Mining Database Structure; Or, How to Build a Data Quality Browser

Tamraparni Dasu, Theodore Johnson, S. Muthukrishnan, Vladislav Shkapenyuk
AT&T Labs–Research
{tamr,johnsont,muthu,vshkap}@research.att.com

ABSTRACT

Data mining research typically assumes that the data to be analyzed has been identified, gathered, cleaned, and processed into a convenient form. While data mining tools greatly enhance the ability of the analyst to make data-driven discoveries, most of the time spent in performing an analysis is spent in data identification, gathering, cleaning and processing the data. Similarly, schema mapping tools have been developed to help automate the task of using legacy or federated data sources for a new purpose, but assume that the structure of the data sources is well understood. However the data sets to be federated may come from dozens of databases containing thousands of tables and tens of thousands of fields, with little reliable documentation about primary keys or foreign keys.

We are developing a system, *Bellman*, which performs data mining on the *structure* of the database. In this paper, we present techniques for quickly identifying which fields have similar values, identifying join paths, estimating join directions and sizes, and identifying structures in the database. The results of the database structure mining allow the analyst to make sense of the database content. This information can be used to e.g., prepare data for data mining, find foreign key joins for schema mapping, or identify steps to be taken to prevent the database from collapsing under the weight of its complexity.

1. INTRODUCTION

A seeming invariant of large production databases is that they become disordered over time. The disorder arises from a variety of causes including incorrectly entered data, incorrect use of the database (perhaps due to a lack of documentation), and use of the database to model unanticipated events and entities (e.g., new services or customer types). Administrators and users of these databases are under demanding time pressures and frequently do not have the time to carefully plan, monitor, and clean their database. For example, the sales force is more interested in making a sale than in correctly modeling a customer and entering all information related to the sale, or a provisioning group may promptly enter in the service/circuits they provision but might not delete them as diligently.

Unfortunately, these disordered databases have a significant cost. Planning, analysis, and data mining are frustrated by incorrect or missing data. New projects which require access to multiple databases are difficult, expensive, and perhaps even impossible to implement.

A variety of tools have been developed for database cleaning [42] and for schema mapping [35], as we discuss in more detail below. In our experience, however, one is faced with the difficult problem of understanding the contents and structure of the database(s) at hand before they can be cleaned or have their schemas mapped. Large production databases often have hundreds to thousands of tables with thousands to tens of thousands of fields. Even in a clean database, discovering the database structure is difficult because of the scale of the problem.

Production databases often contain many additional problems which make understanding their structure much more difficult. Constructing an entity (e.g., a corporate customer or a data service offering) often requires many joins with long join paths, often across databases. The schema documentation is usually sparse and out-of-date. Foreign key dependencies are usually not maintained and may degrade over time. Conversely, tables may contain undocumented foreign keys. A table may contain heterogeneous entities, i.e. sets of rows in the table that have different join paths. The convention for recording information may be different in different tables (e.g. a customer name might be recorded in one field in one table, but in two or more fields in another).

As an aid to our data cleaning efforts, we have developed *Bellman*, a data quality browser. Bellman provides the usual query and schema navigation tools, and also a collection of tools and services which are designed to help the user discover the structure in the database. Bellman uses *database profiling* [13] to collect summaries of the database tablespaces, tables, and fields. These summaries are displayed to the user in an interactive manner or are used for more complex queries. Bellman collects the conventional profiles (e.g., number of rows in a table, number of distinct values in a field, etc.), as well as more sophisticated profiles (which is one of the subjects of this paper).

