Cross-Relational Clustering with User's Guidance*

Xiaoxin Yin
UIUC
xyin1@uiuc.edu

Jiawei Han
UIUC
hanj@cs.uiuc.edu

Philip S. Yu
IBM T. J. Watson Res. Center
psyu@us.ibm.com

ABSTRACT

Clustering is an essential data mining task with numerous applications. However, data in most real-life applications are high-dimensional in nature, and the related information often spreads across multiple relations. To ensure effective and efficient high-dimensional, cross-relational clustering, we propose a new approach, called CROSSCLUS, which performs cross-relational clustering with user's guidance. We believe that user's guidance, even likely in very simple forms, could be essential for effective high-dimensional clustering since a user knows well the application requirements and data semantics. CROSSCLUS is carried out as follows: a user specifies a clustering task and selects one or a small set of features pertinent to the task. CROSSCLUS extracts *the set of highly relevant features* in multiple relations connected via linkages defined in the database schema, evaluates their effectiveness based on user's guidance, and identifies interesting clusters that fit user's needs. This method takes care of both quality in feature extraction and efficiency in clustering. Our comprehensive experiments demonstrate the effectiveness and scalability of this approach.

1. INTRODUCTION

Clustering is a process of partitioning data objects into groups according to some similarity measure [11, 16, 17, 19]. Most existing methods perform clustering within a single table, and many handle only a small number of dimensions. Recent studies on high-dimensional clustering have developed methods for subspace or projected clustering [1, 2, 6] that aim at finding clusters on subsets of dimensions in a high-dimensional space. However, many of such methods ignore user's expectation but only examine the dissimilarities among data objects. The clusters so generated could be lack of semantic meaning and poor in quality.

Real-life applications often pose additional challenges since data can be not only high-dimensional but also with related information spread across multiple relations. Few studies handle clustering across multiple relations. Consider a task of clustering students in a relational database based on their academic performance (as in the database shown in Figure 1). It may not be meaningful to cluster students based on the attributes like phone number, ssn, and residence address, even if they reside in the same relation. However, some crucial information, such as student's registration, courses taken, and research publications, can be stored across multiple relations. Thus it is necessary to explore clustering across multiple relations.

Based on the above observations, we introduce two new concepts: (1) *user-guided clustering*, and (2) *cross-relational clustering*. The necessity to introduce these two notions is obvious. First, a user often knows well the application requirements and data semantics. Thus a user's guidance, even in a simple form such as *clustering students based on their research areas*, could be essential for effective high-dimensional clustering. Second, crucial information is often stored across multiple relations, and it is difficult for a user to specify all pertinent information. For example, to cluster students according to their research areas (see Figure 1), most crucial information may be stored in several relations, such as *Group*, *Advise*, *Course*, and *Publication*.

To handle such a clustering task, we propose a new methodology, called **cross-relational clustering with user's guidance**, or simply, CROSSCLUS. This methodology explores how to specify and utilize user's guidance in clustering and how to perform **cross-relational clustering**, i.e., *partition the data objects into a set of clusters based on their similarity, utilizing information in multiple relations.*

Clustering in multi-relational environments has been studied in [7, 14, 15], in which the similarity between two objects are defined based on tuples joinable with them. However, there are two problems with such approaches. First, although they provide similarity definitions in multi-relational environments, it is usually very expensive to compute such similarities because an object is often joinable with hundreds or thousands of tuples. Second, there are usually a large number of candidate features in a database (such as in Figure 1), generated from different attributes with different join paths. They cover different aspects of information (e.g., research, grades, address), but only a small portion of them are pertinent to the user's goal. However, all features are used indiscriminately in above approaches, which is unlikely to generate desirable clustering results.
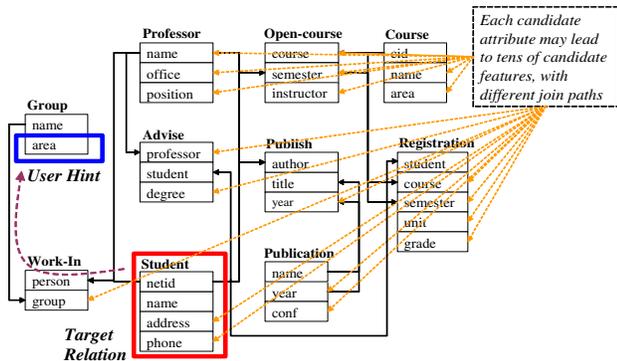
Figure 1: Schema of the CS Dept database

The above problem can be solved by semi-supervised clustering [12, 13, 21], in which a set of similar (or dissimilar) pairs of instances are provided by user. However, this requires a user to have good knowledge about the data and the clustering goal. Moreover, since different users have different clustering goals that can only be expressed by training data, and the complex information in multiple relations may overwhelm the user, it is very burdensome for a user to provide a high quality training set, which is critical to the clustering results.

In this paper we propose a different approach. First, we remove the user's burden of providing training data. In order to express her clustering goal, a user poses a clustering query that consists of a *target relation* and *one or a small set of pertinent attribute(s)*. The target relation may join with each relation via different join paths, which leads to the generation of a huge number of "connected" features with diverse semantic meaning (see Figure 1). Our clustering task is to perform comprehensive analysis across multiple relations to select the set of pertinent features and derive high quality clustering that meets user's expectation.

Example 1. In the CS Dept database in Figure 1, the target relation is *Student*, and the goal is to cluster students according to their research areas. A user query for this goal could be "`CLUSTER` *Student* `WITH` *Group.area*".

Such a user guidance shows her preference in clustering. However, this guidance should not be treated as the specification of class label attribute as in classification or semi-supervised clustering for the following reasons. First, a student may belong to multiple groups or may not be in any group if the student is new or prefers to work alone. Second, "group" may represent only one factor of "research area". There are many other important factors, such as research publications, advisors, projects, and courses_taken. Third, the guidance may serve only as a *loose hint* to the system since a nonexpert user may only have partial knowledge about the data, and thus may not even select the most important attributes in the query. Nevertheless, such a guidance plays a key role in our system since it will help the analyzer find other strongly correlated attributes across multiple relations.

The crucial challenge in cross-relational clustering is how to select pertinent features for clustering in the huge feature space. In a database with nontrivial schema, there are usually a large number of candidate features. Since only a small number of them are pertinent to the clustering goal, a powerful technique for feature search is needed to select good features for clustering. There have been many approaches

for feature selection [4, 8, 18]. But they are mainly designed for classification or trend analysis instead of clustering. For example, two features with very different trends may actually cluster objects in similar ways (details in Section 2).

To successfully perform user-guided clustering in multiple relations, CROSSCLUS developed two new techniques: (1) effective feature selection according to how features cluster tuples, and (2) efficient search for pertinent features across interconnected relations.

In cross-relational clustering each feature clusters target tuples by indicating similarities between them. Thus we introduce *similarity vector*, a new notion for representing a feature. For a feature f, its similarity vector $\mathbf{V}^f$ contains the similarity between each pair of target tuples indicated by f. $\mathbf{V}^f$ is a vector of N^2 dimensions if there are N target tuples. The similarity vector of a feature indicates how it clusters target tuples, and the similarity between two such vectors indicates whether they cluster tuples in a similar way. It is a good measure for feature selection because it captures the essential information for clustering and provides a coherent way to compare different types of features (categorical and numerical). However, it is very expensive to materialize the similarity vectors and compute feature similarity based on them. Thus we propose new methods that compute similarity between different types of features in linear time, without materializing the similarity vectors.

