.4 A Bi-Clustering Alternative?

An anonymous reviewer suggested a bi-partite model for categorical datasets. At first glance, this model seems an appealing alternative to the k-partite model of CLICKS since powerful graph algorithms have already been devised for bi-

partite models. However, this fomalization would *not* be fit with respect to the cluster notion from definition 1.2.6.

The formalization of the bi-clustering problem dates back to the 1970s [28]. More recently, bi-clustering has received increasing attention in the context of computational biology, where it is used to cluster gene expression data [8, 18, 43].

Consider once more the dataset $\mathcal{D}$ from Table 3.1. Let $U = \{1, \ldots, 6\}$ denote the set of all displayed record identifiers, and let $V = \bigcup_{i=1}^{3} D_i$ be the set of attribute values from all domains combined.[4] A natural bi-partite mapping is the graph $G = ((U \cup V), E)$ where $(u, v) \in E$ iff the value v appears in record u. Figure 3.6 shows the resulting graph for the sample dataset. For clarity, only edges from transaction 1 are included.

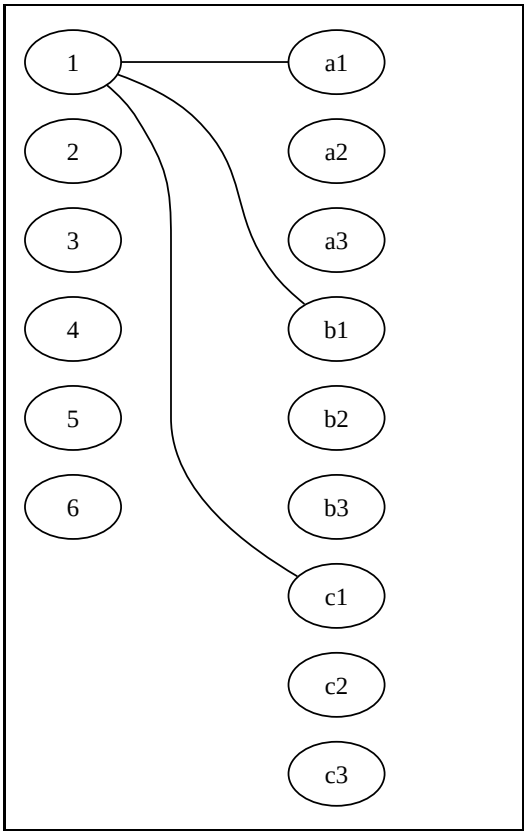

Figure 3.6: A bi-partite Encoding

The reviewer claims that a cluster is equivalent to a maximal bi-partite

[4]Note, that domains are required to be disjoint by definition 1.1.1.

clique (X, Y) with $X \subseteq U$ and $Y \subseteq V$. However, this conjecture is wrong. Recall from Example 3.1.4 that $C_1 = (\{a_1, a_2\} \times \{b_1\} \times \{c_1\})$ is a k-partite maximal clique in the k-partite graph in Figure 3.1. Yet, the corresponding bi-partite clique $(X, \{a_1, a_2, b_1, c_1\})$, where X is the set of supporting transaction IDs, can not be mined from a bi-partite representation. The reason is, that no record u can be connected to both, a_1 and a_2 since they never co-occur. If the algorithm was extended to not consider such edges, the bi-partite model would essentially become the same as the k-partite model used by CLICKS . The k-partite model seems to be the more natural representation after all.

4. Vertical Techniques in Categorical Clustering

Vertical mining approaches have been successfully employed in frequent pattern mining [13, 42, 51]. The basic idea in vertical encodings is to store the set of transaction IDs that support a given pattern. When deriving new patterns from old ones, e.g. when extending k-itemsets to $(k+1)$-itemsets in the a-priori lattice, the supporting transactions of the new pattern can be computed through simple set operations based on the old support sets. Support information for each candidate pattern is always kept current and is available for pruning measures without frequent passes over the dataset. Completeness of the resulting candidates is guaranteed by the *anti-monotonicity* or *downward closed* property of itemset support: If a k-itemset is not frequent then none of its supersets can be frequent.

This section extends the CLICKS algorithm to use a vertical representation of the dataset. Section 4.1 presents a *full vertical approach*, and section 4.2 shows how the completeness of non-vertical CLICKS can be restored by using a *selective vertical approach* in the post-processing phase.

4.1 Full Vertical Mining

The CLICKS algorithm as outlined in section 3 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. An alternative graph representation $G = (V, E)$ 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 $\sigma^*_\alpha(v_i, v_j) = 1$, or

$v_i \in D_o, v_j \in D_{j>0}$ and v_j occurs in transaction with id v_i.

Figure 4.1 shows an example where the attribute values 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.

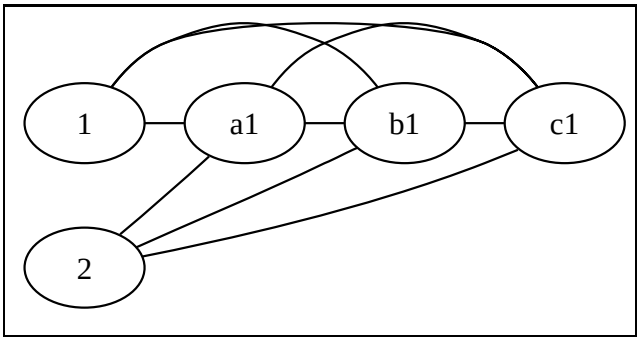

Figure 4.1: Vertical Representation Example

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 if implemented in a naive fashion: If the transaction ids are processed as regular vertices, the regular adjacency matrix used by CLICKS grows quadratically. The resulting representation is most likely too large to fit in main memory. Special techniques for sparse matrices [12] can obviate this problem.

A more straightforward alternative is to annotate the vertices of the k-partite graph $\Gamma(\mathcal{D})$ with the transaction ids that the vertices are supported by, i.e. the records that the corresponding attribute values occur in. Consider once again the example dataset $\mathcal{D}$ in Table 3.1 and its associated graph $\Gamma(\mathcal{D})$ in Figure 3.1. To indicate that the attribute value 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. This set representation mitigates the need for elaborate memory management techniques, since only transactions are annotated where an attribute value actually occurs in. The numerous zeroes ("not in transaction") in the naive encoding are never

considered.