In order to understand the structure of a database, it is necessary to understand how fields relate to one another. Bellman collects concise summaries of the values of the fields. These summaries allow Bellman to determine whether two

ACM SIGMOD '2002 June 4-6, Madison, Wisconsin, USA

fields can be joined, and if so the direction of the join (e.g. one to many, many to many, etc.) and the size of the join result. Even when two fields cannot be joined, Bellman can use the field value summaries to determine whether they are textually similar, and if the text of one field is likely to be contained in another. These questions can be posed as queries on the summarized information, with results returned in seconds. Using the summaries, Bellman can pose data mining queries such as,

- Find all of the joins (primary key, foreign key, or otherwise) between this table and any other table in the database.
- Given field F, find all sets of fields $\{\mathcal{F}\}$ such that the contents of F are likely to be a composite of the contents of $\mathcal{F}$.
- Given a table T, does it have two (largely) disjoint subsets which join to tables $T1$ and $T2$ (i.e. is T heterogeneous)?

These data mining queries, and many others, can also be answered from the summaries, and therefore evaluated in seconds. This interactive database structure mining allows the user to discover the structure of the database, enabling the application of data cleaning, data mining, and schema mapping tools.

1.1 Related Work

The database research community has explored some aspects of the problem of data cleaning [42]. One aspect of this research addresses the problem of finding duplicate values in a table [22, 32, 37, 38]. More generally, one can perform *approximate matching*, in which joins predicates can be based on string distance [37, 39, 20]. Our interest is in finding related fields among all fields in the database, rather than performing any particular join. In [8], the authors compare two methods for finding related fields. However these are crude methods which heavily depend on schema information.

AJAX [14] and ARKTOS [46] are query systems designed to express and optimize data cleaning queries. However, the user must first determine the data cleaning process.

A related line of research is in schema mapping, and especially in resolving naming and structural conflicts [3, 27, 41, 36, 21]. While some work has been done to automatically detect database structure [21, 11, 33, 10, 43, 5], they are aimed at mapping particular pairs of fields, rather than summarizing an entire database to allow data mining queries.

Several algorithms have been proposed to find keys and functional dependencies (both exact and approximate) in tables [23, 4, 44, 45, 30]. While this information is valuable for mining database structure (and in fact we use an algorithm based on the one presented in [23]), additional information about connections between fields is needed.

Data summaries of various forms have been used in databases mainly for selectivity estimation; these typically include histograms, samples or wavelet coefficients [16]. We use several varieties of min-hash signatures and sketches in our work. Min-samples have been used in estimating transitive closure size [9], and selectivity estimation [7]. Sketches have been used in applications such as finding representative trends [24] and streaming wavelets [19]. Their use for data cleaning is novel.

In this paper, we make the following contributions:

- We develop several methods for finding related database fields using small summaries.
- We evaluate the size of the summaries required for accuracy.
- We present new algorithms for mining the structure of a database.

2. SUMMARIZING VALUES OF A FIELD

Our approach to database structure mining is to first collect *summaries* of the database. These summaries can be computed quickly, and represent the relevant features of the database in a small amount of space. Our data mining algorithms operate from these summaries, and as a consequence are fast because the summaries are small.

Many of these summaries are quite simple, e.g. the number of tuples in each table, the number of distinct and the number of null values of each field. etc. Other summaries are more sophisticated and have significantly more power to reveal the structure of the database. In this section, we present these more sophisticated summaries and the algorithms which use them as input.

2.1 Set Resemblance

The *resemblance* of two sets A and B is $\rho = |A \cap B|/|A \cup B|$. The resemblance of two sets is a measure of how similar they are. These sets are computed from fields of a table by a query such as A =Select Distinct R.A. For our purposes we are more interested in computing size of the intersection of A and B, which can be computed from the resemblance by

$$|A \cap B| = \frac{\rho}{\rho + 1}(|A| + |B|) \quad (1)$$

Our system profiles the number of distinct values of each field, so $|A|$ and $|B|$ is always available.