With a good measure for selecting features, CROSSCLUS needs to search for pertinent features across many interconnected relations. Because of the huge feature space, it is impossible to perform exhaustive search, and heuristic algorithms must be used. Two principles are followed by CROSSCLUS in feature search: (1) start from the user-specified attribute and gradually expand the search scope to other relations, and (2) the search must be confined in promising directions to avoid fruitless search. CROSSCLUS finally selects a set of highly pertinent and non-redundant features for clustering. The k-medoids algorithm is selected for clustering target tuples due to its high scalability and applicability in non-Euclidean space.

Our experiments on both real and synthetic data sets show that CROSSCLUS successfully identifies pertinent features for each clustering task, which validates the effectiveness of our approach for searching and selecting features. It also shows the high efficiency and scalability of CROSSCLUS even for databases with complex schemas.

The remaining of the paper is organized as follows. The problem definition is introduced in Section 3. Section 4 describes the approach for feature search, and Section 5 presents the approach for clustering tuples. Experimental results are presented in Section 6. We discuss the future extensions in Section 7 and conclude the study in Section 8.

2. RELATED WORK

Clustering has been extensively studied for decades in different disciplines including statistics, pattern recognition, database, and data mining. There are mainly two categories of clustering approaches: (1) probability-based approaches [3], and (2) distance-based approaches [16, 11]. Many other approaches have been proposed recently that focus on the efficiency and scalability issues (e.g., [19]).

Traditional clustering approaches may not generate good clusters in high dimensional space. Subspace clustering [1, 2, 6] was proposed to address this problem, in which each

cluster is defined based on a subset of dimensions. However, when there are a large number of features with very different semantic meanings, subspace clustering may generate many clusters with different semantic meanings, and only a small set of them fit the user's goal. In contrast, CROSSCLUS explores the concept of user-guided clustering: It selects a set of pertinent features according to user guidance and then performs clustering with them, which leads to more meaningful clustering results.

Clustering in multi-relational environments has been studied in [7, 14, 15]. When computing similarity between two tuples, they consider not only the attributes of the two tuples, but also those of the tuples related to them (joinable via a few joins) [5, 7]. However, these approaches suffer from poor efficiency and scalability, because in a large database an object is often joinable with thousands of tuples via a few joins, which makes it very expensive to compute similarity between each pair of objects. Moreover, they use all features indiscriminately and may not generate good clustering results that fit the user's goal.

Semi-supervised clustering [12, 13, 21] also performs clustering under user's guidance. The user guidance is provided either as a set of similar (or dissimilar) pairs of instances [12, 21] or by relevance feedback [13]. Such guidance is either used to reassign tuples [12] to clusters or to wrap the space to generate better models [21]. However, it is burdensome for each user to provide such training data, and it is often difficult to judge whether two tuples belong to same cluster since all relevant information in many relations needs to be shown to user. In contrast, CROSSCLUS allows the user to express the clustering goal with a simple query containing a pertinent attribute, and will search for more pertinent features across multiple relations.

Selecting the right features is crucial for cross-relational clustering. Although feature selection has been extensively studied for both supervised learning [8] and unsupervised learning [4, 18], the existing approaches may not be appropriate for cross-relational clustering. The widely used criteria for selecting numerical features include Pearson Correlation [9] and least square error [18]. However, such criteria focus more on trends of features, instead on how they cluster tuples. For example, consider the 80 tuples shown in Figure 2, with their values on two features. According to the above criteria, the two features have correlation of zero. If we cluster the tuples into 8 clusters according to each feature, feature 1 will create clusters $\{t_1, \ldots, t_{10}\}, \ldots,$ $\{t_{71}, \ldots, t_{80}\}$, feature 2 will create $\{t_1, \ldots, t_5, t_{76}, \ldots, t_{80}\}$, $\ldots, \{t_{36}, \ldots, t_{40}, t_{41}, \ldots, t_{45}\}$. If two tuples belong to same cluster according to feature 1, they have half chance to belong to same cluster according to feature 2, and vice versa. One can see that feature 1 and feature 2 actually cluster tuples in rather similar ways, although they are "uncorrelated" according to some correlation measures. For categorical features, the most popular criteria include information gain [17] or mutual information [8]. However, it is difficult to define information gain or mutual information for multi-relational features, because a tuple has multiple values on a feature.

3. PROBLEM DEFINITION

3.1 User Queries

CROSSCLUS accepts user queries that contain a *target relation, and one or a small set of pertinent attribute(s)*. Con-
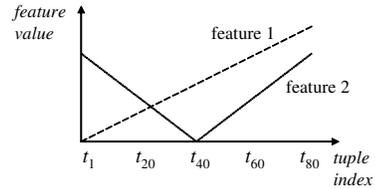

Figure 2: Two "uncorrelated" features

sider the query "CLUSTER *Student* WITH *Group.area*" in Example 1 that aims to cluster students according to their research areas. *Student* is the target relation R_t, and *Group.area* is the pertinent attribute. Since the pertinent attribute provides crucial but very limited information for clustering, CROSSCLUS searches for other pertinent features, and groups the target tuples based on those features. Different relations are linked by joins. As in some other systems on relational databases (such as DISCOVER [10]), only joins between keys and foreign-keys are considered by CROSSCLUS. Other joins are ignored because they do not represent strong semantic relationships between objects. Due to the nature of relational data, CROSSCLUS uses a new definition for objects and features, as described below.

3.2 Multi-Relational Features

A multi-relational feature f is defined by a join path $R_t \bowtie R_1 \bowtie \cdots \bowtie R_k$, an attribute $R_k.A$ of R_k, and possibly an aggregation operator (*e.g.*, average, count, max). f is formally represented by $[f.joinpath, f.attr, f.aggr]$. A feature f is either a *categorical feature* or a *numerical one*, depending on whether $R_k.A$ is categorical or numerical. For a target tuple t, we use $f(t)$ to represent t's value on f. We use $T_f(t)$ to represent the set of tuples in R_k that are joinable with t, via $f.joinpath$.
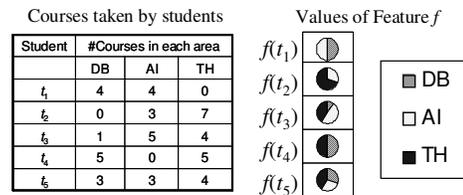

Figure 3: A feature of areas of courses taken by students

If f is a categorical feature, $f(t)$ represents the distribution of values among $T_f(t)$. Suppose $f.attr$ has l values $v_1, \ldots, v_l$. $f(t)$ is an l-dimensional vector $(f(t).p_1, \ldots, f(t).p_l)$, with $f(t).p_i$ representing the proportion of tuples in $T_f(t)$ having value v_i. For example, suppose *Student* is the target relation in the CS Dept database. Consider a feature $f = [Student \bowtie Register \bowtie OpenCourse \bowtie Course, area,$ null[1]] (where *area* represents the areas of the courses taken by a student). An example is shown in Figure 3. A student t_1 takes four courses in database and four in AI, thus $f(t_1) =$ (database:0.5, AI:0.5). In general, $f(t)$ represents t's relationship with each value of $f.attr$. The higher $f(t).p_i$ is, the stronger the relationship between t and v_i is. This vector is usually sparse and only non-zero values are stored.
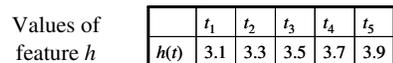
Values of		t_1	t_2	t_3	t_4	t_5
feature h	$h(t)$	3.1	3.3	3.5	3.7	3.9

Figure 4: A feature of students' average grades

If f is numerical, then f has a certain aggregation operator (average, count, max, ...), and $f(t)$ is the aggregated value of tuples in the set $T_f(t)$, as shown in Figure 4.