Definition 4.1.1 (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.*

$$\lambda_{\Gamma(\mathcal{D})}(v_i) = \{t \in \mathcal{D} : t.A_i = v_i\}^5$$

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, *each* of its attribute values 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 (such as the one depicted in Figure 3.4) to offset the additional computation. Hence, the supporting transactions need to be computed every time a clique is extended, and an appropriate pruning criterion has to be defined.

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.

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

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 [51] can be applied to reduce memory consumption.

With transaction information available at every point of the search tree, a pruning criterion can be defined based on the support of the clique constructed up to that point. Care must be taken not to cut potentially successful branches. 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. The reason for this is, that the support is not monotonous along the search path. In the example graph in Figure 3.1 and the associated dataset in Table 3.1, 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 entails 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 4.1.1 (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*

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

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 4.1.1 can be used to extend the original CLICKS algorithm as shown in Figure 4.2. 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 criterion 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 4.1.1 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 support. 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 obviates the need for support checking in the post-processing algorithm shown in Figure 3.5. However, performance studies indicate that the full vertical approach does not yield better performance than the regular approach.

4.2 Restoring Completeness by Selective Vertical Mining

The full vertical mining approach presented in the previous section is affected by two major cost factors

1. The low pruning bound severely reduces the number of possible pruning cases.

2. In large-size graphs, a myriad of backtrack possibilities need to be explored even if one branch of the search tree can be pruned.

The problem of low pruning bounds is a fundamental problem as discussed in section 4.1. The second problem, however, would be partly mitigated if only the search space (i.e. the size of the input graph) was small enough. Clearly, no such constraint can be imposed on the input data to Clicks. Yet, a second possible application for efficient vertical mining exists.

In section 3.2.3 it was noted that the support counting approach in the non-vertical Clicks implementation compromises the completeness of the output

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

4.   R^D = R^P = ∅
5.   foreach v in R − R^D − R^P do
6.       C' = C ∪ {v}
7.       R' = ∅
8.       R^D = R^D ∪ {v}

9.       P = ∅
10.      foreach v' in R − R^D do
11.        if (v' ∈ η(v)) then
12.          R' = R' ∪ {v'}
13.          P = P ∪ {v'}
14.      if (v is first value in R) then R^P = R'

15.      if(Φ(R' ∪ C'))
16.        if(|λ_Gamma(D)(C')| > 0)
17.          if(DetectMaxCliquesVertical(Γ(D), C, R', C') = false)
18.            R^D = R^D − {v}
19.            R^P = R^P − P
20.        else
21.          // Prune, but other search paths must be explored
22.          R^D = R^D − {v}
23.          R^P = R^P − P