The real significance of the resemblance is that it can be easily estimated. Let $\mathcal{U}$ be the universe of values from which elements of the sets are drawn, and let $h : \mathcal{U} \to \mathcal{N}$ map elements of $\mathcal{U}$ uniformly and "randomly" to the set of natural numbers $\mathcal{N}$. Let $s(A) = \min_{a \in A}(h(a))$. Then

$$\Pr[s(A) = s(B)] = \rho$$

That is, the indicator variable $I[s(A) = s(B)]$ is a Bernoulli random variable with parameter ρ. For a proof of this statement, see [6], but we give a quick intuitive explanation. Consider the inverse mapping $h^{-1} : \mathcal{N} \to \mathcal{U}$. The function h^{-1} defines a sampling strategy for picking an element of the set A, namely select $h^{-1}(0)$ if it is in A, else $h^{-1}(1)$, else $h^{-1}(2)$, and so on. The process is similar to throwing darts at a dart board, stopping when an element of the set is hit. Let us consider $h^{-1}(k)$, the first element in the sequence which is in $A \cup B$. The chance that $h^{-1}(k) \in A \cap B$ is ρ, and is indicated by $s(A) = s(B)$.

A Bernoulli random variable has a large variance relative to its mean. To tighten the confidence bounds, we collect N samples. The *signature* of set A is $S(A) = (s_1(A), \ldots, s_N(A))$, where each $s_i(A) = \min_{a \in A}(h_i(a))$. We estimate ρ by

$$\hat{\rho} = \sum_{i=1,\ldots,N} I[s_i(A) = s_i(b)]/N$$

where $N\hat{\rho}$ has a $Binomial(\rho, N)$ distribution. For practical purposes, each of the hash functions h_i could be pairwise independent.

In our use of signatures, we make two modifications for simple and fast implementation. First, we map the set of all finite strings to the range $[0, 2^{31} - 1]$ for convenient integer manipulation. While there is the possibility of a collision between hashed elements, the probability of a collision is small if the set size is small compared to the hash range (e.g. only a few million distinct values) [6]. For very large databases, a hash range of $[0, 2^{63} - 1]$ manipulated by long longs should be used. Second, we do not use "random" permutations, instead we use a collection of hash functions, namely the polynomial hash functions described in [29] page 513. Therefore we need an experimental evaluation (given in Section 5) to determine what size set signature is effective.

The set signature conveys more information than set resemblance, it also can estimate $\rho_{A\setminus B} = |A \setminus B|/|A \cup B|$ and $\rho_{A\setminus B} = |B \setminus A|/|A \cup B|$ by

$$\widehat{\rho}_{A\setminus B} = \sum_{i=1,\ldots,N} I[s_i(A) < s_i(B)]/N$$

$$\widehat{\rho}_{B\setminus A} = \sum_{i=1,\ldots,N} I[s_i(A) > s_i(B)]/N$$

Although this is a simple observation given the context of the min hashes, we are not aware of any previous work that made use of this observation.

If $\hat{\rho}$ and $\widehat{\rho}_{B\setminus A}$ are large, but $\widehat{\rho}_{A\setminus B}$ is small, we can conclude that A is largely contained in B, but B is not contained in A. Finally, we note that set signatures are summable. That is

$$S(A \cup B) = (\min(s_1(A), s_1(B)), \ldots, \min(s_N(A), s_N(B)))$$

We use set signatures to find exact match joins in the database. During profiling, a set signature is collected for every field with at least 20 distinct values (a user adjustable parameter). Non-string fields are converted to their string representations. These are stored in the database using the schema:

SIGNATURE(Tablespace, Table, Field, HashNum, HashVal)

We can find all fields which resemble a particular field X.Y.Z using the query:

```
Select T2.Tablespace, T2.Table, T2.Field, count(*)
From SIGNATURE T1, SIGNATURE T2
Where T1.Tablespace='X' AND T1.Table='Y' AND
    T1.Field='Z' AND T1.HashNum=T2.HashNum
    AND T1.HashVal = T2.HashVal
```

Given a resemblance between two fields, we can compute the size of their intersection using formula 1 and filter out fields with small intersection size.