---

[1] which means no aggregation operator is used in this feature.

When searching for pertinent features during clustering, both categorical and numerical features need to be represented in a coherent way, so that they can be compared and pertinent features can be found. In clustering, the most important information carried by a feature f is how f clusters tuples, which is conveyed by the similarity between tuples indicated by f. Thus in CROSSCLUS a feature is mainly represented by the *similarity vector*, as defined below.

Definition 1. (**Similarity Vector**) Suppose there are N target tuples $t_1, \ldots, t_N$. The similarity vector of feature f, $\mathbf{V}^f$, is an N^2 dimensional vector. $\mathbf{V}^f_{iN+j}$ represents the similarity between t_i and t_j indicated by f.

The similarity vector of feature f in Figure 3 is shown in Figure 5. The two horizontal axes are tuple indices, and the vertical axis is similarity. The similarity vector of a feature represents how it clusters target tuples. It contains the most important information for feature selection in clustering, and is used for identifying pertinent features according to user guidance.
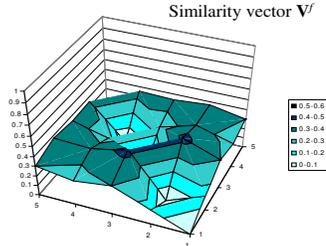
Figure 5: **Similarity vector of a feature**

4. FINDING PERTINENT FEATURES

In this section we first describe the similarity measure between features, then introduce our approach for feature searching based on user's guidance.

4.1 Similarity between Features

Given the user guidance, CROSSCLUS selects pertinent features based on their relationships to the feature specified by user. To achieve this goal, we should compare features to find pertinent features but avoid redundant ones. Features should be compared based on how they cluster target tuples. If two features cluster tuples in a similar way, they should be considered to be related to each other; and vice versa.

Similarity between two features are measured based on their similarity vectors, which represent the ways they cluster tuples. If two features cluster tuples very differently, their similarity vectors are very different, and their similarity is low. If they cluster tuples in almost the same way, their similarity is very high, which indicates that they contain redundant information. In general, the similarity between features gives us good hints about the relationship between them. Moreover, although different types of features contain different formats of information that is hard to compare directly, we can define similarity between them in a coherent way as long as we know their similarity vectors.

4.1.1 Tuple Similarity for Categorical Features

In CROSSCLUS a categorical feature does not simply partition the target tuples into disjoint groups. Instead, a target tuple may have multiple values on a feature, as defined in Section 3.2. Consider a categorical feature f that has

l values $v_1, \ldots, v_l$. A target tuple t's value on f is a vector $(f(t).p_1, \ldots, f(t).p_l)$. $f(t).p_i$ is the proportion of tuples joinable with t (via $f.joinpath$) that have value v_i on $f.attr$, which represents the probability that t is related to v_i.

The similarity between two tuples t_1 and t_2 w.r.t. f is defined as the probability that t_i and t_j is related to the same value on f. An example is shown in Figure 6.

Definition 2. The similarity between two tuples t_1 and t_2 w.r.t. f is defined as

$$ sim_f(t_1, t_2) = \sum_{k=1}^{l} f(t_1).p_k \cdot f(t_2).p_k \qquad (1) $$
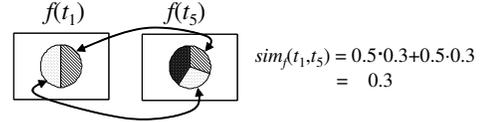
Figure 6: **Tuple similarity of categorical features**

There are other measures for defining similarity between two probabilistic distributions, such as KL-distance (or relative entropy), or cosine similarity between the two distribution vectors. However, such measures focus on the similarity of the two distributions, instead of the properties of the two objects. For example, suppose we are clustering students according to their research areas. If students t_1 and t_2 both have probability of 0.5 to take courses in DB area and 0.5 in AI area, then their similarity is 1 according to KL-distance or cosine similarity. But actually it is quite possible that one is in DB area and the other is in AI area. If t_1 has probability 1.0 in DB, and t_2 has 0.7 in DB and 0.3 in AI, then it is more likely they are in the same area although their similarity is not high according to KL-distance or cosine similarity. We choose a simple but appropriate measure—the probability that t_1 and t_2 take courses in same area, which better represents the relationship between tuples w.r.t. a feature.

4.1.2 Tuple Similarity for Numerical Features

Different numerical features have very different ranges and variances, and we need to normalize them so that they have equal influences in clustering. For a feature h, similarity between two tuples t_1 and t_2 w.r.t. h is defined as follows.

Definition 3. Suppose σ_h is the standard deviation of tuple values of numerical feature h. The similarity between two tuples t_1 and t_2 w.r.t. h is defined as

$$ sim_h(t_1, t_2) = 1 - \frac{|h(t_1) - h(t_2)|}{\sigma_h}, \ if \ |h(t_1) - h(t_2)| < \sigma_h; $$
$$ 0, otherwise. \qquad (2) $$

That is, we first use *Z score* to normalize values of h, so that they have mean zero and variance one. Then the similarity between tuples are calculated based on the Z scores. If the difference between two values is greater than σ_h, their similarity is considered to be zero.

4.1.3 Similarity between Features

When clustering tuples, a feature is used to indicate the relationship between each pair of tuples. CROSSCLUS represents a feature f by the similarities between tuples indicated by f. Suppose there are N target tuples $t_1, \ldots, t_N$. f is represented by its similarity vector $\mathbf{V}^f$ (in which $\mathbf{V}^f_{iN+j} = sim_f(t_i, t_j)$).

When measuring the similarity between two features f and g, we compare their similarity vectors $\mathbf{V}^f$ and $\mathbf{V}^g$, which

represent how they cluster tuples. The similarity between two features is defined as the cosine similarity between their similarity vectors. This definition can be applied on both categorical and numerical features.

Definition 4. The similarity between two features f and g is defined as[2]

$$sim(f, g) = \frac{\mathbf{V}^f \cdot \mathbf{V}^g}{|\mathbf{V}^f| \cdot |\mathbf{V}^g|} \qquad (3)$$

in which $|\mathbf{V}^f| = \sqrt{\mathbf{V}^f \cdot \mathbf{V}^f}$.

$sim(f, g) = 1$ if and only if $\mathbf{V}^f$ and $\mathbf{V}^g$ differ by a constant ratio, and $sim(f, g)$ is small when most pairs of tuples that are similar according to f are dissimilar according to g. Figure 7 shows the values of two features $f = [Student \bowtie Register \bowtie OpenCourse \bowtie Course, area, null]$ (areas of courses taken by each student) and $g = [Student \bowtie WorkIn, group, null]$ (research group of each student). Their similarity vectors $\mathbf{V}^f$ and $\mathbf{V}^g$ are shown in Figure 8. To compute $sim(f, g)$, one needs to compute the inner product of $\mathbf{V}^f$ and $\mathbf{V}^g$, as shown in Figure 8.

