CACTUS–Clustering Categorical Data Using Summaries

Venkatesh Ganti* Johannes Gehrke Raghu Ramakrishnan
Department of Computer Sciences, University of Wisconsin-Madison
{vganti,johannes,raghu}@cs.wisc.edu

Abstract

Clustering is an important data mining problem. Most of the earlier work on clustering focussed on numeric attributes which have a natural ordering on their attribute values. Recently, clustering data with categorical attributes, whose attribute values do not have a natural ordering, has received some attention. However, previous algorithms do not give a formal description of the clusters they discover and assume that the user post-processes the output of the algorithm to identify the final clusters.

In this paper, we introduce a novel formalization of a cluster for categorical attributes by generalizing a definition of a cluster for numerical attributes. We then describe a very fast summarization-based algorithm called CACTUS that discovers exactly such clusters in the data. CACTUS has two important characteristics. First, the algorithm requires only two scans of the dataset, and hence is very fast and scalable. Our experiments on a variety of datasets show that CACTUS outperforms previous work by a factor of 3 to 10. Second, CACTUS can find clusters in subsets of all attributes and can thus perform a subspace clustering of the data. This feature is important if clusters do not span all attributes, a likely scenario if the number of attributes is very large. In a thorough experimental evaluation we study the performance of CACTUS on real and synthetic datasets.

1 Introduction

Clustering is an important data mining problem. The goal of clustering, in general, is to discover dense and sparse regions in a dataset. Most previous work in clustering focussed on numerical data whose inherent geometric properties can be exploited to naturally define distance functions between points. However, many datasets also consist of categorical attributes[1] on which distance or similarity functions are not naturally defined. Recently, the problem of clustering categorical data started receiving interest [GKR98, GRS99].

As an example, consider the `MUSHROOM` dataset in the popular UCI Machine Learning repository [CBM98]. Each tuple in the dataset describes a sample of gilled mushrooms using 22 categorical attributes. For instance, the `cap color` attribute can take values from the domain {`brown, buff, cinnamon,`

* Contact author.

[1] Attributes whose domain is totally ordered are called *numeric*, whereas attributes whose domain is not ordered are called *categorical* (sometimes, also called *nominal).*

gray, green, pink, purple, red, white, yellow}. It is hard to reason that one color is "like" or "unlike" another color in a way similar to real numbers.

An important characteristic of categorical domains is that they typically have a small number of attribute values. For instance, the largest domain for a categorical attribute of any dataset in the UCI Machine Learning repository consists of 100 attribute values (for an attribute of the Pendigits dataset). Categorical attributes with large domain sizes typically do not contain information that may be used to group tuples into classes. For instance, the CustomerId attribute in the TPC-D database benchmark [Cou95] may consist of millions of values; given that a record (or a set of records) takes a certain CustomerId value (or a set of values), we cannot infer any information that is useful for classifying the records. Therefore, it is different from the age or geographical location attributes which can be used to group customers based on their age or location or both. Typically, relations contain 10 to 50 attributes; hence, even though the size of each categorical domain is small, the cross product of all their domains and hence the relation itself can be very large.

In this paper, we introduce a new algorithm called CACTUS[2] for clustering categorical data. CACTUS is a fast summarization-based algorithm that exploits the small domain sizes of categorical attributes. The central idea in CACTUS is that summary information constructed from the dataset is sufficient for discovering well-defined clusters. The properties that the summary information typically fits into main memory, and that it can be constructed efficiently using a single scan of the dataset result in significant performance improvements—a factor of 3 to 10 times over one of the previous algorithms. Our main contributions in this paper are:

1. We formalize the concept of a cluster over categorical attributes (Section 3).
2. We introduce a fast summarization-based algorithm CACTUS for clustering categorical data (Section 4).
3. We then extend (CACTUS) to discover clusters in subspaces, especially useful when the data consists of a large number of attributes (Section 5). Due to space constraints, we describe the extension in Appendix B.
4. In an extensive experimental study, we evaluate CACTUS and compare it with earlier work on synthetic and real datasets (Section 6).

2 Related Work

In this section, we discuss two previous algorithms STIRR and ROCK for clustering categorical data.

Gibson et al. introduce STIRR [GKR98], an iterative algorithm based on non-linear dynamical systems. They represent each attribute value as a weighted vertex in a graph. (Edges between vertices—derived from tuples in the dataset—are not explicitly maintained.) Multiple copies $b_1, \ldots, b_m$, called *basins*, of this set of weighted vertices are maintained; the weights on any given vertex may differ across basins. b_1 is called

[2] **CA**tegorical **C**lus**T**ering **U**sing **S**ummaries

the *principal* basin; $b_2, \ldots, b_m$ are called *non-principal* basins. Starting with a set of weights on all vertices (in all basins), the system is "iterated" until a fixed point is reached. In each iteration, the dataset is scanned once; for each tuple $t = \langle t_1, \ldots, t_n \rangle$, the weight in basin b_i on the vertex t_j is incremented by a function (called the *combining function*) of the set of weights in b_i on vertices other than t_j in t. At the end of each iteration, the sets of weights of attribute values on the same attribute across basins are normalized using the Gram-Schmidt orthonormalization [DER86]. For certain types of combining functions (e.g., sum), the authors showed that the fixed point will be reached.

Gibson et al. argue that when the fixed point is reached, the weights in one or more of the basins $b_2, \ldots, b_m$ isolate two groups of attribute values on each attribute: the first with large positive weights and the second with small negative weights, and that these groups correspond intuitively to projections of clusters on the attribute. However, the automatic identification of such sets of closely related attribute values from their weights is an instance of the 1-dimensional clustering problem, and hence requires a non-trivial post-processing step; such a post-processing step was not addressed in their work. Moreover, the post-processing step will also determine what "clusters" are output. In Section 3.2, we also show that certain classes of clusters cannot be discovered by `STIRR`.

Guha et al. [GRS99] introduce `ROCK`, an adaptation of an agglomerative hierarchical clustering algorithm, which heuristically optimizes a criterion function defined in terms of the number of "links" between tuples. Informally, the number of links between two tuples is the number of common "neighbors" they have in the dataset. Given a *similarity function*, two tuples in the dataset are said to be *neighbors* if the similarity between them is greater than a certain threshold. Starting with each tuple in its own cluster, they repeatedly merge the two "closest" clusters till the required number (say, K) of clusters remain. The closeness between two clusters is defined to be the sum of the number of links between all pairs of tuples—one in each cluster. Since the complexity of this hierarchical algorithm is *quadratic* in the number of tuples in the dataset, they cluster a sample randomly drawn from the dataset, and then partition the entire dataset based on the clusters from the sample. Beyond that the set of all "clusters" together may optimize a criterion function, the set of tuples in each individual cluster is not characterized.

Like `STIRR`, our algorithm `CACTUS` does not require the number of clusters as an input parameter. We define each cluster in terms of the attribute values rather than as a subset of the tuples in the dataset (like in `ROCK`). Thus, in a broad sense, our approach is closer to that of `STIRR` than to that of `ROCK`.

3 Definitions

In this section, we formally define the concept of a cluster over categorical attributes, and other concepts used in the remainder of the paper. We then compare the class of clusters allowed by our definition with those discovered by `STIRR`.

3.1 Cluster Definition

Intuitively, a cluster on a set of numeric attributes identifies a "dense region" in the attribute space. That is, the region consists of a significantly larger number of tuples than expected. We generalize this intuitive notion for the class of hyper-rectangular clusters to the categorical domain.[3]

As an illustrative example, consider a dataset with three categorical attributes A, B, and C with domains $\mathcal{D}_A$, $\mathcal{D}_B$, and $\mathcal{D}_C$, respectively. Let A', B', and C' be three numeric attributes. The class of rectangular regions in the space spanned by A', B', C' can be expressed as the cross product $[a_1, a_2] \times [b_1, b_2] \times [c_1, c_2]$ of intervals $[a_1, a_2]$, $[b_1, b_2]$, and $[c_1, c_2]$ on A', B', and C', respectively. Since domains of categorical attributes are not ordered, the concept of an interval does not exist. However, a straightforward generalization of the concept of an interval to the categorical domain is a *set* of attribute values. Consequently, the generalization of a rectangular region on A', B', C' to the categorical attributes A, B, C is the cross product $S_1 \times S_2 \times S_3$ of three sets S_1, S_2, S_3 of attribute values such that $S_1 \subseteq \mathcal{D}_A$, $S_2 \subseteq \mathcal{D}_B$, and $S_3 \subseteq \mathcal{D}_C$. We call such regions *interval regions*.

Intuitively, a cluster consists of a significantly larger number of tuples than the number expected if all attributes were independent. In addition to the significantly larger number, a cluster also extends to as large a region as possible. We now formalize this notion for categorical domains by first defining the notion of a tuple *belonging* to a region, and then the *support* of a region, which is the number of tuples in the dataset that belong to the region.