2.2 Multiset Resemblance

A field of a table is a multiset, i.e., a given element can appear many times. While set resemblance can provide information about whether two multisets have overlapping values, it does not provide any information about the multiplicity of the elements in the multiset. In many cases, we would like to have the multiplicity information. For example, would the join be one-to-one, one-to-many, or many to many? If it is e.g. one to many, is it 1 to 2, 1 to 20, or 1 to 200? How large would the join be? Is there a dependence between overlapping field values and value frequency? Previous join size estimation methods relied on sampling from relations for each join [15, 2]. Our method gives a alternate approach which is almost "free", given that we collect the set signatures.

A database profile typically contains the number of tuples in a table, and for each field the number of distinct and null values. These numbers can be used to compute the average multiplicity of the field values. However, this is a crude estimate. For example, many fields have one or more default values (other than NULL) which occur very frequently and bias the average value frequency.

Let $\mathbf{A}$ be a multiset, and let A be the set $\{a : a \in \mathbf{A}\}$. Let $m(a, \mathbf{A})$ be the number of times that element a occurs in $\mathbf{A}$. We define the *min hash count* [40] to be $M_i(\mathbf{A}) = m(h_i^{-1}(A), \mathbf{A})$, i.e., the number of times that $h_i^{-1}(A)$ occurs in multiset $\mathbf{A}$. Because $h_i^{-1}(A)$ is selected uniformly randomly among the set of distinct values in A, $m_i(\mathbf{A})$ is an unbiased estimator of the *frequency distribution* $\mathcal{F}(\mathbf{A})$ of $\mathbf{A}$. That is

$$\mathcal{F}(\mathbf{A}) = \left\{ \Pr[m_i(\mathbf{A}) = k] = \frac{|a \in \mathbf{A} \ : m(a, \mathbf{A}) = k|}{|A|} : k \geq 1 \right\}$$

Since each sample $M_i(\mathbf{A})$ is a Bernoulli random variable with parameter $\Pr[m_i(A) = k]$ for each k, we can estimate the $\mathcal{F}(\mathbf{A})$ with $\widehat{\mathcal{F}}(\mathbf{A})$, where

$$\widehat{\mathcal{F}(\mathbf{A})} = \left\{ \frac{\sum_{i=1}^{N} I[M_i(\mathbf{A}) = k]}{N} : k \geq 1 \right\}$$

In conventional sampling, items are uniformly randomly selected from $\mathbf{A}$. By contrast, min hash sampling uniformly randomly selects items from A, and the min hash count collects their multiplicities. The combination of min hash sampling and min hash counts create a *multiset signature*. In [17], the author presents a sophisticated method for sampling distinct values and used it for estimating the number of unique values in a field, but their scheme does not have the estimation properties above of the multiset signatures.

Like set signatures, multiset signatures are summable, using

$$\begin{aligned} s_i(\mathbf{A} \cup \mathbf{B}) &= \min(s_i(\mathbf{A}), s_i(\mathbf{B})) \\ m_i(\mathbf{A} \cup \mathbf{B}) &= m_i(\mathbf{A}) && s_i(\mathbf{A}) < s_i(\mathbf{B}) \\ &= m_i(\mathbf{B}) && s_i(\mathbf{A}) > s_i(\mathbf{B}) \\ &= m_i(\mathbf{A}) + m_i(\mathbf{B}) && s_i(\mathbf{A}) = s_i(\mathbf{B}) \end{aligned}$$

We can use the min hash count in a variety of ways.