Feature f				Feature g		
t ID	DB	AI	TH	t ID	Info sys	Cog sci	Theory
1	0.5	0.5	0	1	1	0	0
2	0	0.3	0.7	2	0	0	1
3	0.1	0.5	0.4	3	0	0.5	0.5
4	0.5	0	0.5	4	0.5	0	0.5
5	0.3	0.3	0.4	5	0.5	0.5	0

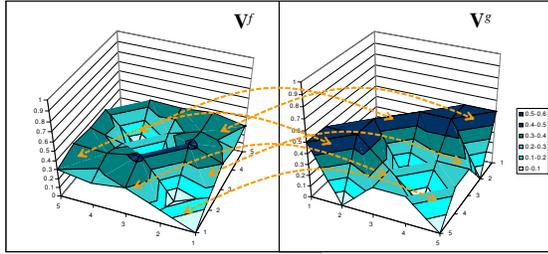**Figure 7: The values on two features f and g**

Figure 8: Inner product of $\mathbf{V}^f$ and $\mathbf{V}^g$

4.1.4 Efficient Computation of Feature Similarity

The similarity between features is expensive to compute, if directly following the definition. We cannot even afford storing N^2 dimensional vectors for many applications. An efficient algorithm is needed to compute $sim(f, g)$ without materializing $\mathbf{V}^f$ and $\mathbf{V}^g$.

We first describe the approach for computing similarities between categorical features. The main idea is to convert the hard problem of computing inner product of two similarity vectors of N^2 dimensions, into an easier problem of computing similarities between feature values, which can be solved in linear time. Figure 9 shows an example, in which seven tuples are connected to different feature values. Similarities between tuples are defined according to their relationships with the feature values, as in Definition 2. The similarities between feature values can be defined similarly according to their relationships with tuples.

Definition 5. The similarity between value v_k of feature f and value v_q of feature g is defined as

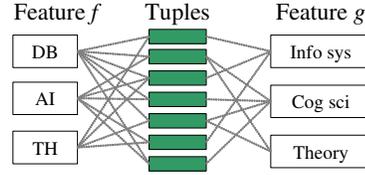$$sim(f.v_k, g.v_q) = \sum_{i=1}^{N} f(t_i).p_k \cdot g(t_i).p_q \qquad (4)$$

Figure 9: An example of tuples and feature values

The definitions of tuple similarity and feature value similarity are similar and symmetric to each other. Because of the symmetric definitions, we can convert the inner product of $\mathbf{V}^f$ and $\mathbf{V}^g$ from summation of tuple similarities into that of feature value similarities.

LEMMA 1.

$$
\begin{aligned}
\mathbf{V}^f \cdot \mathbf{V}^g &= \sum_{i=1}^{N} \sum_{j=1}^{N} sim_f(t_i, t_j) \cdot sim_g(t_i, t_j) \\
&= \sum_{k=1}^{l} \sum_{q=1}^{m} sim(f.v_k, g.v_q)^2 \qquad (5)
\end{aligned}
$$

With Lemma 1, to compute the similarity between f and g, we only need to compute $sim(f.v_k, g.v_q)$ for each value v_k of f and v_q of g. We can compute them by scanning the tuples just once. For each tuple t, we get $f(t)$ and $g(t)$. For each value v_k so that $f(t).v_k > 0$, and each v_q so that $g(t).v_q > 0$, we update $sim(f.v_k, g.v_q)$ by adding $f(t).v_k \cdot g(t).v_q$. In this way $sim(f.v_k, g.v_q)$ $(1 \le k \le l, 1 \le q \le m)$ can be computed in one scan. This requires an entry in memory for each $sim(f.v_k, g.v_q)$. In reality most categorical features have only several or tens of values. If f has too many values, CROSSCLUS will make an indexed scan on f (scanning tuples having each value of f), and it only needs to maintain an entry for each value of g. In general, $\mathbf{V}^f \cdot \mathbf{V}^g$ can be computed in linear time, with very limited extra space.

A different approach is needed for computing similarities involving numerical features, because of the different definitions of tuple similarities. Consider the numerical feature h in Figure 4, whose similarity vector $\mathbf{V}^h$ is shown in Figure 10. For simplicity we assume that $t_1, \dots, t_N$ are sorted according to h.
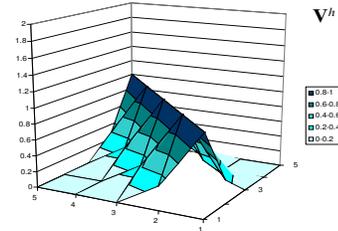

Figure 10: Similarity vector of feature h

As mentioned above, it is expensive to directly compute the inner product of two N^2 dimensional vectors $\mathbf{V}^h$ and $\mathbf{V}^f$. Again we try to convert this problem into a problem that can be solved in linear time. From Section 4.1.2 one can see that only tuples whose values differ by at most σ (standard deviation) have non-zero similarity with each other. When we scan tuples in order of h, only a subset of tuples need to be considered when scanning each tuple. The main idea of our algorithm is to decompose $\mathbf{V}^h \cdot \mathbf{V}^f$ into two parts, so that one part only depends on each tuple, and the other part

contains some statistics of the previously scanned tuples, which can be maintained incrementally. In this way $\mathbf{V}^h \cdot \mathbf{V}^f$ can be computed by one scan.

Because of the symmetric definition of tuple similarity, $sim(t_i, t_j) = sim(t_j, t_i)$ for any feature. Let $\eta(i)$ represent the minimum index j so that $h(t_i) - h(t_j) \le \sigma$. As i increases when scanning tuples, $\eta(i)$ either increases or stays the same. Thus we can find all tuples with $h(t_i) - h(t_j) \le \sigma$ by maintaining two pointers on t_i and $t_{\eta(i)}$. We have

$$\mathbf{V}^h \cdot \mathbf{V}^f = 2 \sum_{i=1}^{N} \sum_{j=\eta(i)}^{i-1} sim_h(t_i, t_j) \cdot sim_f(t_i, t_j), \quad (6)$$

which can be efficiently computed with Lemma 2.

LEMMA 2. $\mathbf{V}^h \cdot \mathbf{V}^f$ can be computed as follows

$$V^h \cdot V^f = 2 \sum_{i=1}^{N} \sum_{j=\eta(i)}^{i-1} \left[1 - \left(h(t_i) - h(t_j)\right)\right] \left[\sum_{k=1}^{l} f(t_i).p_k \cdot f(t_j).p_k \right]$$

$$= 2 \sum_{i=1}^{N} \sum_{k=1}^{l} \boxed{f(t_i).p_k(1 - h(t_i))} \boxed{\left(\sum_{j=\eta(i)}^{i-1} f(t_j).p_k \right)} + 2 \sum_{i=1}^{N} \sum_{k=1}^{l} \boxed{f(t_i).p_k} \boxed{\left(\sum_{j=\eta(i)}^{i-1} h(t_j) \cdot f(t_j).p_k \right)}$$

Only depend on t_i	Statistics of t_j with $j<i$

To compute $\mathbf{V}^h \cdot \mathbf{V}^f$, we scan all tuples in order of h. When scanning each tuple t_i, we get $f(t_i).p_k \cdot (1 - h(t_i))$ and $f(t_i).p_k$ for each value v_k of f. We also dynamically maintain the two statistics $\sum_{j=\eta(i)}^{i-1} f(t_j).p_k$ and $\sum_{j=\eta(i)}^{i-1} h(t_j) \cdot f(t_j).p_k$.[3] In this way we can compute the inner product according to Lemma 2. One can see that $\mathbf{V}^h \cdot \mathbf{V}^f$ can be computed by one scan on the tuples, also with very limited extra space.