Definition 3.1 Let $A_1, \ldots, A_n$ be a set of categorical attributes with domains $\mathcal{D}_1, \ldots, \mathcal{D}_n$, respectively. Let the dataset D be a set of tuples where each tuple t: $t \in \mathcal{D}_1 \times \cdots \times \mathcal{D}_n$. We call $\mathcal{S} = S_1 \times \cdots \times S_n$ an *interval region* if for all $i \in \{1, \ldots, n\}$, $S_i \subseteq \mathcal{D}_i$. Let $a_i \in \mathcal{D}_i$ and $a_j \in \mathcal{D}_j$, $i \neq j$. The *support* $\sigma_D(a_i, a_j)$ of the attribute value pair (a_i, a_j) with respect to D is defined as follows:

$$\sigma_D(a_i, a_j) \stackrel{\text{def}}{=} \frac{|\{t \in D : t.A_i = a_i \text{ and } t.A_j = a_j\}|}{|D|}$$

A tuple $t = \langle t.A_1, \ldots, t.A_n \rangle \in D$ is said to *belong* to the region $\mathcal{S}$ if for all $i \in \{1, \ldots, n\}$, $t.A_i \in S_i$. The *support* $\sigma_D(\mathcal{S})$ of $\mathcal{S}$ is the number of tuples in D that belong to $\mathcal{S}$.

If all attributes $A_1, \ldots, A_n$ are independent (henceforth referred to as the *attribute-independence* assumption) then the *expected support* $E[\sigma_D(\mathcal{S})]$ of a region $\mathcal{S} = S_1 \times \cdots \times S_n$ is $|D| \cdot \frac{|S_1| \times \cdots \times |S_n|}{|\mathcal{D}_1| \times \cdots \times |\mathcal{D}_n|}$; the expected support of (a_i, a_j) $E[\sigma_D(a_i, a_j)]$ is $|D| \cdot \frac{1}{|\mathcal{D}_i| \times |\mathcal{D}_j|}$. Since the dataset D is understood from the context, we write $\sigma(\mathcal{S})$ instead of $\sigma_D(\mathcal{S})$, and $\sigma(a_i, a_j)$ instead of $\sigma_D(a_i, a_j)$.

Intuitively, $\sigma(a_i, a_j)$ captures the co-occurence, and hence the similarity, of attribute values a_i and a_j. Values a_i and a_j are said to be *strongly connected* if their co-occurrence ($\sigma(a_i, a_j)$) is significantly higher (by

[3]Classes of clusters that correspond to arbitrarily shaped regions in the numeric domain cannot be generalized as cleanly to the categorical domain because the categorical attributes do not have a natural ordering imposed on their domains. Therefore, we only consider the class of hyper-rectangular regions.

some factor α) than the value expected under the attribute-independence.[4] We now define σ^* to formalize this intuition, and then give a formal definition of a cluster.

Definition 3.2 Let $a_i \in \mathcal{D}_i$, $a_j \in \mathcal{D}_j$, and $\alpha > 1$. The attribute values a_i and a_j are *strongly connected* with respect to D if $\sigma_D(a_i, a_j) > \alpha \cdot \frac{|D|}{|\mathcal{D}_i| \cdot |\mathcal{D}_j|}$. The function ${\sigma^*}_D(a_i, a_j)$ is defined as follows:

$${\sigma^*}_D(a_i, a_j) \stackrel{\text{def}}{=} \begin{cases} \sigma_D(a_i, a_j), & \text{if } a_i \text{ and } a_j \text{ are strongly connected} \\ 0, & \text{otherwise} \end{cases}$$

Let $S_i \subseteq \mathcal{D}_i$ and $S_j \subseteq \mathcal{D}_j$, $i \neq j$, be two sets of attribute values. An element $a_i \in S_i$ is strongly connected with S_j if, for all $x \in S_j$, a_i and x are strongly connected. S_i and S_j are said to be strongly connected if each $a_i \in S_i$ is strongly connected with S_j and each $a_j \in S_j$ is strongly connected with S_i.

Definition 3.3 For $i = 1, \ldots, n$, let $C_i \subseteq \mathcal{D}_i$, $|C_i| > 1$, and $\alpha > 1$. Then $C = \langle C_1, \ldots, C_n \rangle$ is a *cluster* over $\{A_1, \ldots, A_n\}$ if the following three conditions are satisfied. (1) For all $i, j \in \{1, \ldots, n\}, i \neq j$, C_i and C_j are strongly connected. (2) For all $i, j \in \{1, \ldots, n\}$, $i \neq j$, there exists no $C'_i \supset C_i$ such that for all $j \neq i$, C'_i and C_j are strongly connected. (3) The support $\sigma_D(C)$ of C is at least α times the expected support of C under the attribute-independence assumption.

We call C_i the *cluster-projection* of C on A_i. C is called a *sub-cluster* if it satisfies conditions (1) and (3). A cluster C over a subset of all attributes $\mathcal{S} \subset \{A_1, \ldots, A_n\}$ is called a *subspace cluster* on $\mathcal{S}$; if $|\mathcal{S}| = k$ then C is called a *k-cluster*.

We now extend our notion of similarity to attribute value pairs on the same attribute. Let $a_1, a_2 \in \mathcal{D}_i$ and $x \in \mathcal{D}_j$. If (a_1, x) and (a_2, x) are strongly connected then (a_1, a_2) are "similar" to each other with respect to A_j. The level of similarity is the number of such distinct attribute values $x \in \mathcal{D}_j$. We now formalize this intuition.

Definition 3.4 Let $a_1, a_2 \in \mathcal{D}_i$. The *similarity* $\gamma^j(a_1, a_2)$ between a_1 and a_2 with respect to A_j $(j \neq i)$ is defined as follows.

$$\gamma^j(a_1, a_2) \stackrel{\text{def}}{=} |\{x \in \mathcal{D}_j : \sigma^*(a_1, x) > 0 \text{ and } \sigma^*(a_2, x) > 0\}|$$

Below, we define the summary information which we need later to describe the `CACTUS` algorithm. The summary information is of two types: (1) *inter-attribute* summaries and (2) *intra-attribute* summaries. The inter-attribute summaries consist of all strongly connected attribute value pairs where each pair has attribute values from different attributes; the intra-attribute summaries consist of similarities between attribute values of the same attribute.

[4] Because a deviation of 2 or 3 times the expected value is quite significant [BD76], typical values of α are between 2 and 3.

Definition 3.5 Let $A_1, \ldots, A_n$ be a set of categorical attributes with domains $\mathcal{D}_1, \ldots, \mathcal{D}_n$ respectively, and let D be a dataset. The *inter-attribute summary* Σ_{IJ} is defined as: $\Sigma_{IJ} \stackrel{\text{def}}{=} \{\Sigma_{ij} : i, j \in \{1, \ldots, n\}, i \neq j\}$ where

$$\Sigma_{ij} \stackrel{\text{def}}{=} \{(a_i, a_j, \sigma^*_D(a_i, a_j)) : a_i \in \mathcal{D}_i, a_j \in \mathcal{D}_j, \text{ and } \sigma^*_D(a_i, a_j) > 0\}$$

The *intra-attribute summary* Σ_{II} is defined as: $\Sigma_{II} \stackrel{\text{def}}{=} \{\Sigma^j_{ii} : i, j \in \{1, \ldots, n\} \text{ and } i \neq j\}$ where

$$\Sigma^j_{ii} \stackrel{\text{def}}{=} \{(a_{i1}, a_{i2}, \gamma^j(a_{i1}, a_{i2})) : a_{i1}, a_{i2} \in \mathcal{D}_i, \text{ and } \gamma^j(a_{i1}, a_{i2})) > 0\}$$

Let $\mathcal{G} = \langle V, \mathcal{E} \rangle$ be a graph such that $V = \cup^n_{i=1} \mathcal{D}_i$ and $(a_i, a_j) \in \mathcal{E}$ iff $\sigma^*_D(a_i, a_j) > 0$. The graph $\mathcal{G}$ is called the *similarity graph* on $A_1, \ldots, A_n$ with respect to the dataset D. Let $\mathcal{G}^j_i = \langle \mathcal{D}_i, \mathcal{E}^j_{ii}$ be a graph such that $(a_{i1}, a_{i2}) \in \mathcal{E}^j_{ii}$ iff $\gamma^j(a_{i1}, a_{i2})) > 0$. The graph $\mathcal{G}^j_i$ is called the *similarity graph* on A_i with respect to A_j.