- The min hash count provides an accurate estimate of the tail of the distribution of values in $\mathbf{A}$. In conjunction with techniques which accurately estimate the head of the distribution (e.g., end biased histograms[25]), we can characterize the entire frequency distribution of $\mathbf{A}$.
- The min hash count not only estimates the frequency distribution, but does so in conjunction with a set signature. We can therefore compute a variety of useful information about the nature of a join.
  - We can compute the *direction* of the join, i.e. is it one-to-one, one-to-many, many-to-many, and

how many? Suppose we are given the multiset signatures of **A** and **B**. Then, frequency distribution of the elements of **A** which join with **B** (respectively, **B** with **A**) is given by

$$\widehat{\mathcal{F}}(\mathbf{A}|\mathbf{B}) = \left\{ \frac{\sum I[M_i(\mathbf{A}) = k] * I[S_i(\mathbf{A}) = S_i(\mathbf{B})]}{\sum_i I[S_i(\mathbf{A}) = S_i(\mathbf{B})]} : k \geq 1 \right\}$$

- We can compute a *distribution dependence* of **A** on **B** (and vice versa) by comparing $\widehat{\mathcal{F}}(\mathbf{A})$ and $\widehat{\mathcal{F}}(\mathbf{A}|\mathbf{B})$, e.g. by comparing their mean values, using a χ^2 test, etc.
- We can estimate the *join size* of $\mathbf{A} \bowtie \mathbf{B}$ using multiset signatures. Let:

$$E[M(\mathbf{A} \bowtie \mathbf{B})] = \frac{\sum_{i=1}^{N} M_i(\mathbf{A}) M_i(\mathbf{B}) I[S_i(\mathbf{A}) = S_i(\mathbf{B})]}{\sum_{i=1}^{N} I[S_i(\mathbf{A}) = S_i(\mathbf{B})]}$$

The estimate $E[M(\mathbf{A} \bowtie \mathbf{B})]$ is the average number of tuples that each element in $A \cap B$ contributes to the join result. Therefore the estimated join size is :

$$\widehat{|\mathbf{A} \bowtie \mathbf{B}|} = E[M(\mathbf{A} \bowtie \mathbf{B})] \frac{\hat{\rho}}{1+\hat{\rho}} (|A| + |B|)$$

- We can estimate the frequency distribution of the join result using

$$\widehat{\mathcal{F}}(A \bowtie \mathbf{B}) = \left\{ \frac{\sum I[M_i(\mathbf{A}) M_i(\mathbf{B}) = k] * I[S_i(\mathbf{A}) = S_i(\mathbf{B})]}{\sum I[S_i(\mathbf{A}) = S_i(\mathbf{B})]} : k \geq 1 \right\}$$

The multiset signatures can be stored in a database using a schema similar to that used for the set signatures, e.g.,

MULTISETSIG(Tablespace, Table, Field, HashNum, HashVal, MinCnt)

For an example, we can find the estimated frequency distribution of field X.Y.Z using

```
Select MinCnt count(*)
From MULTISETSIG
Where Tablespace = 'X' AND Table = 'Y' AND Field = 'Z'
Group By MinCnt
```

2.3 Substring Resemblance

Databases maintained by different organizations often represent data using different formats, but in "textually similar" ways. For example, database D_1 might store customer names in a field *CustName* in the format 'LastName, FirstName', while database D_2 might store customer names in two fields, *LastName* and *FirstName*. Another common occurrence is for a key to have extra text appended, prepended, or inserted into the middle. For example, D_1 might store Social Security numbers in the format 'ddd-dd-dddd', while D_2 uses the format 'SSddd-dd-dddd' (often the additional text has context-specific meaning).

Finding join paths in which the keys must be catenated, transformed, or both is very difficult, because automated join testing does not incorporate transforms. It is often the case that humans can identify common patterns in two fields and manually determine the transform. This process is labor intensive, so a filtering mechanism is needed to eliminate fields which are obviously poor matches.

Finding substring similarity between two fields is a difficult problem because of the huge number of substrings of the two fields which must be compared. A typical approach for reducing the complexity of the problem is to summarize the substrings in a field with *q-grams*, the set of all q-character substrings of the field (see [20] and the references therein).

For our purposes, the set of all q-grams is likely to be too large for convenient storage and manipulation (e.g., there are 2,097,152 possible 7 bit ASCII 3-grams). Therefore we will view the q-grams of a field as a set or multiset and store a summary of it.