The similarity between two numerical features h and g is computed in a different way. Suppose tuples are sorted according to their values on h. According to Definition 3, when scanning tuple t^*, only tuples t with $|h(t) - h(t^*)| < \sigma_h, |g(t) - g(t^*)| < \sigma_g$ need to be considered, which is usually a small subset of tuples. Therefore, we compute $\mathbf{V}^h \cdot \mathbf{V}^g$ in the following way.
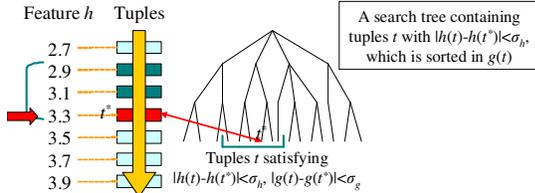

Figure 11: Computing similarity between h and g

As shown in Figure 11, we scan tuples in order of h. A search tree is dynamically maintained, which contains the indices of all tuples t with $h(t^*) - h(t) < \sigma_h$. This tree is indexed by tuple values on g. When scanning a tuple t^*, it is inserted into the tree, and all tuples t with $h(t^*) - h(t) < \sigma_h, |g(t) - g(t^*)| < \sigma_g$ can be easily found in the tree by an indexed search. $\mathbf{V}^h \cdot \mathbf{V}^g$ can be updated with the values of these tuples according to Definitions 3 and 4. Because the number of tuples satisfying $h(t^*) - h(t) < \sigma_h, |g(t) - g(t^*)| < \sigma_g$ is usually quite small, $\mathbf{V}^h \cdot \mathbf{V}^g$ can be efficiently computed by one scan on the tuples.

Although similarity between two features can usually be computed in linear time, it is still expensive to compute

---

[3]This can be done by adding the corresponding part of newly scanned tuple and removing the corresponding part of those tuples going out of the range.

similarities between many pairs of features when the number of target tuples is large. We observe that $sim(f, g)$ is actually the cosine of the angle between two vectors $\mathbf{V}^f$ and $\mathbf{V}^g$, which lie in the Euclidean space. Thus we can utilize *triangular inequality* to further improve efficiency.

According to the law of cosines, the distance between $\mathbf{V}^f$ and $\mathbf{V}^g$ can be computed by

$$|\mathbf{V}^f - \mathbf{V}^g| = \sqrt{|\mathbf{V}^f|^2 + |\mathbf{V}^g|^2 - 2|\mathbf{V}^f||\mathbf{V}^g|sim(f, g)} \quad (7)$$

For any three features f, g and h, since $\mathbf{V}^f$, $\mathbf{V}^g$ and $\mathbf{V}^h$ are vectors in Euclidean space, according to triangular inequality,

$$|\mathbf{V}^f - \mathbf{V}^g| \ge \left| |\mathbf{V}^f - \mathbf{V}^h| - |\mathbf{V}^h - \mathbf{V}^g| \right|,$$
$$|\mathbf{V}^f - \mathbf{V}^g| \le |\mathbf{V}^f - \mathbf{V}^h| + |\mathbf{V}^h - \mathbf{V}^g| \quad (8)$$

With triangular inequality, before computing the similarity between features f and g, a range can be determined for $sim(f, g)$ using their similarities to other features. This helps save some computation in searching for features similar to a certain feature.

4.2 Efficient Generation of Feature Values

When searching for pertinent features, CROSSCLUS needs to generate the values of target tuples on many different features. It is often very expensive to join all relations along the join path of a feature. Thus we need an efficient approach for generating feature values. CROSSCLUS uses a technique called *Tuple ID Propagation* [20], which propagates the IDs of target tuples to other relations, in order to find tuples joinable with each target tuple.

Tuple ID propagation is a way to virtually join tuples in target relation R_t with tuples in another relation R_k via a certain join path $p = R_t \bowtie R_1 \bowtie \cdots \bowtie R_k$. Its main idea is to propagate the IDs of target tuples from R_t to the relations along the join path one by one, so that for each relation R_i on the join path, each tuple t' of R_i is associated with the IDs of all target tuples joinable with t'. After propagating IDs along a join path, the IDs are stored on each relation on the path and can be further propagated to other relations.

Suppose a feature f is defined by a join path $p = R_t \bowtie R_1 \bowtie \cdots \bowtie R_k$, and an attribute $R_k.A$. To generate the values of each target tuple on f, we first propagate IDs along path p. (If we have already propagated IDs along a prefix of p, those IDs can be used.) The IDs associated with every tuple t' in R_k represent target tuples joinable with t'. From this information we can easily compute the tuples in R_k joinable with each target tuple, then compute the proportion of tuples having each value on $R_k.A$ among them, or any aggregation of these tuples. In this way the value on f for each target tuple can be computed.

4.3 Searching for Pertinent Features

Given a clustering task with user guidance, CROSSCLUS searches for pertinent features across multiple relations. There are two major challenges in feature search. (1) *The number of possible features is huge in multi-relational environment.* The target relation R_t can usually join with each relation R via many different join paths, and each attribute in R can be used as a feature. It is impossible to perform any kind of exhaustive search in this huge feature space. (2) *Among the huge number of features, some are pertinent to the user query (e.g., a student's advisor is related to her research area), while many others are irrelevant (e.g., a student's classmates' personal information).* It is a challenging

task to identify pertinent features while avoiding aimless search in irrelevant regions in the feature space.

To overcome the above challenges, CROSSCLUS must confine the search process in promising directions. It adopts a heuristic approach, which starts search from the user-specified feature, and then repeatedly searches for useful features in the neighborhood of existing features. In this way it gradually expands the search scope to related relations, but will not go deep in random directions.

The large number of possible features cover different aspects of information. For example, some features are about a student's research area, such as her research group or conferences of publications. Some others are about her academic performance, such as her grade or publications. Our experiments show that features in the same aspect usually have high similarities between each other, while features in different aspects usually have low similarities. A user is usually interested in clustering tuples based on a certain aspect of information, and indicates this by a query. An initial feature is created for the query, and more pertinent features can be found by computing similarities between different features in different relations.

We also consider three properties of a feature f: (a) *fan out*: the average number of tuples joinable with each target tuple via $f.joinpath$, (b) *coverage*: proportion of target tuples that have values on f, and (c) the length of $f.joinpath$. A feature is not considered if it has too high fan out, too low coverage, or too long join path.
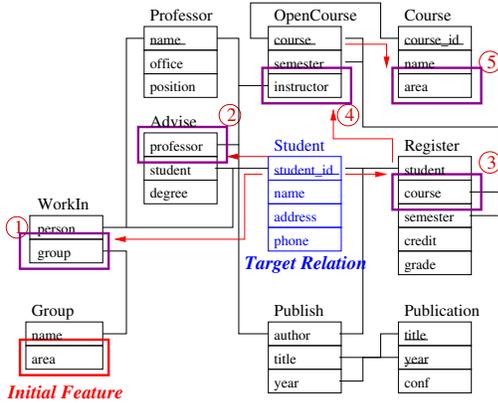

Figure 12: Feature Search in CROSSCLUS

Example 2. Suppose a user wants to cluster students according to research areas, and thus query with "CLUSTER *Student* WITH *Group.area*".

To create the initial feature for this query, CROSSCLUS searches for the shortest join path from the target relation *Student* to relation *Group*, and creates a feature f using this path. If f is not qualified (*e.g.*, f has too low coverage), CROSSCLUS searches for the next shortest join path, and so on. As in Figure 1, an initial pertinent feature [*Student* ⋈ *WorkIn* ⋈ *Group, area*, null] is created for this query.