3.2 Discussion

We now compare the class of clusters allowed by our definition with the clusters discovered by STIRR. For the comparison, we generate test data using the data generator developed by Gibson et al. for evaluating STIRR [GKR98]. We consider three datasets shown in Figures 1, 2, and 3. Each dataset consists of 100000 tuples. DS1 and DS2 have two attributes, DS3 has three attributes where each attribute consists of 100 attribute values. We control the location and the size of clusters in each dataset by distributing an additional number of tuples (5% of the total number in the dataset) in regions designated to be clusters thus increasing their supports above the expected value under the attribute-independence assumption. The remaining tuples are distributed over all attribute values on each attribute according to the attribute-independence assumption. In Figures 1, 2, and 3, the cluster-projection of each cluster is shown within an ellipse. The boundaries of the cluster-projections (ellipses) of a cluster are connected by lines of the same type (e.g., solid, dashed etc.).

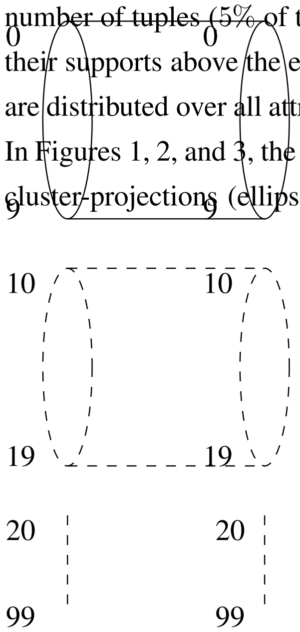

Figure 1: DS1

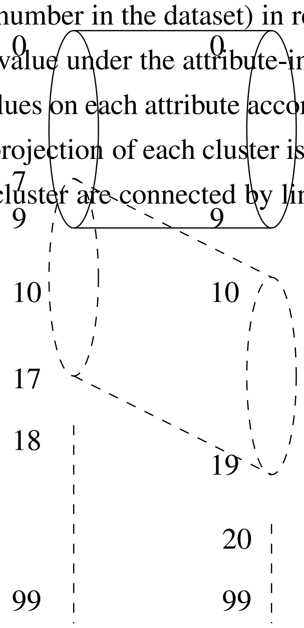

Figure 2: DS2

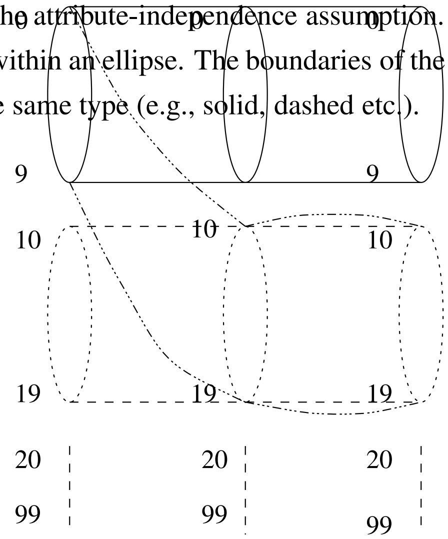

Figure 3: DS3

We ran STIRR on the datasets shown in Figures 1, 2, and 3, and manually selected the basin that assigns positive and negative weights respectively to attribute values in different cluster-projections. To identify the cluster projections, we observe the weights allocated by STIRR and split them into two groups of positive and negative weights. The cluster-projections identified by STIRR are shown in Figures 4, 5, and 6.

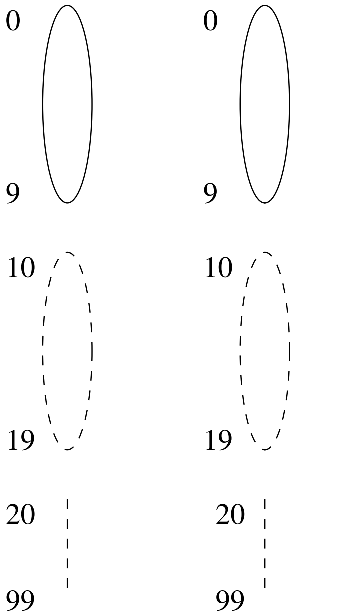

Figure 4: DS1:`STIRR`'s O/P

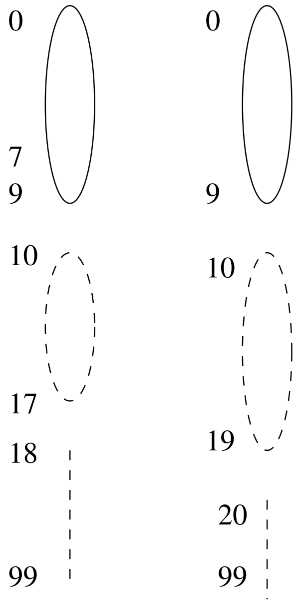

Figure 5: DS2:`STIRR`'s O/P

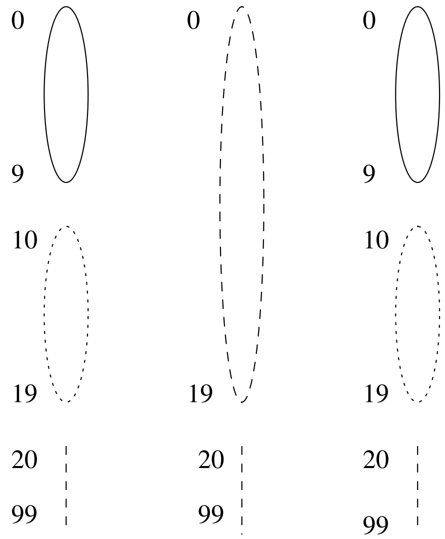

Figure 6: DS3:`STIRR`'s O/P

The cluster-projections for `DS1` were recognized by `STIRR` on the first non-principal basin (b_2) on every attribute (as shown in Figure 4). When run on the dataset `DS2`, the first non-principal basin (b_2) on A_1 identifies the two groups: $\{0, \ldots, 9\}$ and $\{10, \ldots, 17\}$ (as shown in Figure 5). The second non-principal basin (b_3) on A_1 identifies the following two groups: $\{0, \ldots, 6\}$ and $\{7, \ldots, 17\}$. Thus, no basin identifies the overlap between the cluster-projections. It may be possible to identify such overlaps through a non-trivial post processing step. However, it is not clear how many basins are required and how to recognize that cluster-projections overlap from the weights on attribute values. Moreover, we believe that any such post-processing step itself will be similar to the `CACTUS` algorithm. The result of running `STIRR` on the dataset `DS3` is shown in Figure 6. `STIRR` merged the two cluster-projections on the second attribute, possibly because one of the cluster-projections participates in more than one cluster.

From these experiments, we conclude that `STIRR` fails to discover the following classes of clusters: (1) clusters consisting of overlapping cluster-projections on any attribute, (2) clusters where two or more clusters share the same cluster-projection. However, intuitively, these two classes of clusters are valid classes of clusters, and our cluster definition includes these classes. `CACTUS` correctly discovers all the implanted clusters from the datasets `DS1`, `DS2`, and `DS3`. Thus, our definition of a cluster and hence `CACTUS`, which discovers all clusters allowed by our definition, seems to identify a broader class of clusters than that discovered by `STIRR`. Since it is not possible to characterize clusters discovered by `STIRR`, we could not construct any example datasets from which `CACTUS` does not retrieve the expected clusters and `STIRR` does.

4 `CACTUS`

In this section, we describe our three-phase clustering algorithm `CACTUS`. The central idea behind `CACTUS` is that a summary of the entire dataset is sufficient to compute a set of "candidate" clusters which can then be validated to determine the actual set of clusters. `CACTUS` consists of three phases: *summarization*, *clustering*, and *validation*. In the summarization phase, we compute the summary information from the dataset. In the clustering phase, we use the summary information to discover a set of candidate clusters. In

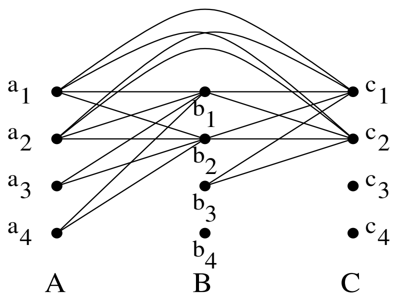

Figure 7: Similarity Graph on A, B, C

A w.r.t. B	B w.r.t. C	C w.r.t. B
a_1, a_2: 2	b_1, b_2: 2	c_1, c_2: 3
a_1, a_3: 2	b_1, b_3: 2	c_1, c_3: 2
a_1, a_4: 2	b_2, b_3: 2	
a_2, a_3: 2		
a_2, a_4: 2		
a_3, a_4: 2		

Figure 8: Intra-attribute Summaries

the validation phase, we determine the actual set of clusters from the set of candidate clusters. We introduce a hypothetical example which we use throughout the paper to illustrate the successive phases in the algorithm. Consider a dataset with three attributes A, B, and C with domains $\{a_1, a_2, a_3, a_4\}$, $\{b_1, b_2, b_3, b_4\}$, and $\{c_1, c_2, c_3, c_4\}$, respectively. Let the similarity graph derived from the dataset be as shown in Figure 7.