2.3.1 Q-gram Signature

A *q-gram signature* is a set signature of the set of q-grams of a set A or a multiset **A**. A q-gram signature is computed by first computing the $QGRAM(A)$, set of all q-grams of A, then computing the set signature of the q-gram set. The q-gram resemblance of two sets A and B is:

$$\rho^q = \frac{|QGRAM(A) \cap QGRAM(B)|}{|QGRAM(A) \cup QGRAM(B)|}$$

and is estimated by

$$\hat{\rho}^q = \sum_{i=1,\ldots,N} I[s_i(QGRAM(A)) = s_i(QGRAM(B))]/N$$

Since we compute $QGRAM(A)$ before computing its set signature, we can store $|QGRAM(A)|$ as well as the q-gram signature. Therefore we can estimate the size of the intersection of two q-gram sets by

$$\widehat{|QGRAM(A) \cap QGRAM(B)|} = \frac{\hat{\rho}^q}{1+\hat{\rho}^q} (|QGRAM(A)| + |QGRAM(B)|)$$

Two fields which have a small or zero set resemblance but a large q-gram resemblance are likely to be a related by a small transformation. However, it is often the case that the set of q-grams of one field B are (largely) contained in the set of q-grams of another field B (for example, if values in field B contain a few extra characters, or are composed of values from field A catenated with values from field C). We recall from Section 2.1 a couple of useful properties of set signatures.

- We can determine q-gram set containment using the q-gram analogs of $\rho_{A \setminus B}$ and $\rho_{B \setminus A}$,

$$\hat{\rho}^q_{A \setminus B} = \sum_{i=1}^{N} I[s_i(QGRAM(A)) < s_i(QGRAM(B))]/N$$

$$\hat{\rho}^q_{B \setminus A} = \sum_{i=1}^{N} I[s_i(QGRAM(A)) > s_i(QGRAM(B))]/N$$

If A is mostly composed of substrings of B, then $\hat{\rho}^q$ will be large and $\hat{\rho}^q_{A \setminus B}$ will be small.

- Recall that set signatures are summable by taking $\min(S_i(A), S_i(B)), i = 1, \ldots, N$. Suppose we are given two fields A and C whose q-gram sets are contained in field B's q-gram set. If $QGRAM(A) \cup QGRAM(C)$ covers a significantly larger fraction of $QGRAM(B)$ than either set of q-grams alone, it is likely that B is a composite of A and C. When applied to finding join paths, we can restrict our attention to fields from table

T_1 which are contained in a field of T_2, leaving a highly tractable search process.

- The q-gram signatures can be stored in the database and queries in a manner analogous to that shown in Section 2.1.

2.4 Q-gram Sketches

Another method for determining substring similarity is to use *sketches* [26], a dimensionally reduced representation of a vector. Let V be a d dimensional vector, and let $X_1, \ldots, X_k$ be k d-dimensional vectors with randomly chosen values (typically, $k \ll d$). The *sketch* of V, $Sk(V)$, is

$$Sk(V) = (V \cdot X_1, \ldots, V \cdot X_k)$$

That is, a sketch of a vector is a collection of random projections. The L_2 distance between vectors V_1 and V_2, $L_2(V_1, V_2)$ is approximated by the function

$$\widehat{L_2}(V_1, V_2) = \sqrt{\sum_{i=1}^{k} (Sk(V_1)[i] - Sk(V_2)[i])^2 / k}$$

In our application, we are interested in the *q-gram vector distance*. The q-gram vector of a field is a normalized count of the number of times each possible times a q-gram appears in field A:

$$QV(A)[i] = \frac{m(q_i, QGRAM(A))}{\sqrt{\sum_i m(q_i, QGRAM(A))^2}}$$

where q_i is the i^{th} q-gram. The *q-gram vector distance* is the distance between the q-gram vectors of two fields, e.g.

$$QD(A, B) = \sqrt{\sum_i (QV(A)[i] - QV(B)[i])^2}$$

We compute and store the sketches of the q-gram vectors, $SK(QV(A))$. Following [1], the entries in the random vectors X_j are 1 with probability 1/2, and -1 with probability 1/2. Suppose there are n vectors $QV(A_j)$ to be compared. If $k \geq 8/\epsilon^2$, then $\widehat{L_2}(QV(A_1), QV(A_2))$ is in a (ϵ, ϵ) confidence interval around $L_2(QV(A_1), QV(A_2))$.