Then CROSSCLUS searches for pertinent features among multiple relations. It considers the relational schema as a graph, with relations being nodes and joins being edges. CROSSCLUS starts from the node of the initial feature, and explores the surrounding nodes of pertinent features. It keeps adding in new pertinent features, and expanding the search scope to all neighbor nodes of those containing pertinent features. In this way it avoids aimless search in the huge feature space by confining the search in promising di-

ALGORITHM 1. **Feature Searching**

Input: *A relational database D, a set of initial features F.*

Output: *A set of pertinent features.*

candidate feature set $C \leftarrow \emptyset$
for each $f \in F$
   add features with join path $f.joinpath$ to C
   for each relation R that can be appended to $f.joinpath$
      $p \leftarrow f.joinpath + R$
      add features with join path p to C
remove features in F from C
for each $f \in C$
   find L features in F most similar to f
repeat
   $f \leftarrow$ feature with maximum weight in C
   add f to F
   remove f from C
   for each relation R that can be appended to $f.joinpath$
      $p \leftarrow f.joinpath + R$
      add features with join path p to C
   for each $f \in C$
      find L features in F most similar to f
until $(1 - \varepsilon)$ target tuples are covered by at least H features
return F

Figure 13: Algorithm Feature Searching

rections. On the other hand, this strategy allows it to gradually expand the search scope to many relations, so that most pertinent features can be found and sufficient information can be provided for clustering target tuples.

We simulate the procedure of feature searching, as shown in Figure 12. At first CROSSCLUS considers features in the following relations: *Advise*, *Publish*, *Register*, *WorkIn*, and *Group*. Suppose the best feature [*Student* ⋈ *Advise*, *professor*, null] (advisor of a student), which brings *Professor* relation into consideration in further search. CROSSCLUS will search for more pertinent features, until most tuples are sufficiently covered.

Each feature is given a weight between 0 and 1, which is determined by its relationship to other pertinent features. The weight of the initial pertinent feature is 1. For each newly added feature f, f should have high weight if it is very similar to some highly pertinent features, and vice versa. Therefore, f's weight is based on the L features that f is most similar to. Suppose f is most similar to $f'_1, \ldots, f'_L$.

$$f.weight = \sum_{i=1}^{L} f'_i.weight \cdot sim(f, f'_i) \qquad (9)$$

Figure 13 shows the algorithm of feature search. At first the initial feature is the only pertinent feature. At each step, CROSSCLUS gets all candidate features and evaluates each candidate feature f_c by the L pertinent features most similar to f_c. *Triangular inequality* is used to infer the ranges of similarities between f_c and some pertinent features, which may exclude the possibility that f_c is similar to some features. The candidate feature with the highest weight, f_c^*, is added to the set of pertinent features at each step, and new candidate features related to f_c^* are then added. We say a tuple is *covered* by a feature if it has non-empty value on this feature. The algorithm stops when most target tuples have been covered by at least H features.

A set of pertinent features are generated by feature search, but some of them may contain redundant information. CROSSCLUS further selects a set of non-redundant features, so that the similarity between any two features is no greater than sim_{max}. This can be easily done because similarity between each pair of pertinent features has been computed.

5. CLUSTERING TUPLES

Given a group of features $f_1, \ldots, f_h$, CROSSCLUS groups the target tuples into clusters that contain tuples similar to each other. We will first introduce the similarity measure between tuples, then introduce the clustering algorithm.

5.1 Similarity Measure

The similarity between tuples w.r.t. each feature has been defined in Section 4.1.1 and 4.1.2, and such similarity has been used for selecting features for clustering. The same definition is used for measuring tuple similarity in clustering.

Consider two tuples t_1 and t_2, which have values on h features $f_1, \ldots, f_h$. Each feature f_i has weight $f_i.weight$. The similarity between t_1 and t_2 is defined as the weighted sum of their similarity on each feature.

$$sim(t_1, t_2) = \sum_{i=1}^{h} sim_{f_i}(t_1, t_2) \cdot f_i.weight \qquad (10)$$

5.2 Clustering Algorithm

With the potentially large number of target tuples, an efficient and scalable clustering algorithm should be chosen. Since the features do not form a Euclidean space, we choose k-medoids [11, 19], a most widely used clustering algorithm that only requires distance measure between tuples. We choose CLARANS [19], an efficient k-medoids algorithm for CROSSCLUS. The main idea of CLARANS is to consider the whole space of all possible clusterings as a graph, and use randomized search to find good clustering in this graph. It starts with k initial medoids and constructs k clusters. In each step an existing medoid is replaced by a new medoid that is randomly picked. If the replacement leads to better clustering, the new medoid is kept. This procedure is repeated until the clusters remain stable.

Finally, CROSSCLUS provides the clustering results to the user, together with information about each feature. From the features' join paths, attributes, and aggregation operators, the user will know the meaning of this clustering, which helps her understand the clustering results.

6. EXPERIMENTAL RESULTS

We report comprehensive experiments on both synthetic and real databases. All tests were done on a 2.4GHz Pentium-4 PC with 1GB memory, running Windows XP. The following parameters are used in CROSSCLUS: $H = 10$, $L = 3$ and $\varepsilon = 0.05$ in Algorithm 1. $sim_{max} = 0.95$. $k = 20$ in k-medoids algorithm. We compare CROSSCLUS with three other approaches: *Baseline*, PROCLUS, and *RDBC*.

Baseline clusters data by the initial feature only, and is very efficient because no feature search is performed.

PROCLUS [2] is the state-of-the-art subspace clustering approach that works on single tables. To apply PROCLUS, we first convert relational data into single tables. For each clustering query, we perform a breadth-first search of depth three from the initial feature in the schema graph, and use every attribute in every relation as a feature (except features with too high fan out or too low coverage). This generates a table with 50 to 100 features for each query.

RDBC [14, 15] is a recent clustering approach for first-order representations. It uses the similarity measure in [5], which considers the related objects when computing the similarity between two objects. *RDBC* can utilize different clustering algorithms, and we choose the same CLARANS al-

gorithm [19] as in CROSSCLUS. When measuring similarities between two objects, only directly related objects (objects of depth 0) are used, because it becomes too slow with larger depth.

All four algorithms use the same philosophy in clustering (k-medoids). PROCLUS and *RDBC* are slightly modified to utilize the user guidance. In PROCLUS it is enforced that the initial feature is included in the feature set of every cluster. In *RDBC*, the objects related to each target tuple t via the initial feature are added to the set of related objects of t.

6.1 Clustering Effectiveness

In this experiment we test whether CROSSCLUS can produce reasonable clusters with user queries. Two real databases are used. The first is the CS Dept database[4] (as shown in Figure 1), which is collected from web sources of Dept. of CS, UIUC. It has 10 relations and 4505 tuples. All relations contain real data, except the *Register* relation which is randomly generated. The second is the Movie database from UCI repository, whose schema is shown in Figure 14. This database has 7 relations and 61660 tuples.
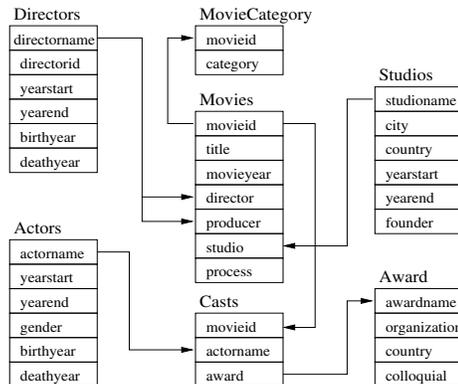

Figure 14: Schema of the Movies database

A few clustering tasks are chosen for each database, and a clustering query is given for each task. To test the accuracy of a clustering algorithm, we need a *standard clustering* for each task, which represents the correct partitioning of the target tuples. The standard clustering for a clustering task is created by a *standard feature set*, which is the right set of pertinent features for this task. The following principle is used to identify the standard feature set:

For a clustering task, the standard feature set is the set of features that contain information directly related to the task.