4.1 Summarization Phase

In this section, we describe the summarization phase of CACTUS. We show how to efficiently compute the inter-attribute and the intra-attribute summaries, and then describe the resource requirements for maintaining these summaries.

Categorical attributes usually have *small* domains. Typical categorical attribute domains considered for clustering consist of less than a hundred or, rarely, a thousand attribute values.[5] An important implication of the compactness of categorical domains is that the inter-attribute summary Σ_{ij} for any pair of attributes A_i and A_j fits into main memory because the number of all possible attribute value pairs from A_i and A_j equals $|\mathcal{D}_i| \cdot |\mathcal{D}_j|$. For the rest of this section, we assume that the inter-attribute summary of any pair of attributes fits easily into main memory. (We will give an example later to support this assumption, and to show that typically inter-attribute summaries for many pairs of attributes together fit into main memory.) However, for the sake of completeness, we extend our techniques in Section 5 to handle cases where this trait is violated. The same argument holds for the intra-attribute summaries as well.

4.1.1 Inter-attribute Summaries

We now discuss the computation of the inter-attribute summaries. Consider the computation of Σ_{ij}, $i \neq j$. We initialize a counter to zero for each pair of attribute values $(a_i, a_j) \in \mathcal{D}_i \times \mathcal{D}_j$, and start scanning

[5]For instance, the maximum number of values assumed by a categorical attribute of any dataset in the popular UCI Machine Learning test suite is 100 (for the Pendigits dataset) [CBM98].

the dataset D. For each tuple $t \in D$, we increment the counter for the pair $(t.A_i, t.A_j)$. After the scan of D is completed, we compute σ^* by setting to zero all counters whose value is less than the threshold $\kappa_{ij} = \alpha \cdot \frac{|D|}{|\mathcal{D}_i| \times |\mathcal{D}_j|}$. Thus, counts of only the strongly connected pairs are retained. The number of strongly connected pairs is usually much smaller than $|\mathcal{D}_i| \cdot |\mathcal{D}_j|$. Therefore, the set of strongly connected pairs can be maintained in specialized data structures designed for sparse matrices [DER86].[6]

We now present a hypothetical example to illustrate the resource requirements of the simple strategy described above. Consider a dataset with 50 attributes each consisting of 100 attribute values. With 100 MB of main memory (easily available on current desktop systems), assuming that each counter requires 4 bytes we can maintain counters for $2500 = \frac{100 \times 10^6}{100 \times 100 \times 4}$ attribute pairs simultaneously. With 50 attributes, we have to evaluate 1225 attribute pairs. Therefore, we can compute all inter-attribute summaries together in just one scan of the dataset. The computational and space requirements here are similar to that of obtaining counts of pairs of items while computing frequent itemsets [AMS$^+$96].

Quite often, a single scan is sufficient for computing Σ_{IJ}. In some cases, we may need to scan D multiple times—each scan computing Σ_{ij} for a (different) set of (i, j) pairs. The computation of the inter-attribute summaries is CPU-intensive, especially when the number of attributes is high, because for each tuple in the dataset, we have to increment $\frac{n(n-1)}{2}$ counters where n is the number of attributes. Even if we require multiple scans of the dataset, the I/O time for scanning the dataset goes up but the total CPU time—for incrementing the counters—remains the same. Since the CPU time dominates the overall summary-construction time, the relative increase due to multiple scans is not significant. For instance, consider a dataset of 1 million tuples defined on 50 attributes, each consisting of 100 attribute values. Experimentally, we found that the total time for computing the inter-attribute summaries of the dataset with 1 million tuples is 1040 seconds, whereas a scan of the dataset takes just 28 seconds. Suppose we partition all the 1225 pairs of attributes into three groups consisting of 408, 408, and 409 pairs respectively. The computation of the inter-attribute summaries of attribute pairs in each group requires a scan of the dataset. The total computation time is around 1096 seconds, which is only slightly higher than the time for a single scan.

4.1.2 Intra-attribute Summaries

In this section, we describe the computation of the intra-attribute summaries. We again exploit the characteristic that categorical domains are very small and thus assume that the intra-attribute summary of any attribute A_i fits in main memory. Our procedure for computing Σ_{ii}^j reflects the evaluation of the following SQL query:

Select	$T1.A, T2.B, count(*)$
From	Σ_{ij} as T1(A,B), Σ_{ij} as T2(A,B)
Where	T1.A $\neq$ T2.A and T1.B = T2.B
Group by	T1.A, T2.A
Having	$count > 0$;

[6] In our current implementation, we maintain the counts of strongly connected pairs in an array and do not optimize for space.

The above query joins Σ_{ij} with itself to compute the set of attribute value pairs of A_i strongly connected to each other with respect to A_j.[7] Since Σ_{ij} fits in main memory the self-join and hence the computation of Σ_{ii}^j is very fast. We will observe in the next section that, at any stage of our algorithm, we only require Σ_{ii}^j for a particular pair of attributes A_i and A_j. Therefore, we compute Σ_{ii}^j, $(j \neq i)$, for each (i, j) pair whenever it is required.

Consider the example shown in Figure 7. (We use the notation Σ_{XY} to denote the inter-attribute summary between attributes X and Y.) The inter-attribute summaries Σ_{AB}, Σ_{BC}, and Σ_{AC} correspond to the edges of the similarity graph on A, B, C shown in the figure. The intra-attribute summaries Σ_{AA}^B, Σ_{BB}^C, Σ_{CC}^B are shown in Table 8.

4.2 Clustering Phase

In this section, we describe the two-step clustering phase of CACTUS that uses the attribute summaries to compute candidate clusters in the data. In the first step, we analyze each attribute to compute all cluster-projections on it. In the second step, we synthesize, in a level-wise manner, candidate clusters on sets of attributes from the cluster-projections on individual attributes. That is, we determine candidate clusters on a pair of attributes, then extend the pair to a set of three attributes, and so on. We now describe each step in detail.

4.2.1 Computing Cluster-Projections on Attributes

Let $A_1, \ldots, A_n$ be the set of attributes with domains $\mathcal{D}_1, \ldots, \mathcal{D}_n$ respectively. The central idea for computing all cluster-projections on an attribute is that a cluster $\langle C_1, \ldots, C_n \rangle$ over the set of attributes $\{A_1, \ldots, A_n\}$ induces a sub-cluster over any attribute pairs (A_i, A_j), $i \neq j$. In addition, the cluster-projection C_i on A_i of the cluster C is the intersection of a set of cluster-projections on A_i of 2-clusters over attribute pairs (A_i, A_j), $j \neq i$. For example, the cluster-projection $\{b_1, b_2\}$ on the attribute B in Figure 7 is the intersection of $\{b_1, b_2, b_3\}$ (the cluster-projection on B of the 2-cluster $\langle \{b_1, b_2, b_3\}, \{c_1, c_2\} \rangle$) and $\{b_1, b_2\}$ (the cluster-projection on B of the 2-cluster $\langle \{a_1, a_2, a_3, a_4\}, \{b_1, b_2\} \rangle$). We formalize the idea in the following lemma.

Lemma 4.1 Let $C = \langle C_1, \ldots, C_n \rangle$ be a cluster on the set of attributes $\{A_1, \ldots, A_n\}$. Then, the following properties are true.
(1) For all $i \neq j, i, j \in \{1, \ldots, n\}$, $\langle C_i, C_j \rangle$ is a sub-cluster over the pair of attributes (A_i, A_j).
(2) There exists a set $\{C_i^j : j \neq i$ and $\langle C_i^j, C_j^i \rangle$ is a 2-cluster over $(A_i, A_j)\}$ such that $C_i = \cap_{j \neq i} C_i^j$.

Lemma 4.1 motivates the following two-step approach. In the first *pairwise cluster-projection* step, we cluster each attribute A_i with respect to every other attribute A_j, $j \neq i$ to find all cluster-projections on A_i of 2-clusters over (A_i, A_j). In the second *intersection* step, we compute all the cluster-projections on A_i of clusters over $\{A_1, \ldots, A_n\}$ by intersecting sets of cluster-projections from 2-clusters computed in the first

[7] For an exposition of join processing, see any standard textbook on database systems, e.g., [Ram97].

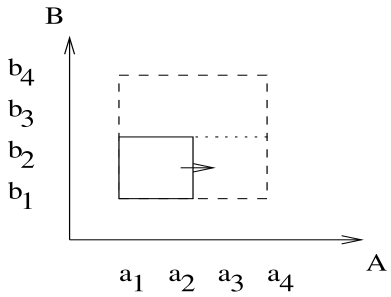

Figure 9: Extending $\{a_1, a_2\}$ w.r.t. B

step. However, the problem of computing cluster-projections of 2-clusters in the pairwise cluster-projection step is at least as hard as the NP-complete *clique* problem [GJ79].[8] The following lemma formalizes the computational complexity. The proof is given in Appendix A.