24.  if(No DetectMaxCliquesVertical call returned true)
25.    return false
26.  else
27.    return true
```

Figure 4.2: The Vertical CLICKS Clique Detection

clusters. While all maximal cliques are reported by the clique detection phase, some are pruned out based on the support criterion. Subcliques of a pruned clique, however, might have had proper support. This problem exists due to the separation between the clique generation and the support verification phases. An

exploration of the subcliques induced by every rejected clique can be made to restore the completeness. Alas, enumerating all possible subcliques and counting their support is computationally expensive. A more affordable approach is to understand the pruned clique as the k-partite graph of a reduced clique detection problem. The reduced problem is likely to be significantly smaller in terms of occuring attribute values, making it an ideal candidate for vertical mining. For better distinction from the full vertical approach presented in section 4.1, this technique is called *selective vertical mining*.

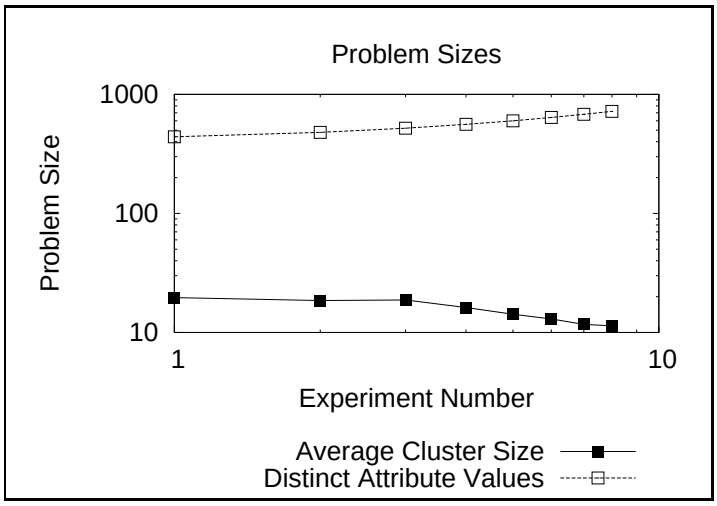

Figure 4.3: Cluster sizes in Synthetic Datasets

Figure 4.3 shows the results of an experiment that illustrates the effect of different problem sizes on a logarithmic scale. Eight subspace clustering runs were conducted on synthetic datasets with 120.000 records and 4 attributes. Three fixed clusters (10.000 tuples each) were injected on attribute values $[0, 4]$, $[3, 7]$, and $[5, 9]$. In all runs α was set to 1.6 and the *minsup* value was chosen as 0.01.

The domain size was now varied from 110 in the first experiment to 190 in the last experiment, increasing the total problem size from $440 = 4 \times 110$ to $760 = 4 \times 190$. Note, that this is the problem size for a *full vertical* mining approach. On the other hand, the problem size for the *selective* problem, i.e. the average size of the generated clusters, was one to two orders of magnitude smaller in every case. Clearly, this advantage is partly compensated by the fact

that multiple problems need to be solved in selective vertical mining, namely one for every clique that does not fulfill the *minsup* requirement. However, the impact of the *number of problems* is linear, where the impact of the total *problem size* is exponential. The result is a substantial net gain in performance over the full vertical approach.

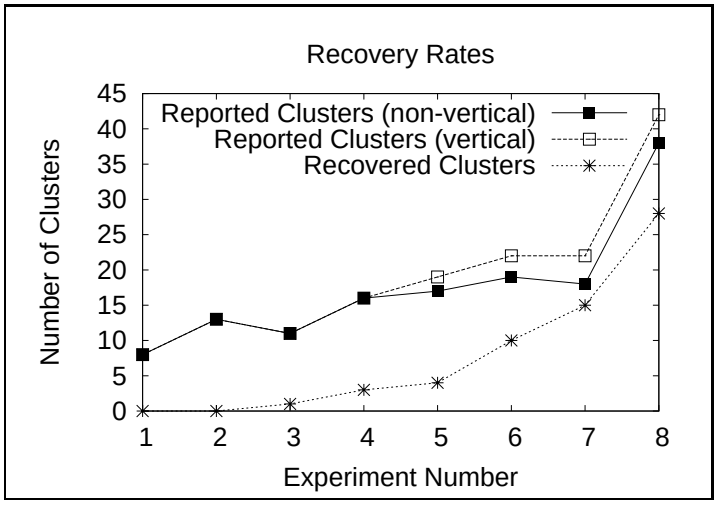

Figure 4.4: Recovery Rates with Selective Vertical Mining

For the experiments above, the number of reported clusters in the selective vertical case, the non-vertical case, and the number of recovered clusters was recorded (see Figure 4.4). Note, that the number of recovered clusters refers to the number of clusters *before* merging, hence their number is greater than the difference between the reported clusters *after* post-processing for the two cases. The figure indicates that the fraction of clusters that are lost in post-processing increases significantly as the dataset gets sparser. This is natural, considering that more (subspace) clusters are generated based on connectedness information that need to be dismissed for their final support in the dataset. In contrast, only about 5 percent of all clusters are missing from the final result when not using selective vertical mining. However, two effects need to be considered

- The experiments were conducted at a relatively low merging threshold. The gap between the actual clusters and the detected clusters *after* post processing will increase as the merging threshold grows.

- Even though the recovered cliques contribute only a small number of new clusters to the final output, they do contribute significantly to the shape of the output clusters. In the experiments, a large faction of recovered clusters added to the fringe areas of the final output clusters (see Section 3.3).

The algorithm used as part of the post-processing routine for the selective vertical approach is basically identical to the full vertical algorithm in Figure 4.2. Only a specific attribute value ranking has to be constucted for every pruned clique that is to be explored. Moreover, vertical information (i.e. the vertical labeling function λ_Γ) has to be computed when reading in the dataset in pre-processing. This information is assumed to be present in the following discussion.

PostProcessSelective($\mathcal{D}$, $\mathcal{C}$, α, *minsup*)
1. Scan $\mathcal{D}$ and check support of each $C \in \mathcal{C}$
2. $\mathcal{C}^P = \{C \in \mathcal{C} | \sigma_{\mathcal{D}}(C) < minsup\}$
3. $\mathcal{C}^R = \{C \in \mathcal{C} | \sigma_{\mathcal{D}}(C) \geq minsup\}$

4. $\mathcal{C}' = \mathcal{C}^R$
5. **foreach** C **in** $\mathcal{C}^P$ **do**
6. $\quad \mathcal{C}^I = \emptyset, \mathcal{R} = \mathcal{R}(C)$
7. $\quad$ **DetectMaxCliquesVertical**$(\Gamma(\mathcal{D}), \mathcal{C}^I, \mathcal{R}, \emptyset)$
8. $\quad \mathcal{C}' = \mathcal{C}' \cup \mathcal{C}^I$

9. $\mathcal{C} = \mathcal{C}'$
10. $\mathcal{F}_{\mathcal{D}}$ = Maximal Frequent Clique Set of $\mathcal{D}$
11. Sort $\mathcal{F}_{\mathcal{D}}$ by coverage weights ω

12. ... (like before)