For example, if the task is to cluster students according to research area, the standard feature set consists of student's research groups, group areas, advisor, and conferences of publications. If the task is to cluster actors according to their eras, the standard feature set consists of the actor's birth year, death year, start and end year of career, and years of movies she participated. Indirectly related information, such as birth year of directors of movies she participated, is not considered.

The similarity between two clusterings C and C' is measured by how much the clusters in C and C' overlap with each other. Suppose C has n clusters $c_1, \ldots, c_n$, and C' has n' clusters $c'_1, \ldots, c'_{n'}$. The degree of C subsuming C' is defined as,

---

$$deg(C \subset C') = \frac{\sum_{i=1}^{n} \max_{1 \le j \le n'}(|c_i \cap c'_j|)}{\sum_{i=1}^{n} |c_i|} \qquad (11)$$

$deg(C \subset C')$ represents the average proportion that each cluster in C is contained in some cluster in C'. The similarity between C and C' is defined as the average degree that one of them subsumes another.

$$sim(C, C') = \frac{deg(C \subset C') + deg(C' \subset C)}{2} \qquad (12)$$

$sim(C, C') = 1$ if and only if C and C' are identical.

For each clustering task, the standard clustering is generated by applying the k-medoids algorithm on the standard feature set. The four algorithms are tested on this task. *The clustering result of each algorithm is compared with the standard clustering, and their similarity is used as the accuracy of this algorithm.* Each experiment is done 10 times and the average is reported.

Task 1: clustering professors according to research areas in CS Dept database. The query is "CLUSTER *Professor* WITH *Group.area*". The standard feature set is
1. [$Professor \bowtie WorkIn \bowtie Group$, *area*, null],
2. [$Professor \bowtie WorkIn$, *group*, null],
3. [$Professor \bowtie OpenCourse \bowtie Course$, *area*, null],
4. [$Professor \bowtie Publish \bowtie Publication$, *conf*, null].

The feature set selected by CROSSCLUS contains five features, including the first three standard features, plus [$Professor \bowtie OpenCourse$, *course*, null] and [$Professor \bowtie OpenCourse \bowtie Course \bowtie OpenCourse$, *instructor*, null]. The accuracies and runtime (in seconds) are shown in Table 1.

	Baseline	PROCLUS	RDBC	CROSSCLUS
Accuracy	0.696	0.668	0.532	0.824
Runtime	0.14	1.81	0.73	0.71

Table 1: Experiment 1 on CS Dept database

There are only a small number of distinct values for *Group.area*. CROSSCLUS successfully identifies clusters of professors working in different areas, some of which are not specified in *Group.area*:
Theory: J. Erickson, S. Har-Peled, L. Pitt, E. Ramos, D. Roth, M. Viswanathan.
Graphics: J. Hart, M. Garland, Y. Yu.
Database: K. Chang, A. Doan, J. Han, M. Winslett, C. Zhai.
Numerical computing: M. Heath, T. Kerkhoven, E. de Sturler.
Real-time systems: M. Caccamo, J. Hou, L. Sha.
AI: G. Dejong, M. Harandi, S. Lavalle, J. Ponce, L. Rendell.
Architecture: S. Adve, L. Kale, M. Snir, C. Zilles.
and other clusters such as operating system, programming language, etc.

Task 2: clustering students according to research areas in CS Dept database. The query is "CLUSTER *Student* WITH *Group.area*". The standard feature set is
1. [$Student \bowtie WorkIn \bowtie Group$, *area*, null],
2. [$Student \bowtie WorkIn$, *group*, null],
3. [$Student \bowtie Advise$, *professor*, null],
4. [$Student \bowtie Publish \bowtie Publication$, *conf*, null].

The feature set selected by CROSSCLUS contains the first three of the standard features. The fourth feature is not selected because of its high cardinality. Again CROSSCLUS successfully identifies clusters of students in different areas.

	Baseline	PROCLUS	RDBC	CROSSCLUS
Accuracy	0.548	0.446	0.322	0.861
Runtime	0.41	7.72	18.0	1.18

Table 2: Experiment 2 on CS Dept database

Task 3: clustering directors according to the genres and styles of their movies in Movies database. The query is "CLUSTER *Directors* WITH *Movies.category*". The movies' styles are related to their categories (genres), countries, studios, and award information. The standard feature set is
1. [$Director \bowtie Movies \bowtie MovieCategory$, *category*, null],
2. [$Director \bowtie Movies$, *studio*, null],
3. [$Director \bowtie Movies \bowtie Studios$, *country*, null],
4. [$Director \bowtie Movies \bowtie Casts$, *award*, null],

The feature set selected by CROSSCLUS includes all above features, plus *Movies.year* and *Movies.process* (b&w, color, silent, etc). The results are shown in Table 3.

	Baseline	PROCLUS	RDBC	CROSSCLUS
Accuracy	0.557	0.387	0.192	0.588
Runtime	5.53	123	341	25.6

Table 3: Experiment 1 on Movies database

In the clusters generated by CROSSCLUS, we easily find some directors of famous movies, as follows.
Sci.& Fic.: George Lucas (*Star Wars*), Montgomery Tully (*Battle Beneath the Earth, The Terrornauts*), Robert Wiemer (*Star Trek*), Robert Zemeckis (*Back to the Future 1,2*), Andy Wachowski (*Matrix1,2,3*), ...
Kid and Ani.: Hamilton Luske (*Cinderella, Peter pan*), Charles Nichols (*Tom and Jerry*), Roger Allers (*Lion King, Aladdin*), John Lasseter (*Finding Nemo, Monster Inc.*), ...
Action: James Cameron (*Terminators, Aliens, True Lies*), Brett Ratner (*Rush Hour 1,2*), Quentin Tarantino (*Kill Bill 1,2*), Tony Scott (*Top Gun, Enemy of the State, Crimson Tide*), ...
and other clusters such as romance, thrillers and musicals.

Task 4: clustering actors according to their eras in Movies database. The query is "CLUSTER *Actors* WITH *Movies.movieyear*". The standard feature set is
1. [$Actors \bowtie Casts \bowtie Movies$, *movieyear*, average],
2. [$Actors$, *yearstart*, average],
3. [$Actors$, *yearend*, average],
4. [$Actors$, *birthyear*, average],
5. [$Actors$, *deathyear*, average],

The feature set selected by CROSSCLUS contains feature 1,2,5, and another feature [$Actors \bowtie Casts \bowtie Movies \bowtie Directors$, *yearend*, average], which also indicates an actor's era. Some standard features are missed because they are considered redundant. The results are shown in Table 4.[5]

	Baseline	PROCLUS	RDBC	CROSSCLUS
Accuracy	0.321	0.413	0.380	0.520
Runtime	7.16	1008	1080	60.3

Table 4: Experiment 2 on Movies database

From the above experiments one can see that CROSSCLUS successfully finds features pertinent to the clustering tasks, which shows the effectiveness of our approach for feature selection. CROSSCLUS generates reasonable clustering results and achieves significantly higher accuracy and efficiency than the other three approaches. It shows that user guidance really plays an important role in clustering.