Like the q-gram signature, the q-gram sketch can be used to find pairs of fields which are textually similar. While the q-gram signature represents the set of q-grams in a field, the q-gram sketch represents the multiset of q-grams in a field, The q-gram sketch is therefore the more discriminating summary because its measure is based on the distribution of field values. This can be a positive or a negative attribute, depending on the type of search being done.

Unlike q-gram signatures, it is difficult to determine set containment using q-gram sketches. However, sketches are summable, being linear combinations.

The q-gram sketches can be stored in the database using the schema:

QSKETCH(Tablespace, Table, Field, SkNum, SkVal)

We can find the q-gram vector distance from a particular field X.Y.Z to all other fields in the database using the result of the following query (and dividing by k, then taking the square root):

```
Select T2.Tablespace, T2.Table, T2.Field,
    Sum( (T1.SkVal-T2.SkVal)*(T1.SkVal-T2.SkVal))
From QSKETCH T1, QSKETCH T2
Where T1.Tablespace='X' AND T1.Table='Y' AND
    T1.Field='Z' AND T1. SkNum = T2.SkNum
```

2.5 Finding Keys

In order to determine the structure of a database, we need to determine the minimal keys for each table, e.g. to determine primary key-primary key or primary key-foreign key join paths. We implemented a levelwise key finding algorithm similar to *Tane* [23]. Since our algorithm is subsumed by Tane, we note only the significant differences between our implementations:

- We are interested only in keys, not all functional dependencies. Therefore we used a much simpler and more aggressive pruning condition.
- Tane uses a special file structure to accelerate the key finding process. Our software, Bellman, is intended to be an easily portable program residing on the client machine. Creating many large file on the client seems inadvisable, so instead we submit Count Distinct queries to the database.

3. MINING DATABASE STRUCTURES

The tools presented in the preceding section allow one to quickly determine components of the database structure. Once a summary of the database has been computed, they allow the user to submit browsing queries, such as *What is the set of keys for this table*, or *what other fields have values that are similar to this field*?

One can also use these tools to ask further questions about structure of the database. For example *What other tables join to this table?*, or *Is this field a composite of two or more fields in another table?* In this section, we outline how the following three queries can be quickly answered using the signatures and sketches:

- Finding join paths.
- Finding composite fields.
- Finding heterogeneous tables.

3.1 Finding Join Paths

Gives a table T, we would like to find all ways in which another table T' can join with T. Further, we would like to restrict our attention to joins involving keys of T, T', or both. It is clear that we can answer this query using the set signatures (or multiset signatures) and the list of keys of the tables.

1. Find all pairs of fields $J = \{(f_T, f'_T)\}$ such that the pair has a high resemblance, f_T is a field in T, and f'_T is a field in $T' \neq T$.
2. Partition J into $\{J_{T_1}, \ldots J_{T_m}\}$, where each J_{T_i} contains all and only those pairs with fields from tables T and T_i.
3. For each partition J_{T_i},
   (a) For each key K_T of table T such that all fields in K_T are in J_{T_i}, and can be matched with different fields of T_i.

i. Output all possible matchings of K_T with fields of T_i drawn from J_{T_i} such that each field of K_T is matched with a different field of T_i.

(b) Find all keys K_{T_i} of table T_i such that all fields in K_{T_i} are in J_T, and are matched with different fields of T.

By using multiset signatures, it is possible to evaluate the strength of the join. It the key K is on a single field, then the join size and direction estimation methods described in Section 2.2 are applicable. If key K has multiple components, we cannot use the join size and direction estimates directly. However, we can still get an indication of the strength of the join.