Figure 4.5: CLICKS **Post Processing for Selective Vertical Mining**

Consider the revised post-processing algorithm depicted in Figure 4.5. Based on the *minsup* parameter, the originally detected cliques $\mathcal{C}$ are partitioned into the *pruned cliques* $\mathcal{C}^P$ and the *retained cliques* $\mathcal{C}^R$ (lines 2 - 3). While the original algorithm simply deleted the cliques in $\mathcal{C}^P$, further exploration is now made to restore potential frequent subcliques.

The retained cliques are unconditionally copied to the complete clique set $\mathcal{C}'$. For every clique in $\mathcal{C}^P$ the induced subcliques $\mathcal{C}^I$ are determined through

DetectMaxCliquesVertical. As parameters, the regular dataset graph $\Gamma(D)$, the output clique set $\mathcal{C}^I$, the attribute value ranking $\mathcal{R}$, and the empty start clique $\emptyset$ are passed (line 7). The attribute value ranking is constructed so that only attribute values that occur within C are used (line 6). This way, the search space is restricted to subcliques of C. Note, that **DetectMaxCliquesVertical** ensures that cliques in $\mathcal{C}^I$ are frequent through vertical information. Hence, no further support checking needs to be performed on the result.

Finally, the complete clique set C' of all retained cliques plus the frequent subcliques is copied back to $\mathcal{C}$. Subsequent merging operations can be performed on $\mathcal{C}$ like in Figure 3.5. Performance benchmarks for the modified post-processing procedure can be found in Section 5.5.

5. Experimental Study

This chapter presents an extensive performance study of CLICKS 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 dedicated local 100MBit network. The code for CACTUS was obtained directly from its authors. The experiments will show that CLICKS 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 [19]; 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 subspace. Ganti et al. choose $\kappa = 2$ and $\alpha = 3$ in their own CACTUS performance tests. In all present performance tests CACTUS was run with these settings. CLICKS was also configured to use $\alpha = 3$.

5.1 CACTUS Extension

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 [17]

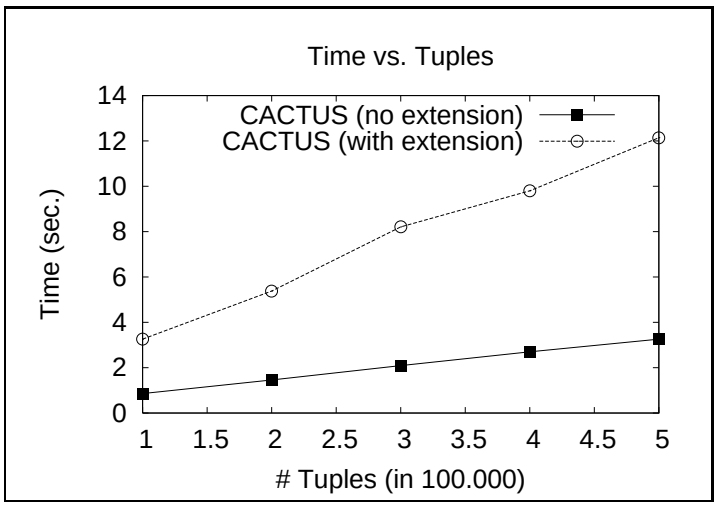

Figure 5.1: Performance Impact of Extensions

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 5.1 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 [17], 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.

5.2 STIRR and ROCK

The STIRR [19] algorithm, as implemented by Ganti et al. was also benchmarked. STIRR outputs the non-principal basins, i.e., weighted vertices, that identify the cluster projection on each attribute. As in the case of CACTUS, no clusters are actually output. Ganti et al. report in [17] 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 CLICKS. While CLICKS uses up to 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 5.2 shows that CLICKS still outperforms ROCK by orders of magnitude. The practical application of ROCK is thus limited to datasets of well below 10000 records where CLICKS scales into the million record range.

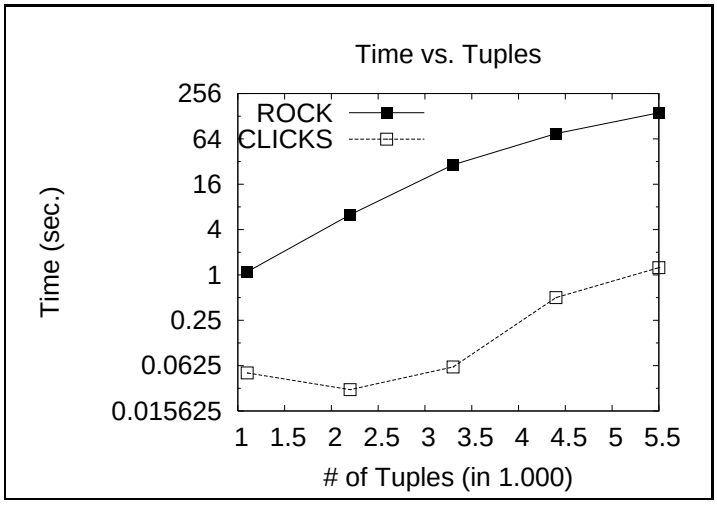

Figure 5.2: CLICKS **vs. ROCK**

Given these limitations, the performance comparison only benchmarks the CLICKS runtimes against CACTUS. Some sample outputs of STIRR are shown

in section 5.6 where the clustering quality of CLICKS is compared with CACTUS and STIRR.

5.3 Performance Comparison

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

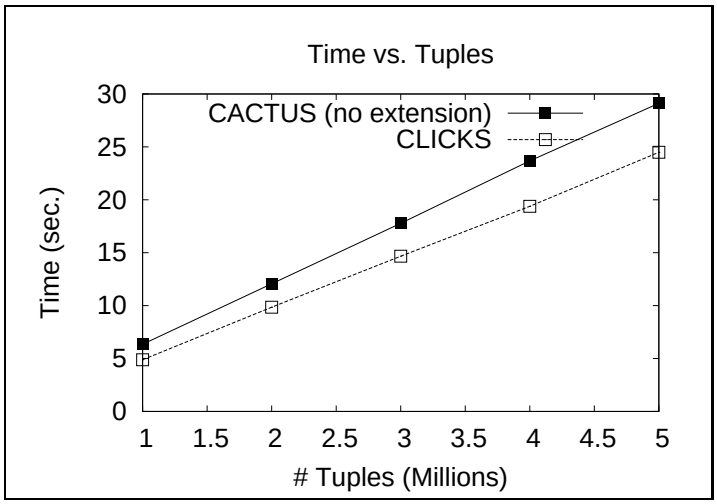

Figure 5.3: CLICKS vs. CACTUS (Tuples)

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 5.3. CLICKS 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 from 50 to 500. Figure 5.4 shows that both methods perform equally well for 300 and less 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

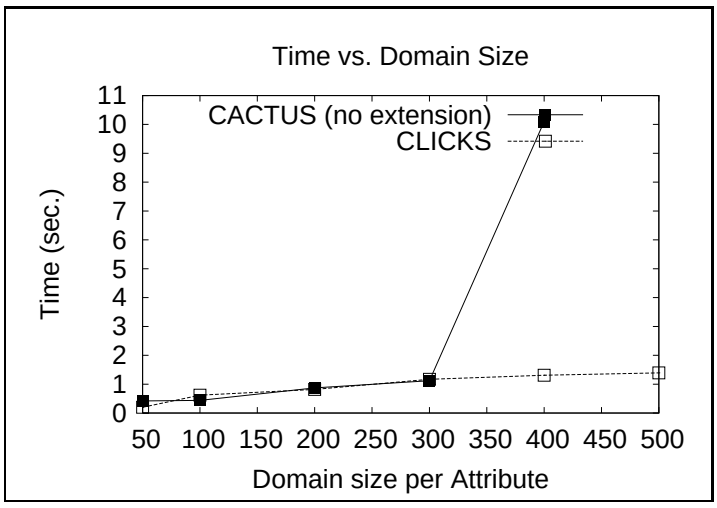

Figure 5.4: CLICKS **vs. CACTUS (Domain Size)**

compress the memory consumption of their approach. The underlying idea is to collapse multiple attribute values into one value. To fit the domain $\mathcal{D}_i$ of attribute A_i into a main memory area of capacity M, the reduction ratio c is chosen to be $c = \left\lceil \frac{|\mathcal{D}_i|}{M} \right\rceil$. The reduced domain $\mathcal{D}'_i$ is constructed as

$$\mathcal{D}'_i = \{f(1), \ldots, f(|\mathcal{D}_i|)\} \text{ with } f(i) = \left\lfloor \frac{i}{c} \right\rfloor + 1$$

Hence, attribute values that are strongly connected in the original domain are also strongly connected in the transformed domain. The opposite is not necessarily true. When the algorithm needs inter-attribute connectedness information for (a'_i, a_j) where $a'_i \in D'_i, a_j \in \mathcal{D}_j$ and $j \neq i$, the collapsed value is expanded to the original values $\{a'_i \cdot c + 1, \ldots, a'_i \cdot c + c\}$. The dataset is then scanned to count the support of the original pairs $(a'_i \cdot c + 1, a_j), \ldots, (a'_i \cdot c + c, a_j)$ which restores the original resolution. Ganti et al. propose that the reduction process be applied recursively if the expanded support counts fail to fit into main memory as intended, hence the name "multi layered approach". CLICKS scales well beyond this point without a need for additional memory compression.