6.2 Scalability Results

---

[5]Clustering with a set of coherent numerical features often has low accuracy, because numerical features do not provide clear ways for partitioning tuples. For example, if actors are clustered with their birthyear by two algorithms, it is quite possible that one generates clusters like (1920—1929, 1930—1939, ...), while the other generates (1925—1934, 1935—1944, ...).

We first test the scalability of CrossClus, Proclus, and RDBC w.r.t. the sizes of databases. We use TPC-H databases[6] of raw data sizes from 5MB to 25MB. The following query is used "CLUSTER *Customer* WITH *Orders.totalprice*". For each database, the CrossClus algorithm selects the following four features for clustering: average total price of orders, priority of orders, mktsegments of customers, and regions of customers. Please notice that CrossClus finds categorical pertinent features although the initial feature is numerical.

We test the running time of CrossClus, Proclus, and RDBC as shown in Figure 15 (a) (in log scale), and that of CrossClus is shown in Figure 15 (b). It can be seen that CrossClus and Proclus are linearly scalable w.r.t. database size, and CrossClus is substantially more efficient. RDBC becomes unaffordable for large datasets.
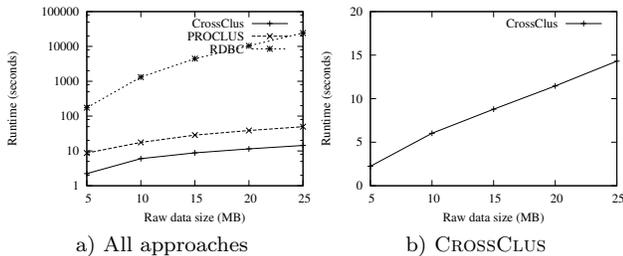

Figure 15: Scalability on TPC-H

We also test scalability w.r.t. number of relations in databases. We use the data generator used in [20], which randomly generates relational schemas and fills data according to some rules. The expected number of tuples in each relation is to 1000. The results are shown in Figure 16. It can be seen that CrossClus is scalable w.r.t. the number of relations.
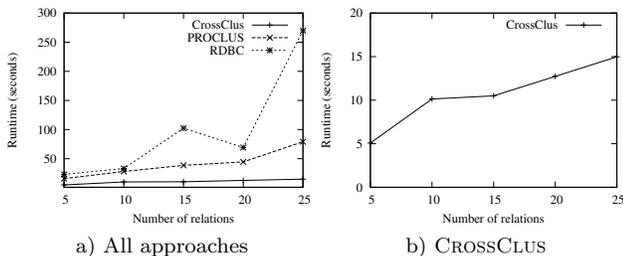

Figure 16: Scalability vs. #relation

7. DISCUSSION ON SCALABILITY

When a database cannot fit in memory, random access on disk-resident data will be expensive. Here we discuss solutions for different components of CrossClus when data is disk-resident. When generating values for a tuple on a feature (Section 4.2), tuple ID propagation can be done in a similar way as joining relations in a relational database. Actually, if the data is stored in a relational database, an SQL query can be used to propagate tuple IDs along a join path. When computing the similarity between two features f and g (Section 4.1), only one sequential scan is needed. Moreover, the similarities between multiple pair of features can be computed simultaneously by one scan. If f is a numerical feature, we may need to sort tuples according to f. After sorting, we can use one sequential scan to compute similarities between f and all other existing features. In general, CrossClus can select pertinent features with a

small number of scans on the dataset. For clustering target tuples (Section 5.2), CrossClus uses Clarans [19], a scalable clustering algorithm. Therefore, in general CrossClus is adaptable to disk-resident data in large databases.

8. CONCLUSIONS

In this paper we propose CrossClus, an efficient and effective approach for cross-relational clustering. Unlike previous approaches that work only on single tables, CrossClus works on multi-relational data, thus can be applied in many new fields. Because there exist numerous features with various semantic meaning, CrossClus employs a new concept: *user-guided clustering*, which selects features and performs clustering based on user's guidance. We also propose a new similarity measure for feature selection based on how they cluster tuples, which successfully identify pertinent features in experiments. Our experiments show that CrossClus generates meaningful clusterings that match the users' expectation and achieves high efficiency and scalability. We believe CrossClus represents a promising direction for clustering high-dimensional data in multiple relations.

9. REFERENCES

[1] C.C. Aggarwal, P.S. Yu. Finding Generalized Projected Clusters in High Dimensional Spaces. *SIGMOD*, 2000.

[2] C.C. Aggarwal, C. Procopiuc, J.L. Wolf, P.S. Yu, J.S. Park. Fast Algorithms for Projected Clustering. *SIGMOD*, 1999.

[3] P. Cheeseman, et al. AutoClass: A Bayesian Classfication System. *ICML*, 1988.

[4] J.G. Dy, C.E. Brodley. Feature Selection for Unsupervised Learning. *J. Machine Learning Research*, 2004.

[5] W. Emde, D. Wettschereck. Relational Instance-Based Learning. *ICML*, 1996.

[6] V. Ganti, J. Gehrke, R. Ramakrishnan. CACTUS - Clustering Categorical Data Using Summaries. *KDD*, 1999.

[7] T. Gärtner, J. W. Lloyd, P. A. Flach. Kernels and Distances for Structured Data. *Machine Learning*, 57, 2004.

[8] I. Guyon, A. Elisseeff. An Introduction to Variable and Feature Selection. *J. Machine Learning Research*, 2003.

[9] M.A. Hall. Correlation-based Feature Selection for Discrete and Numeric Class Machine Learning. *ICML*, 2000.

[10] V. Hristidis, Y. Papakonstantinou. DISCOVER: Keyword Search in Relational Databases. *VLDB*, 2002.

[11] L. Kaufman, P.J. Rousseeuw. Finding Groups in Data: an Introduction to Cluster Analysis. John Wiley and Sons, 1990.

[12] K. Wagstaff, C. Cardie, S. Rogers, S. Schroedl. Constrained k-means clustering with background knowledge. *ICML*, 2001.

[13] H. Kim, S. Lee. A semi-supervised document clustering technique for information organization. *CIKM*, 2000.

[14] M. Kirsten, S. Wrobel. Relational Distance-Based Clustering. *ILP*, 1998.

[15] M. Kirsten, S. Wrobel. Extending K-Means Clustering to First-order Representations. *ILP*, 2000.

[16] J. MacQueen. Some Methods for Classification and Analysis of Multivariate Observations. Berkeley Symposium, 1967.

[17] T.M. Mitchell. Machine Learning. McGraw Hill, 1997.

[18] P. Mitra, C.A. Murthy, S.K. Pal. Unsupervised Feature Selection Using Feature Similarity. *PAMI*, 2002.

[19] R.T. Ng, J. Han. Efficient and Effective Clustering Methods for Spatial Data Mining. *VLDB*, 1994.

[20] X. Yin, J. Han, J. Yang, P.S. Yu. CrossMine: Efficient Classification Across Multiple Database Relations. *ICDE*, 2004.

[21] E. P. Xing, A. Y. Ng, M. I. Jordan, S. Russell. Distance metric learning, with application to clustering with side-information. *NIPS*, 2002.

---

[6]TPC-H benchmark database. http://www.tpc.org/tpch.