Lemma 4.2 Let A_i and A_j be two attributes. Let $\mathcal{G}_i^j$ be the similarity graph on A_i with respect to A_j. The problem of computing all cluster-projections on A_i of 2-clusters over (A_i, A_j) is NP-complete.

To reduce the computational complexity of the cluster-projection problem, we exploit the following property which, we believe, is usually exhibited by clusters in the categorical domain. If a cluster-projection C_i on A_i of one (or more) cluster(s) is larger than a fixed positive integer, called the *distinguishing number* (denoted κ), then it consists of a small identifying set—which we call the *distinguishing set*—of attribute values such that they will not *together* be contained in any other cluster-projection on A_i. Thus, the distinguishing set distinguishes C_i from other cluster-projections on A_i. Note that a proper subset of the distinguishing set may still belong to another cluster-projection, and that two distinct clusters may share an identical cluster-projection (as in Figure 1).

We believe that the distinguishing subset assumption holds in almost all cases. Even for the most restrictive version, which occurs when the distinguishing number is 1 and all cluster-projections of the set of clusters are distinct, the assumption only requires that each cluster consist of a set of attribute values—one on each attribute—that does not belong to any other cluster. For the example in Figure 7, the sets $\{a_1\}$ or $\{a_2\}$ identify the cluster-projection $\{a_1, a_2\}$ on the attribute A. We now formally state the assumption.

Distinguishing Subset Assumption: Let C_i and C_i' each of size greater than κ be two distinct cluster-projections on the attribute A_i. Then there exist two sets S_i and S_i' such that

$$|S_i| \leq \kappa, S_i \subset C_i, |S_i'| \leq \kappa, S_i' \subset C_i', \text{ and } S_i \not\subset C_i', S_i' \not\subset C_i$$

We call κ the *distinguishing number*.

[8] A *clique* in $\mathcal{G}_i^j$ is a set of vertices that are connected to each other by edges with non-zero weights. Given a graph $\mathcal{G} = \langle V, \mathcal{E} \rangle$ and a constant J, the clique problem determines if $\mathcal{G}$ consists of a clique of size at least J.

Pairwise Cluster-Projections

The procedure for computing cluster-projections on A_i of 2-clusters over the attribute pair (A_i, A_j) consists of two steps. In the first step, we find all possible distinguishing sets (of size less than or equal to κ) on A_i. In the second step, we extend with respect to A_j some of these distinguishing sets to compute cluster-projections on A_i. In this section, we sometimes write "cluster-projection on A_i" instead of a "cluster-projection on A_i of a 2-cluster over (A_i, A_j)."

Distinguishing Set Computation: In the first step, we rely on the following two properties to find all possible distinguishing sets on A_i. (1) A distinguishing set is a clique in the similarity graph $\mathcal{G}_i^j$. (2) Any subset of a clique is also a clique (*monotonicity property*). These two properties allow a level-wise clique generation similar to the candidate generation in apriori [AMS$^+$96]. That is, we first compute all cliques of size 2, which are then used to compute cliques of size 3, and so on until all cliques of size less than or equal to κ are computed.

Let C_k denote the set of all cliques of size less than or equal to k. We give an inductive description of the procedure to generate the set C_k. The base case C_2 when $k = 2$ consists of all pairs of strongly connected attribute values in $\mathcal{D}_i$. These pairs can easily be found from Σ_{ii}^j. The set C_{k+1} is computed from the set C_k ($k \geq 2$) by "joining" C_k with itself. The join is the subset join—used in the candidate generation step of the apriori algorithm for frequent itemset computation [AMS$^+$96]. We also remove all the candidates in C_{k+1} that contain a proper k-subset not in C_k (a la subset pruning in apriori).

Extension Operation: In the second step, we "extend" with respect to A_j some of the candidate distinguishing sets computed in the first step to compute cluster-projections on A_i. The intuition behind the extension operation is illustrated in Figure 9. Suppose we want to extend $\{a_1, a_2\}$ on A with respect to B. We compute the set $\{b_1, b_2\}$ of attribute values on B strongly connected with $\{a_1, a_2\}$. We then extend $\{a_1, a_2\}$ with the set of all other values $\{a_3, a_4\}$ on A that is strongly connected with $\{b_1, b_2\}$.

Informally, the extension of a distinguishing set $S \subset \mathcal{D}_i$ adds to S all attribute values in $\mathcal{D}_i$ that are strongly connected with the set of all attribute values in $\mathcal{D}_j$ that S is strongly connected with. We now introduce some terminology to formally describe the extension operation.

Definition 4.1 Let A_i and A_j be two attributes with domains $\mathcal{D}_i$ and $\mathcal{D}_j$. Let $\mathcal{DS}_i^j$ be a set of candidate distinguishing sets, with respect to A_j, on attribute A_i. Let $\mathcal{CS}_i^j$ be the set of cluster-projections on A_i of 2-clusters over (A_i, A_j). The *sibling set* S_j^i of $S_i \in \mathcal{DS}_i^j$ with respect to the attribute A_j is defined as follows:

$$S_j^i = \{a_j \in \mathcal{D}_j : \text{ for all } a_i \in S_i, \sigma^*(a_i, a_j) > 0\}$$

$|S_i^j|$ is called the *sibling strength* with respect to A_j of the distinguishing set S_i.
The *subset flag* of $S_i \in \mathcal{DS}_i^j$ with respect a collection of sets $\mathcal{C}_s$ is said to be *set* (to 1) if there exists a set $S \in \mathcal{C}_s$ such that $S_i \subseteq S$. Otherwise, the subset flag of S_i is not set.
The *participation count* of $S_i \in \mathcal{DS}_i^j$ with respect to $\mathcal{C}_s$ is the sum of the sibling strengths with respect to A_j of all supersets of S_i in $\mathcal{C}_s$.

Informally, the subset flag and the participation count serve the following two purposes. First, a cluster-projection may consist of more than one distinguishing set in $\mathcal{DS}_i^j$. Therefore, if we extend each set in $\mathcal{DS}_i^j$ a particular cluster-projection may be generated several times, once for each distinguishing set it contains. To avoid the repeated generation of the same cluster-projection, we associate with each distinguishing set a subset flag. The subset flag indicates whether the distinguishing set is a subset of an existing cluster-projection in $\mathcal{CS}_i^j$. Therefore, if the subset flag is set then the distinguishing set need not be extended. For the example shown in Figure 7, the distinguishing sets $\{a_1\}$ and $\{a_2\}$ on A can both be extended to the cluster-projection $\{a_1, a_2\}$. Second, the distinguishing subset assumption applies only to cluster-projections of size greater than κ. Therefore, a clique of size less than or equal to κ may be a cluster-projection on its own even though it may be a subset of some other cluster-projection. To recognize such small cluster-projections, we associate a participation count with each distinguishing set. If the participation count of a distinguishing set with respect to $\mathcal{CS}_i^j$ is equal to its sibling strength then it cannot be a cluster-projection on its own. Otherwise, it may be a small cluster-projection.

Algorithm 4.1 Extend($\mathcal{DS}_i^j, \Sigma_{ij}$)

/* Output: $\mathcal{CS}_i^j$ */

begin

/* Initialization */

$\mathcal{CS}_i^j = \phi$

Reset the subset flags and the participation counts of all distinguishing sets in $\mathcal{DS}_i^j$ to zero

foreach $S_i \in \mathcal{DS}_i^j$

if the subset flag of S_i is not set then

Extend S_i to C_i^S

Set the subset flags and increment by the sibling strength of S_i the participation counts of all subsets of C_i^S in $\mathcal{DS}_i^j$.

end /*if*/

end /*for*/

Identify and add small cluster-projections (of size $\leq \kappa$) to $\mathcal{CS}_i^j$

end

The pseudocode for the computation of $\mathcal{CS}_i^j$ is shown in Figure 4.1. Below, we describe each step in detail.

Initialization: The first three steps initialize the procedure: we set $\mathcal{CS}_i^j = \phi$, and all distinguishing sets in $\mathcal{DS}_i^j$ have their subset flags and their participation counts reset to zero.

Extending S_i: Let S_i^j be the sibling set of S_i with respect to A_j. Let C_i^S be the sibling set of S_i^j with respect to A_i. Then, we extend S_i to the cluster-projection C_i^S. Add C_i^S to $\mathcal{CS}_i^j$.

Prune subsets of C_i^S: Suppose C_i^S was extended from S_i. Then, by definition, subsets of C_i^S cannot be the distinguishing sets of other cluster projections on A_i. Therefore, we set (to 1) all subset flags of subsets of C_i^S (including S_i) in $\mathcal{DS}_i^j$. The participation count of each of these subsets is also increased by $|S_i^j|$—the sibling strength of S_i.