Performance vs. Dimensionality: CLICKS is especially scalable with regards to higher dimensional data. On a dataset with 1 million tuples and 100 attribute

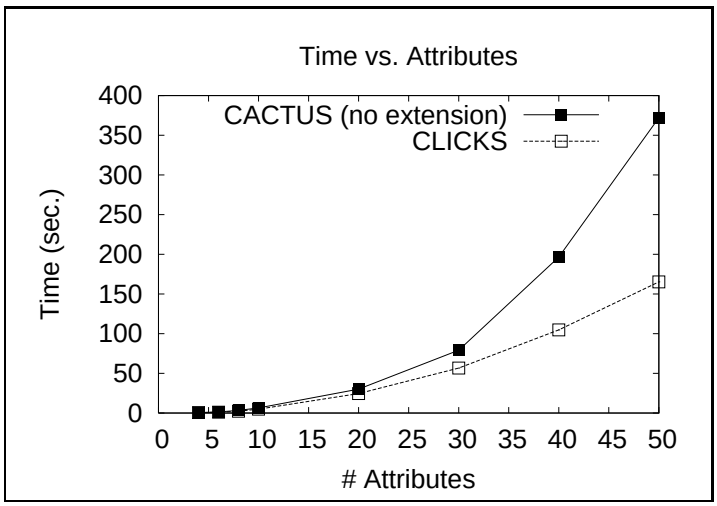

Figure 5.5: CLICKS **vs. CACTUS (Attributes)**

values per dimension, CLICKS outperforms CACTUS by a factor 2 - 3 when varying the number of attributes from 5 to 50 as shown in Figure 5.5.

5.4 Full Vertical Mining Performance

Section 4.1 presents a full vertical mining extension to the standard CLICKS algorithm that mitigates the need for a second pass over the dataset. Instead, the vertical CLICKS method requires additional memory and additional CPU time for set computations.

The full vertical approach was implemented in CLICKS and benchmarked against the CLICKS 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.

Performance vs. Dataset size: Figure 5.6 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.

Performance vs. Domain size: As in the previous case, the CLICKS baseline version yields better results than the vertical implementation (Figure 5.7).

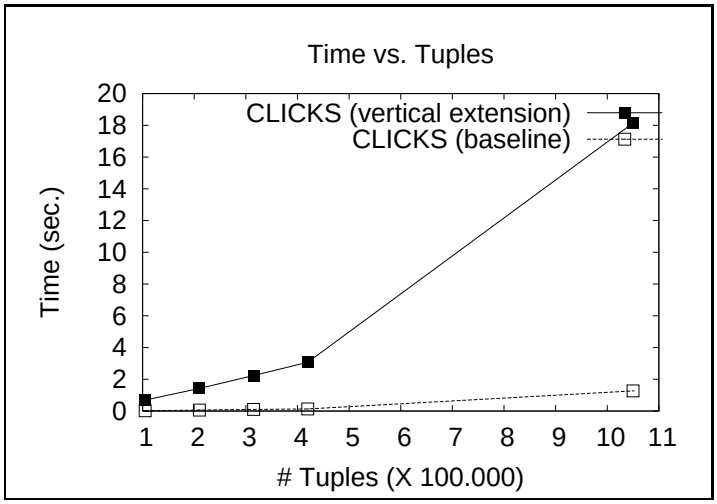

Figure 5.6: CLICKS **Full Vertical Performance (Tuples)**

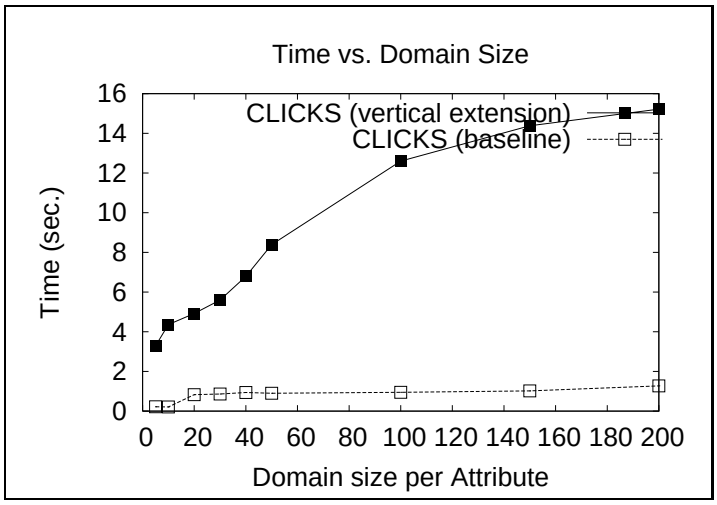

Figure 5.7: CLICKS **Full Vertical Performance (Domain Size)**

Performance vs. Dimensionality: Finally, the execution time was measured against the number of attributes in the dataset (Figure 5.8). 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 algorithm gained advantages with the number of tuples, as expected. However, the cost of hard disk

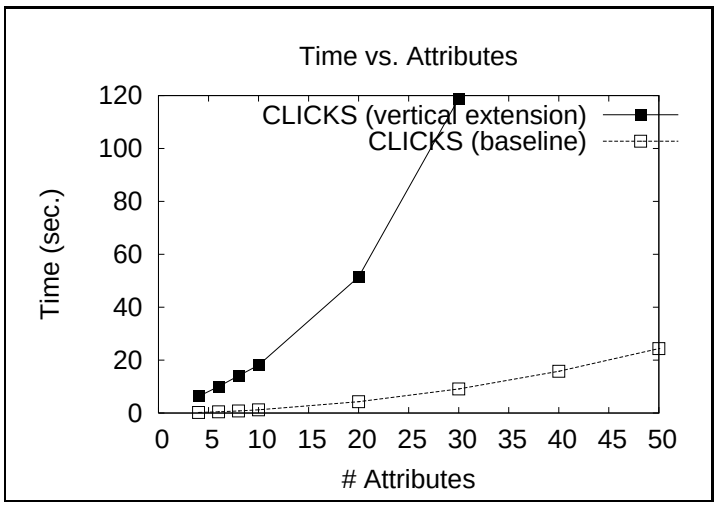

Figure 5.8: CLICKS **Full Vertical Performance (Attributes)**

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 CLICKS baseline version.

5.5 Selective Vertical Mining Performance

As an alternative to full vertical mining, section 4.2 proposes a selective vertical approach that restores the completeness of CLICKS through an additional mining step in the post-processing phase. Clearly, this method cannot be faster than the baseline CLICKS version as it adds additional post-processing. The question in this scenario is whether the generated overhead is acceptable given completeness. Again, the runtime was measured against the number of tuples, the dimensionality, and the domain size of the dataset. CLICKS was configured to run with subspace clustering enabled to reveal the full effect of the restoration procedure: many of the induced subliques are likely to be found in subspaces.

Performance vs. Dataset size: The CLICKS baseline version is about five times faster than the selective vertical CLICKS version (see Figure 5.9). More detailed analysis reveals that this overhead is largely induced by building the vertical information that is now needed. Even though the added cost is significant

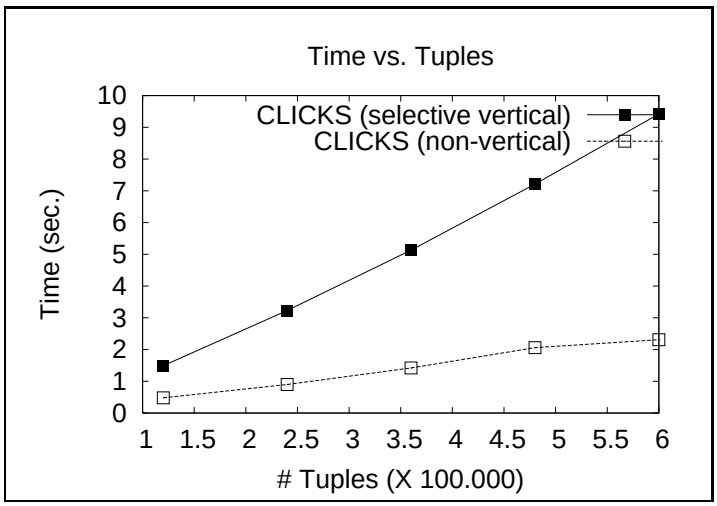

Figure 5.9: CLICKS **Selective Vertical Performance (Tuples)**

the added cost factor remains constant, where in full vertical mining it increases with the number of tuples.

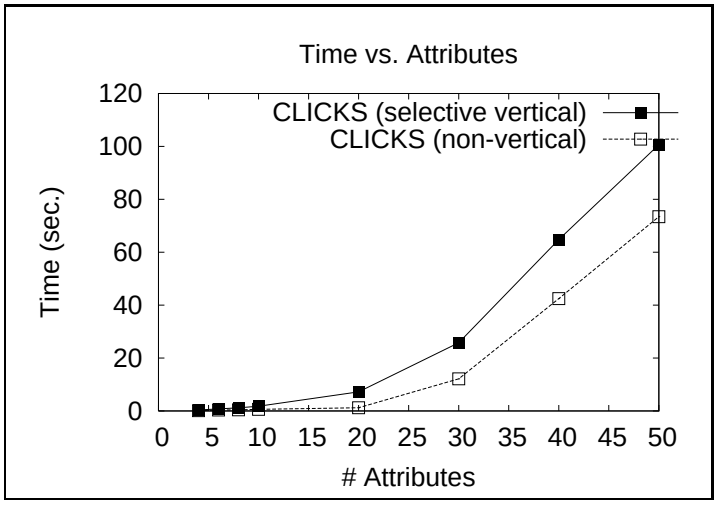

Figure 5.10: CLICKS **Selective Vertical Performance (Attributes)**

Performance vs. Dimensionality: Figure 5.10 shows how the selective vertical CLICKS version handles an increasing number of attributes within the dataset. Again, the additional post-processing induces computational cost. However, the percentage of overhead generated by the extension remains constant at an estimated 20 percent. This behavior is much unlike the full vertical implementation, where the gap increased dramatically (see Figure 5.8) with the number of dimen-

sions.

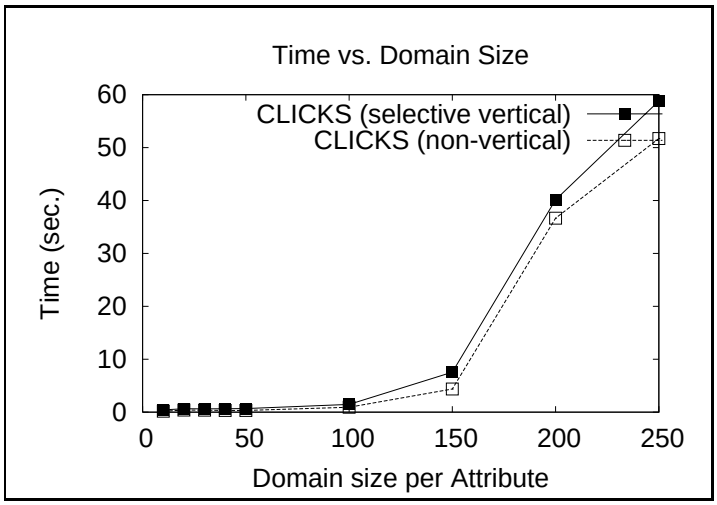

Figure 5.11: CLICKS **Selective Vertical Performance (Domain Size)**

Performance vs. Domain size: Figure 5.11 reveals that varying the domain size creates only a minor performance impact.

Overall, the selective vertical mining technique proves to be an efficient supplement to CLICKS . For scenarios where completeness of the clustering results is desired, it can find the erroneously pruned clusters at a cost that is much lower than doing full vertical mining from the start. Most importantly, the added overhead is linear with respect to the cost that is incurred in the non-vertical scenario, making it a computationally viable option even for large datasets.

5.6 Clustering Quality

To evaluate the quality of the clusters generated by CLICKS, 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.

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

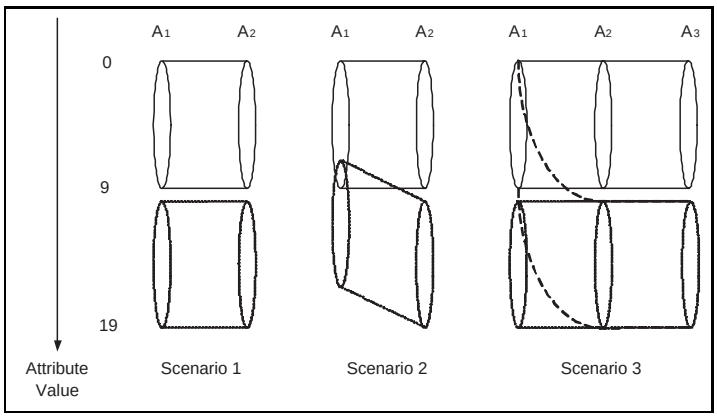

Figure 5.12: Cluster Quality Comparison

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 [19] this corresponds to a single cluster on $[0, 19] \times [0, 19]$, confirming the lack of separation found in [17].

Scenario 2 A dataset with a slight overlap between two clusters on one attribute. CLICKS 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 1.2.6.