Identifying small cluster-projections: While extending distinguishing sets, we only choose sets whose subset flags are not set. We check if each unextended distinguishing set S_i whose subset flag is set can be a small (of size less than κ) cluster-projection. If the participation count of S_i equals its sibling strength, then S_i cannot be a cluster-projection on its own. Otherwise, S_i may be a cluster-projection. Therefore, we add S_i to $\mathcal{CS}_i^j$.

Note that the computation of cluster-projections on A_i requires only the inter-attribute summary Σ_{ij} and the intra-attribute summary Σ_{ii}^j. Since Σ_{ij} and Σ_{ii}^j fit into main memory, the computation is very fast.

Intersection of Cluster-projections

Informally, the intersection step computes the set of cluster-projections on A_i of clusters over $\{A_1, \ldots, A_n\}$ by successively joining sets of cluster-projections on A_i of 2-clusters over attribute pairs (A_i, A_j), $j \neq i$. For describing the procedure, we require the following definition.

Definition 4.2 Let $\mathcal{S}_1$ and $\mathcal{S}_2$ be two collections of sets of attribute values on A_i. We define the *intersection join* $\mathcal{S}_1 \sqcap \mathcal{S}_2$ between $\mathcal{S}_1$ and $\mathcal{S}_2$ as follows:

$$\mathcal{S}_1 \sqcap \mathcal{S}_2 = \{s : \text{there exist } s_1 \in \mathcal{S}_1 \text{ and } s_2 \in \mathcal{S}_2 \text{ such that } s = s_1 \cap s_2 \text{ and } |s| > 1\}$$

Let $\mathcal{CS}_i^j$ be the set of cluster-projections on A_i with respect to A_j, $j \neq i$. Let $j_1 = 1$ if $i > 1$, else $j_1 = 2$. Starting with $S = \mathcal{CS}_i^{j_1}$, the intersection step executes the following operation for all $k \neq i$.

$$S = S \sqcap \mathcal{CS}_i^k, \text{ if } k \neq i$$

The resulting set S is the set of cluster-projections on A_i of clusters over $\{A_1, \ldots, A_n\}$. Besides being a main-memory operation, the number of cluster-projections on A_i with respect to any other attribute A_j is usually small; therefore, the intersection step is quite fast.

Further optimizations are possible over the basic strategy described above for computing cluster projections. For instance, we can combine the computation of $\mathcal{CS}_i^j$ and that of $\mathcal{CS}_j^i$ because, for each cluster-projection in $\mathcal{CS}_i^j$, we compute its sibling set which is a cluster-projection in $\mathcal{CS}_j^i$. The second optimization improves the computation of cluster-projections. Instead of computing cluster-projections on the attribute A_i with respect to every other attribute it may, in practice, be sufficient to do so with respect to a few attributes. However, we do not consider such optimizations because the clustering phase takes a small fraction (less than 10%) of the time taken by the summarization phase. (Our experiments in Section 6 confirm this observation.)

4.2.2 Level-wise Synthesis of Clusters

In this section, we describe the synthesis of candidate clusters from the cluster-projections on individual attributes (computed as described in Section 4.2). The central idea is that a cluster on a set of attributes induces a sub-cluster on any subset of the attributes (*monotonicity property*). The monotonicity property follows directly from the definition of a cluster. We also exploit the fact that we want to compute clusters over the set of all attributes $\{A_1, \ldots, A_n\}$. Informally, we start with cluster-projections on A_1 and then extend them to clusters over (A_1, A_2), then to clusters over (A_1, A_2, A_3), and so on.

Let $\mathcal{C}_i$ be the set of cluster-projections on the attribute A_i, $i = 1, \ldots, n$. Let $\mathcal{C}^k$ denote the set of candidate clusters defined over the set of attributes $A_1, \ldots, A_k$. Therefore, $\mathcal{C}^1 = \mathcal{C}_1$. We successively generate $\mathcal{C}^{k+1}$ from $\mathcal{C}^k$ until $\mathcal{C}^n$ is generated or $\mathcal{C}^{k+1}$ is empty for some $k+1 < n$. The generation of $\mathcal{C}^{k+1}$ from $\mathcal{C}^k$ proceeds as follows. Set $C^{k+1} = \phi$. For each element $c^k = \langle c_1, \ldots, c_k \rangle \in \mathcal{C}^k$, we attempt to augment c^k with a cluster projection c_{k+1} on the attribute A_{k+1}. If for all $i \in \{1, \ldots, k\}$, $\langle c_i, c_{k+1} \rangle$ is a sub-cluster on (A_i, A_{k+1})—which can be checked by looking up $\Sigma_{i(k+1)}$—we augment c^k to generate $c^{k+1} = \langle c_1, \ldots, c_{k+1} \rangle$ and add c^{k+1} to $\mathcal{C}^{k+1}$.

For the example in Figure 7, the computation of the set of candidate clusters proceeds as follows. We start with the set $\{a_1, a_2\}$ on A. We then find the candidate 2-cluster $\{\langle \{a_1, a_2\}, \{b_1, b_2\} \rangle\}$ over the attribute pair (A, B), and then the candidate 3-cluster $\{\langle \{a_1, a_2\}, \{b_1, b_2\}, \{c_1, c_2\} \rangle\}$ over $\{A, B, C\}$.

4.3 Validation

We now describe a procedure to compute the set of actual clusters from the set of candidate clusters. Some of the candidate clusters may not have enough support because some of the 2-clusters that combine to form a candidate cluster may be due to different sets of tuples. To recognize such false candidates, we check if the support of each candidate cluster is greater than the required threshold. Only clusters whose support on D passes the threshold requirement are retained.

After setting the supports of all candidate clusters to zero, we start scanning the dataset D. For each tuple $t \in D$, we increment the support of the candidate cluster to which t belongs. (Because the set of clusters correspond to disjoint interval regions, t can belong to at most one cluster.) At the end of the scan, we delete all candidate clusters whose support in the dataset D is less than the required threshold: α times the expected support of the cluster under the attribute-independence assumption.

By construction, `CACTUS` discovers all clusters that satisfy our cluster definition, and hence the following theorem follows.

Theorem 4.1 Given that the distinguishing subset assumption holds, `CACTUS` finds all and only those clusters that satisfy Definition 3.3.

5 Extensions to Large Attribute Value Domains

In this section, we extend CACTUS to handle unusually large attribute value domains.[9] Until now, we assumed that the domains of categorical attributes are such that the inter-attribute summary of any pair of attributes and the intra-attribute summary of any attribute fits in main memory. Even though we argued with compelling examples (UCI Machine Learning repository) that most real datasets satisfy this assumption, for the sake of completeness, we modify the summarization phase of CACTUS to handle arbitrarily large domain sizes.

Recall that we are only interested in strongly connected pairs of attribute value pairs in the summary information. For large domain sizes, the number of strongly connected (with respect to another attribute A_j) attribute value pairs of an attribute A_i relative to the the number of all possible attribute value pairs is very small. We exploit this observation to collapse disjoint sets of attribute values into a single attribute value thus creating a new transformed domain with a reduced domain size. The intuition is that if a pair of attribute values in the original domain are strongly connected, then the corresponding pair of transformed attribute values are also strongly connected provided the threshold for strong connectivity between attribute values involving the tranformed domain is the same as that of using the original domain.

Let A_i be an attribute with an unusually large domain $\mathcal{D}_i$. Without loss of generality, let $\mathcal{D}_i$ be the set $\{1, \ldots, |\mathcal{D}_i|\}$. Let $M < |\mathcal{D}_i|$ be the maximum number of attribute values per attribute so that the inter-attribute summaries and the intra-attribute summaries involving any attribute fit in main memory. Let $c = \lceil \frac{|\mathcal{D}_i|}{M} \rceil$. We construct $\mathcal{D}'_i$ of size M from $\mathcal{D}_i$ by mapping for a given $x \in \{0, \ldots, M-1\}$, the set of attribute values $\{x \cdot c + 1, \ldots, x \cdot c + c\}$ to the value $x + 1$. Formally,

$$\mathcal{D}'_i = \{f(1), \ldots, f(|\mathcal{D}_i|)\}, \text{ where } f(i) = \lfloor \frac{i}{k} \rfloor + 1$$

We set the threshold for the strong connectivity involving attribute values in $\mathcal{D}'_i$ to the exact same value if $\mathcal{D}_i$ was being used. We then compute the inter-attribute summaries involving A_i using the transformed domain $\mathcal{D}'_i$ instead of $\mathcal{D}_i$. For each attribute value $a'_i \in \mathcal{D}'_i$ that participates in a strongly connected pair (a'_i, a_j) ($a_j \in \mathcal{D}_j$, $j \neq i$), we expand a'_i to the set of all attribute values $\{a'_i \cdot c + 1, \ldots, a'_i \cdot c + c\} \subset \mathcal{D}_i$ that map into a'_i and form the pairs $(a'_i \cdot c + 1, a_j), \ldots, (a'_i \cdot c + c, a_j)$. We then scan the dataset D to count the supports of all these pairs, and select the strongly connected pairs among them; they constitute the inter-attribute summary Σ_{ij}.