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 CLICKS 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. 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 with respect to the cluster definition 1.2.6, 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 CLICKS 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.

5.7 Real Data Clustering

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

Mushroom Dataset The Mushroom dataset is part of the UCI Machine Learning Repository [6] 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 [17] contains 7766 records of database publications [48], and 30919 records of papers on theoretical Computer Science [41]. Each records has four 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 articles talk about, as SGML markup. To make the data accessible to CLICKS the articles were pre-processed using a morphology function [14] to collapse different verb forms and tenses (e.g. *am*, *was*, *is* are all mapped to *be*). This transformation is also known as *stemming*. A frequency dictionary was built and pruned to the 193 most frequent words. Finally, the first 10000 articles were

[6] 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 5.1: 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 5.2: Confusion Matrix Mushrooms (Subspace)

encoded according to the scheme presented in section 1.3. The result was a boolean dataset with 193 attributes and 10000 records on which CLICKS 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, CLICKS 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 5.1 and 5.2 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 CLICKS generated.

Full dimensional clustering (Table 5.1) initially yielded 256 candidate clusters which were then reduced to 213 clusters using a *minsup* value of 0.5% for

	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 5.3: 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 5.4: Confusion Matrix Votes (Subspace)

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 5.2 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 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 5.3 shows the results for a full dimensional clustering with CLICKS.

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 5.4). 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 5.5 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

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 5.5: 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 5.6: Clusters for Reuters 21578

dictionary CLICKS could be used to cluster according to such specific dimensions.

CLICKS 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 5.6. Clearly, the clustering found by the algorithm gives a meaningful picture 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.

5.8 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 [17]. 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.

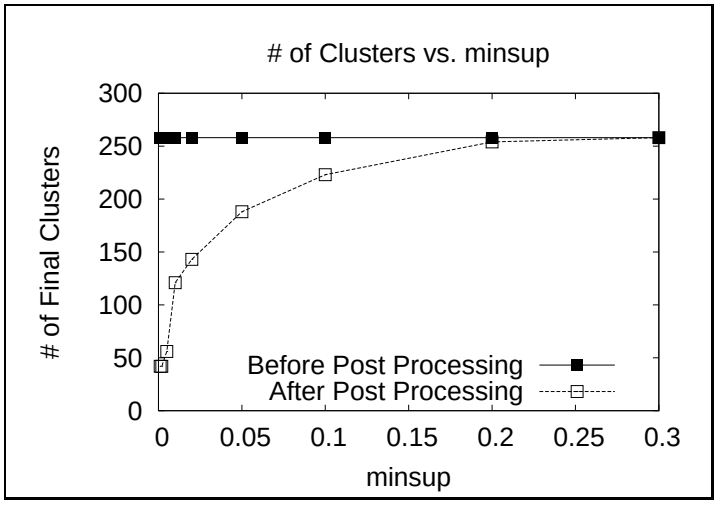

Figure 5.13: Post-processor Reduction

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 5.13 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.

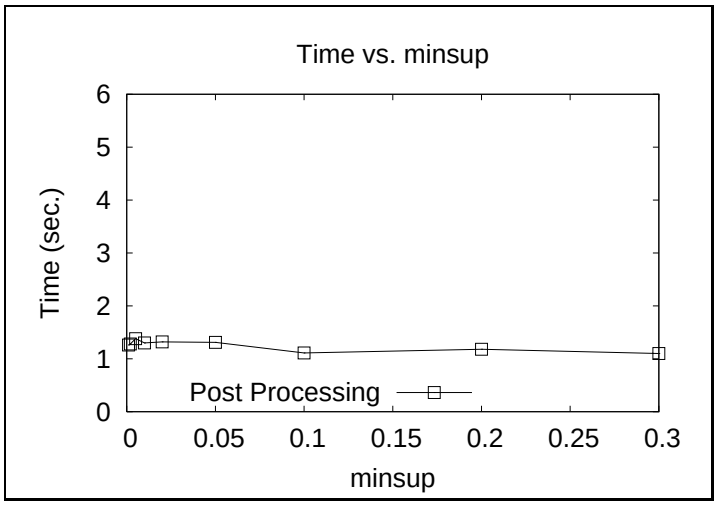

Figure 5.14: Post-processor Performance

Moreover, the merging performance is not affected by the chosen *minsup* value, as can be seen in Figure 5.14. 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.

6. Discussion and Conclusions

This work introduced CLICKS, 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 CLICKS. 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 on a comprehensive set of real and synthetic datasets. The experiments indicate substantial performance gains of factor two to three over previous approaches, especially on high dimensional datasets.

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

Despite the relatively poor performance of full vertical clique mining, useful applications exist for the vertical encoding. A *selective vertical* approach was introduced that restores the completeness of the output like full vertical mining while adding only an acceptable level of overhead. Selective vertical mining was shown to be a viable option for scenarios where completeness is desired.

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

Clique-based categorical clustering . The CLICKS 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 proven 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 . CLICKS naturally integrates subspace clustering capabilities into the clustering process. This is a 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 *full 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 reduced version called *selective vertical* mining was presented, that recovers cliques not previously detected due to the separation between clique generation and support counting in the non-vertical approach. Selective vertical mining is a cost-effective alternative to the incomplete CLICKS baseline implementation.

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. If the current connectivity ap-

proach was to be kept, sampling could be a fruitful alternative to the existing connectivity heuristic. The added accuracy in guidance could potentially offset the cost of the sampling process.

Secondly, 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.

APPENDIX A
CLICKS command line usage

Syntax

```
./kcdriver <input file> <alpha> <minsup> <benchmark file>
           [FULL|SUB] [VERTICAL] [QUIET] [CONFUSION] [MAP <mapfile>]
```

Example

```
./kcdriver mush.click 1.2 0.8 bench.txt SUB CONFUSION MAP map.txt
```

Description

`<input file>` An input dataset in binary CLICKS format. CLICKS datasets can either be generated using a modified version of the synthetic dataset generator from [19], or with the `mconvert` tool that allows conversion of space-separated-value files.

`<alpha>` The α multiplier introduced in section 1.2.

`<minsup>` Minimum fraction of transactions that must support a valid cluster.

`<benchmark file>` Filename where benchmarking information for the CLICKS run is appended. The benchmarking file contains one row for every CLICKS run where the space-separated columns are: (1) total time, (2) number of tuples, (3) number of attributes, (4) maximum number of values for any attribute, (5) final number of cliques, (6) cliques after clique detection, (7) pre-processing time, (8) clique detection time, (9) post-processing time

`SUB` When the `SUB` option is specified, CLICKS searches for full dimensional *and* subspace clusters. Without the `SUB` option, CLICKS only finds full dimensional clusters.

`VERTICAL` Activates full vertical mining. Note, that this option is ignored when CLICKS is compiled in lean mode (i.e. if the `KC_LEAN` directive is activated in `KCGlobal.h`). In this case, CLICKS indicates the lean compilation in its output.

`SELECTIVE` Activates selective vertical mining. If CLICKS is configured to run in full vertical mode this option is ignored.

`QUIET` Prevents CLICKS from outputting the final clusters after merging.

`CONFUSION` Confusion information is written to `click_confusion.txt`. The file contains one line per transaction in the data file. Each line consists of a space separated list of cluster indices that the transaction falls into. If a transaction is not part of a cluster, the line contains -1. Confusion files can be used to generate confusion matrices with the `mconvert` utility.

`MAP <mapfile>` Causes CLICKS to read in the mapping information from the given `mconvert` map file. When CSV files are converted to CLICKS data with `mconvert` the original attribute values are converted to numbers. CLICKS outputs the clusters in terms of the original values in the CSV file if the map file is available.

APPENDIX B

`mconvert` command line usage

The `mconvert` utility can be used for a variety of data conversion tasks.

Syntax