The number of new pairs whose supports are to be counted is less than or equal to $c \cdot |\Sigma_{ij}|$ where $|\Sigma_{ij}|$ represents the number of strongly connected pairs in $\mathcal{D}_i \times \mathcal{D}_j$. If this set of pairs is still larger than main memory, we can repeat the above transformation trick. However, we believe that such a repeated application will rarely be necessary.

[9]Due to space constraints, we describe the extension to find clusters in subspaces in Appendix B.

6 Performance Evaluation

In this section, we show the results of a detailed evaluation of the speed and scalability of CACTUS on synthetic and real datasets and we examined whether the clusters discovered where intuitive and sensible. We also compared the performance of CACTUS with the performance of STIRR.[10] Our results show that CACTUS is very fast and scalable; it outperforms STIRR by a factor between 3 and 10.

6.1 Synthetic Datasets

We first used synthetic datasets for our experiments. The test datasets were generated using the data generator developed by Gibson et al. [GKR98] to evaluate STIRR. (See Section 3.2 for a description of the data generator.) We set the number of tuples to 1 million, the number of attributes to 10 and the number of attribute values for each attribute to 100. In all datasets, the cluster-projections on each attribute were $[0, 9]$ and $[10, 19]$ (as shown in Figure 1). Unless otherwise mentioned, we set the value of α at 3, and the value of the distinguishing number κ to 2. For STIRR, we fixed the number of iterations to be 10—as suggested by Gibson et al. [GKR98].

As discussed in Section 3.2, CACTUSdiscovers a broader class of clusters than STIRR, and our experiments confirmed this.

Figure 10 shows the scalability of CACTUS and STIRR while increasing the number of tuples from 1 to 5 million. In Figure 11, the number of attributes is increased from 4 to 50. For the results shown in Figure 12, the number of attribute values is increased from 50 to 1000 while fixing the number of attributes at 4. While varying the number of attribute values, we assumed that until 500 attribute values, the inter-attribute summaries would fit into main memory; for a larger number of attribute values we took the multi-layered approach described in Section 5. In all cases, CACTUS is 3 to 10 faster than STIRR, despite scanning the dataset two times for constructing the inter-attribute summaries in some cases.

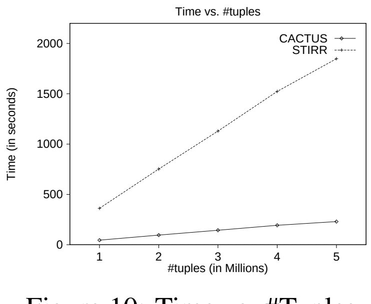

Figure 10: Time vs. #Tuples

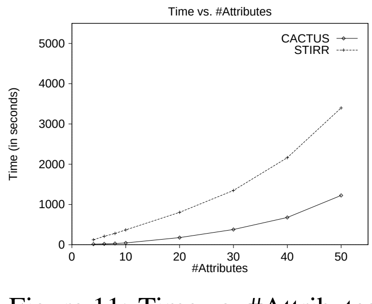

Figure 11: Time vs. #Attributes

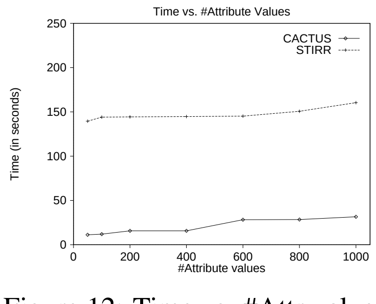

Figure 12: Time vs. #Attr-values

[10] We intend to compare CACTUS and ROCK after our ongoing implementation of ROCK is complete.

First Author	First Author (contd.)	Second Author	Second Author (contd.)
Katz, Stonebraker, Wong	Ceri, Navathe	Katz, Wong	Ceri, Navathe
DeWitt, Hsiao	Abiteboul, Grumbach	DeWitt, David	Vianu, Grumbach
DeWitt, Ghandeharizadeh	Korth, Levy	DeWitt, Ghandeharizadeh	Silbershatz, Levy
Kanellakis, Beeri, Vardi	Agrawal, Gehani	Abiteboul, Beeri	Jagadish, Gehani
Ramakrishnan, Beeri	Chen, Hua, Su	Beeri, Srivastava	Su, Chen, Chu
Bancilhon, Kifer	Chen, Hua, Lam	Ramakrishnan, Kim	Su, Lee
Afrati, Cosmadakis	Collmeyer, King, Shemer	Papadimitriou, Cosmadakis	Collmeyer, Shemer
Alonso, Barbara, GarciaMolina	Copeland, Lipovski, Su	GarciaMolina, Barbara	Su, Lipovski, Copeland
Devor, Elmasri	Cornell, Dan, Iyer, Yu	Devor, ElMasri, Weeldreyer	Yu, Dias
Barsolou, Keller, Wiederhold	Chang, Gupta	Keller, Wiederhold	Lee, Cheng
Barsalou, Keller, Shalom	Fischer, Griffeth, Lynch	Keller, Wiederhold	Griffeth, Fischer

Table 1: 2-clusters on the pair of first author and second author attributes

ACMSIGMOD Management, VLDB, ACM TODS, ICDE, ACMSIGMOD Record
ACMTG, COMPGEOM, FOCS, GEOMETRY, ICALP, IPL, JCSS, JSCOMP, LIBTR, SICOMP, TCS, TR
PODS, ALGORITHMICA, FOCS, ICALP, INFCTRL, IPL, JCSS, SCT, SICOMP, STOC

Table 2: Cluster-projections on Conference w.r.t. the First Author

6.2 Real Datasets

In this section, we discuss an application of `CACTUS` to a combination of two sets of bibliographic entries. The results from the application show that `CACTUS` finds intuitively meaningful clusters from the dataset thus supporting our definition of a cluster.

The first set consists of 7766 bibliographic entries for articles related to database research [Wie] and the second set consists of 30919 bibliographic entries for articles related to Theoretical Computer Science and related areas [Sei]. For each article, we use the following four attributes: the first author, the second author, the conference or the journal of publication, and the year. If an article is singly-authored then the author's name is repeated as the second author as well. The sizes of the first author, the second author, the conference, and the year attribute domains for the database-related, the theory-related, and the combined sets are $\{3418, 3529, 1631, 44\}$, $\{8043, 8190, 690, 42\}$, and $\{10212, 10527, 2315, 52\}$ respectively. Note that for these domains, some of the inter-attribute summaries and the intra-attribute summaries—especially those involving the first author and the second author dimensions—do not fit in main memory. However, we choose this particular dataset because it is easier to judge the validity of the resulting clusters (than for some other publicly available datasets, e.g., the `MUSHROOM` dataset from the UCI Machine Learning repository). We combine the two sets together to check if `CACTUS` is able to identify the differences and the overlap between the two communities.

We now discuss an important implication of the unusually large domains of some of the categorical attributes and the relatively small number of tuples in this dataset. Because of the large domain sizes, the expected support under the attribute-independence assumption for any pair of attribute values from two different attributes is almost *zero*: the maximum is 0.32 for the (conference, year) pair. Therefore, for a tuple $t = \langle a_1, a_2, c, y\rangle$ in the dataset, the pairs $(a_1, a_2), (a_1, c), (a_1, y), (a_2, c), (a_2, y), (c, y)$ are all strongly connected. This property has two implications. First, the strong-connectedness relation between attribute values is an equivalence relation. Second, the cluster-projections on any attribute are determined by the 2-clusters involving at least one of the first author and the second author attributes. Therefore, we first present the cluster-projections of 2-clusters over the (first author, second author) attribute pair. Then, we present cluster-projections on the conference and the year attributes with respect to the first author attribute.

Table 6.1 shows some of the 2-clusters on the first author and the second author attribute pair. We only present the database-related cluster-projections to illustrate that CACTUS identifies the differences between the two communities. We verified the validity of each cluster-projection by querying on the *Database Systems and Logic Programming* bibliography at the web site maintained by Michael Ley [Ley]. Similar cluster-projections identifying groups of theory-related researchers as well as groups that contribute to both fields also exist. We show some cluster-projections corresponding to the latter two types in Appendix C.

Table 2 shows some of the cluster-projections on the conference attribute computed with respect to the first author attribute. The first row consists exclusively of a group of database-related conferences, the second consists exclusively of theory-related conferences, and the third a mixture of both reflecting a considerable overlap between the two communities.

7 Conclusions and Future Work

In this paper, we formalized the definition of a cluster when the data consists of categorical attributes, and then introduced a fast summarization-based algorithm CACTUS for discovering such clusters in categorical data. We then evaluated our algorithm against both synthetic and real datasets.