```
./mconvert OUTCLICK <click file>
./mconvert OUTCACTUS <cactus file>
./mconvert CSVTOCLICK <CSV file> <confusion file> <mapping file>
                      <total columns> <label column> {ignored}*
./mconvert CONFUSION <confusion file> <confusion file>
                     <confusion mapping>
./mconvert CLICKTOROCK <click file> <item limit>

./mconvert CSVTOCACTUS <csv file> <total columns> {ignored}*
```

Examples

```
./mconvert OUTCLICK mush.click
./mconvert CSVTOCLICK mush.csv mush.confusion mush.map 23 0 > out
```

Description

`<click file>` An input dataset in binary Clicks format.

`<cactus file>` An input dataset in the CACTUS format.

`<CSV file>` An ASCII dataset containing one record per row where the individual attribute values are space-separated.

`<confusion file>` Filename to write the confusion information from the conversion process. The confusion file contains one row for each record in the

CSV file, where each row contains the index of the cluster that the transaction belongs to. The cluster index of a transaction is determined by the value of the i-th column, where i can be specified via the `<label column>` parameter.

`<mapping file>` ASCII file containing the mapping between the original attribute values in a CSV file and the numbers used by CLICKS. The mapping file can be used with CLICKS to output the clusters in terms of the original attribute values.

`<total columns>` Total number of columns contained in the CSV file.

`<label column>` Column of a CSV file used to derive the cluster that a record belongs to (see `<confusion file>`). The column index is zero-based, i.e. the first column is 0, the second column 1, and so on.

{`ignored`}* Space separated list of columns in a CSV file that should not be considered during conversion. The list can be empty, the column indices are zero-based.

`<confusion mapping>` ASCII file containing a mapping between cluster indices in the two confusion files. Each row contains two space-separated integers giving a (from-index to-index) mapping that is applied to the cluster indices in the first confusion file.

`<item limit>` ROCK data files contain a distance matrix. The `<item limit>` parameter gives the size (in each dimension) of this matrix.

LITERATURE CITED

[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] M. Ankerst, M. Breunig, H.-P. Kriegl, and J. Sander. Optics: Ordering points to identify the clustering structure. In *Proceedings of the ACM SIGMOD Intl. Conference on Management of Data*, pages 49–60, 1999.

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

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

[8] Y. Cheng and G.M. Chruch. Biclustering of expression data. In *Proceedings of the ISMB*, pages 93–103, 2000.

[9] W.C. Cooley and P.R. Lohnes. *Multivariate Data Analysis*. Wiley, 1971.

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

[11] 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.

[12] I.S. Duff, A.M. Erisman, and J.K. Reid. *Direct Methods for Sparse Matrices*. Oxford University Press, 1986.

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

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

[15] 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.

[16] Martin Ester, Hans-Peter Kriegel, Jorg Sander, and Xiaowei Xu. A distribution-based clustering algorithm for mining in large spatial databases. In *14th Intl. Conf. on Data Engineering*, April 1998.

[17] 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.

[18] G. Getz, E. Levine, and E. Domany. Coupled two-way clustering analysis of gene microarray data. In *Proceedings of the National Acadademy of Science*, pages 12079–12084, 2000.

[19] 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.

[20] D.E. Goldberg. *Genetic Algorithms in Search, Optimization, and Machine Learning*. Addison Wesley, 1989.

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

[22] S. Guha, R. Rastogi, and K. Shim. Cure: An efficient algorithm for clustering large databases. In *ACM SIGMOD Conf. Management of Data*, June 1998.

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

[24] S. Hammarling. The singular value decomposition in multivariate statistics. In *ACM Signum Newsletter*, volume 20, pages 2–25, 1985.

[25] 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.

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

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

[28] J.A. Hartigan. *Clustering Algorithms*. Wiley, 1975.

[29] 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.

[30] A.K. Jain and R.C. Dubes. *Algorithms For Clustering Data*. Prentice-Hall, Englewood Cliffs, 1988.

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

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

[33] 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.

[34] J. McQueen. Some methods for classification and analysis of multivariate observations. In *Proceedings of the Fifth Berkeley Symposium on Mathematical Statistics and Probability*, pages 281–297, 1967.

[35] H. Nagesh, S. Goil, and A. Choudhary. Mafia: Efficient and scalable subspace clustering for very large data sets. In *Technical Report 9906-010, Center for Parallel and Distributed Computing, Northwestern University*, 1999.

[36] 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.

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

[38] 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.

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

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

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

[42] 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.

[43] A. Tanay, R. Sharan, and R. Shamir. Discovering statistically signifnicant biclusters in gene expression data. In *Proceedings of ISMB*, pages 136–144, 2002.

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

[45] L. Volkmann. *Fundamente der Graphentheorie*. Springer, 1996.

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

[47] P.D. Wasserman. *Advanced Methods in Neural Computing*. Thomson Learning, 1993.

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

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

[50] 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.

[51] 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.

[52] T. Zhang, R. Ramakrishnan, and M. Livny. BIRCH: An efficient data clustering method for very large databases. In *ACM SIGMOD Conference on Management of Data*, 1996.

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