In future, we intend to extend CACTUS in the following three directions. First, we intend to relax the cluster definition by allowing sets of attribute values on each attribute which are "almost" strongly connected to each other. That is, each attribute value is strongly connected to a large percentage, say 90%, of the attribute values in the sets on other attributes. Second, motivated by the observation that inter-attribute summaries can be incrementally maintained under addition and deletion of tuples, we intend to derive an incremental clustering algorithm from CACTUS. Third, we intend to "rank" the clusters based on a measure of interestingness, say, some function of the support of a cluster.

Acknowledgements: We thank Prabhakar Raghavan for sending us the bibliographic data.

References

[AGGR98] Rakesh Agrawal, Johannes Gehrke, Dimitrios Gunopulos, and Prabhakar Raghavan. Automatic subspace clustering of high dimensional data for data mining. In *Proceedings of the ACM SIGMOD Conference on Management of Data*, 1998.

[AMS+96] Rakesh Agrawal, Heikki Mannila, Ramakrishnan Srikant, Hannu Toivonen, and A. Inkeri Verkamo. Fast Discovery of Association Rules. In Usama M. Fayyad, Gregory Piatetsky-Shapiro, Padhraic Smyth, and Ramasamy Uthurusamy, editors, *Advances in Knowledge Discovery and Data Mining*, chapter 12, pages 307–328. AAAI/MIT Press, 1996.

[BD76] Peter J. Bickel and Kjell A. Doksum. *Mathematical Statistics: Basic Ideas and Selected Topics*. Prentice Hall, 1976.

[CBM98] E. Keogh C. Blake and C.J. Merz. UCI repository of machine learning databases, 1998.

[Cou95] Transaction Processing Performance Council, May 1995. http://www.tpc.org.

[DER86] I.S. Duff, A.M. Erisman, and J.K. Reid. *Direct Methods for Sparse Matrices*. Oxford University Press, 1986.

[GJ79] M. R. Garey and D. S. Johnson. Computers and intractability — A guide to the theory of NP-completeness. *Freeman; Bell Lab, Murray Hill NJ*, 1979.

[GKR98] David Gibson, Jon Kleinberg, and Prabhakar Raghavan. Clustering categorical data: An approach based on dynamical systems. In *Proceedings of the 24th International Conference on Very Large Databases*, pages 311–323, New York City, New York, August 24-27 1998.

[GRS99] Sudipto Guha, Rajeev Rastogi, and Kyuseok Shim. Rock: A robust clustering algorithm for categorical attributes. In *Proceedings of the IEEE International Conference on Data Engineering*, Sydney, March 1999.

[Ley] Michael Ley. Computer science bibliography. http://www.informatik.uni-trier.de/ ley/db/index.html.

[Ram97] Raghu Ramakrishnan. *Database Management Systems*. McGraw Hill, 1997.

[Sei] J. Seiferas. A large bibliography on theory/foundations of computer science. http://liinwww.ira.uka.de/bibliography/Theory/Seiferas.

[Wie] G. Wiederhold. Bibliography on database systems. http://liinwww.ira.uka.de/bibliography/Database/Wiederhold.

A Proofs

Proof for Lemma 4.1

Proof: Property (1) follows trivially from the definition of a cluster. We now prove Property (2). From Property (1), we observe that $C_i \subseteq \cap_{j \neq i} C_i^j$. Suppose the equality does not hold in the above relationship, i.e., $C_i \subset \cap_{j \neq i} C_i^j$. Let $C_i' = \cap_{j \neq i} C_i^j$. From the definition of a cluster, we observe that C_i can be augmented with at least one attribute value in $C_i' - C_i$. Therefore, C is a sub-cluster but not a cluster. □

Proof for Lemma 4.2

Proof: The proof follows from the construction of an instance of the cluster-projection problem from an instance of the clique problem. Let $\mathcal{G} = \langle V, \mathcal{E} \rangle$ be a graph. Create two attributes A_i and A_j each with domain V. For each edge $e = (v_1, v_2) \in \mathcal{E}$, we make v_1 on A_i and v_2 on A_j strongly connected, and v_2 on A_i and v_1 on A_j strongly connected. This construction yields a similarity graph on A_i and A_j which satisfies the following condition. If C is a clique in $\mathcal{G}$ then C is a cluster-projection on A_i with respect to A_j, and vice-versa. Therefore, any algorithm for computing cluster-projections also solves the clique problem. Hence, the cluster-projection problem is NP-complete. □

B Clusters in Subspaces

We extend the CACTUS algorithm to find clusters in subspaces. Recall that a subspace cluster is a cluster on a subset $\mathcal{S}$ of the set of all attributes $\{A_1, \ldots, A_n\}$. The CACTUS algorithm does not discover clusters in subspaces for the following two reasons. First, a subspace cluster C does not span the set of all attributes. Second, the order $A_1, \ldots, A_n$ in which cluster-projections on individual attributes are combined may not be the right order to find C. For instance, if C spans the subspace defined by a set of attributes $\{A_2, A_3, A_4\}$ (when $n \geq 4$) then the level-wise synthesis described in Section 4.2.2 will not find C.

The central idea behind the extension of CACTUS to find subspace clusters is that they satisfy the monotonicity property. That is, a cluster in a subspace $\mathcal{S}$ induces a cluster on any subset of $\mathcal{S}$. The monotonicity property motivates the apriori-style level-wise synthesis of candidate clusters from the cluster-projections on individual attributes. However, we skip the intersection of cluster-projections on an attribute A_i with respect to every other attribute A_j for the following two reasons. First, a cluster in subspace $\mathcal{S}$ may not induce a 2-cluster on a pair of attributes not in $\mathcal{S}$, and hence the intersection of cluster-projections on an attribute in $\mathcal{S}$ with respect to every other attribute returns an empty set. Second, the intersection may cause the loss of maximality (condition (2) in Definition 3.3) of a subspace cluster. For instance, a cluster-projection on A_i with respect to A_j corresponds to a 2-cluster over (A_i, A_j) which, by definition, is a subspace cluster; truncating such a cluster-projection in the intersection step will no longer yield a maximal cluster on (A_i, A_j).

Let $\mathcal{C}_i$ be the set of cluster-projections on the attribute A_i, $i = 1, \ldots, n$ with respect to every attribute $A_j \neq A_i$. Let $\mathcal{C}^k$ denote the set of candidate clusters defined on any set of k-attributes (not necessarily $\{A_1, \ldots, A_k\}$). $\mathcal{C}^1 = \cup_{i=1}^n \mathcal{C}_i$. Using $\mathcal{C}^1$, we generate $\mathcal{C}^2$, and so on. We first describe the computation of $\mathcal{C}^2$

because it is different from that of $\mathcal{C}^k$, $(k > 2)$.

For each possible pair (c_i, c_j) ($c_i \in \mathcal{C}_i, c_j \in \mathcal{C}_j$), we check if (c_i, c_j) is a sub-cluster on the attribute pair (A_i, A_j) by confirming, using Σ_{ij}, that all attribute value pairs in $c_i \times c_j$ are strongly connected. If (c_i, c_j) is a sub-cluster, then it is inserted in the set of 2-clusters $\mathcal{C}^2$.

We compute $\mathcal{C}^{k+1}$ from $\mathcal{C}^k$, $k \geq 2$, by "joining" $\mathcal{C}^k$ with itself. The join is a prefix join similar to the apriori candidate generation [AMS$^+$96]. We successively generate $\mathcal{C}^{k+1}$ from $\mathcal{C}^k$ until $\mathcal{C}^{k+1}$ is empty for some $k + 1 < n$ or$\mathcal{C}^n$ is generated. For each newly generated candidate, we check that all its subsets are also candidate clusters; candidates that do not pass this check are pruned out. (We omit the details of the prefix join and subset pruning, and refer the reader to the original paper [AMS$^+$96].)

For a cluster $c \in \mathcal{C}^k$ in a subspace consisting of k attributes, the above procedure examines $2^k - (k + 1)$ candidates. Depending on the value of k (say, larger than 15), the number of candidate clusters can be prohibitively high. The problem of examining a large number of candidate clusters has been addressed by Agrawal et al. [AGGR98]. They use the *minimum description length* principle to prune the number of candidate clusters. Their techniques are directly applicable in our scenario as well. Therefore, we do not address this problem; instead, we refer the reader to the original paper [AGGR98].

C Results from the Bibliographic Data

First Author	Second Author
Abiteboul, Kolaitis	Vardi, Papadimitriou
Papadimitriou, Vazirani	Yannakakis, Vazirani
Sudarshan, Gopalakrishnan	Ramakrishnan, Rangan
Edelsbrunner, Chazelle	Welzl, Preparata, Sharir, Guibas
Allender, Book	Wilson, Watanabe

Table 3: 2-clusters on the pair of first author and second author attributes