For concreteness, let the key be $K_T = (A, B)$. Let $Isct(T.A, T'.A)$ be the size of the intersection of sets $T.A$ and $T'.A$. Let the *coverage* of $T'.A$ be

$$Cvr(T'.A, T.A) = \frac{Isct(T.A, T'.A) \sum_{i=1}^{N} M_i(T'.A) I[s_i(T.A) = s_i(T'.A)]}{\sum_{i=1}^{N} I[s_i(T.A) = s_i(T'.A)]}$$

Similarly, define the intersection and coverage of $T'.B$ with $T.B$. Then we can estimate an upperbound on the number of tuples of T that will be joined with T' by

$$\min(Isct(T.A, T'.A) Isct(T.B, T'.B), Cvr(T'.A, T.A), Cvr(T'.B, T.B))$$

3.2 Finding Composite Fields

A common problem in finding joins between tables in different databases (and often, within the same database) is that two fields might contain the same information, but in a slightly different way (i.e., one field is a *transformation* of another). Often, the transformation from one field to another involves appending or prepending a short text (e.g. a customer identifier '10974' becomes 'C10974'). The q-gram signature and q-gram sketch summaries are useful for identifying the similarity of these pairs.

Other field transformations are more complex. One common complex transformation is when a field in one table is a composite of two or more fields in another table. For example a customer's name might be stored as 'Hamid Ahmadi' in the *Name* field of table $T1$, and as 'Hamid' and 'Ahmadi' in the *FirstName* and *LastName* fields, respectively, of table $T2$.

Substring search and indexing is a difficult problem in general. While we cannot identify precisely which fields combine to make another, we can use the properties of the q-gram signature to produce a small number of candidates which might combine to form the composite field.

1. Given field f,
2. Find the set of fields F with a high q-gram resemblance to f and whose q-grams are largely a subset of those in field F.
3. Partition F into $\{F_{T_1}, \ldots, F_{T_m}\}$ where each F_{T_i} contains all and only those fields in F drawn from table T_i.
4. For each partition F_{T_i},
   (a) Add all combinations of 2 or more fields of F_{T_i} to the candidate list C.
5. Rank the entries in C and sort by ranking.

Recall from Section 2.1 that we can determine whether f_1 is largely a subset of f_2 by using $\widehat{\rho}_{f_1 \setminus f_2}$. Depending on the minimum level of q-gram resemblance required for admission to f, the number of entries in C might be very large. In this case, the ranking step is critical for presenting a useful set of results to the user. Fortunately, there are several simple but useful ranking heuristics available. First, a smaller set of fields is more likely to be interesting than a larger one. Second, the composite set of fields can be ranked by their q-gram resemblance or their q-gram sketch distance from f (Recall that the signature or sketch of the q-grams of the union of two fields is easily computed from the signature or sketch of the q-grams of the two individual fields). Third, combinations of fields which increase their resemblance to f (or decrease their q-gram sketch distance) as compared to the individual fields might gain in ranking.

The algorithm step 4a) requires time proportional to $|F_{T_i}|!$, although in practice the output would be limited to sets with at most e.g. three fields. To understand what the output size would be, we ran a query to collect all pairs of fields with a q-gram resemblance of at least 40% such that $\hat{\rho}^q_{A \setminus B}$ or $\hat{\rho}^q_{B \setminus A}$ is no larger than 5%. We found that the average size of F_{T_i} is 3.8, the largest F_{T_i} contains 18 elements, and that 90% of the F_{T_i} have 7 or fewer elements.

3.3 Finding Heterogeneous Tables

Large production databases often become disordered because a new and unanticipated entity must be modeled (e.g. a new service offering, a new customer type, a new accounting procedure, etc.). Supporting the new entity often entails "pushing a square peg into a round hole" – the new entity is made to fit into the existing tables, often with the help of new supporting tables and dimension tables. After several of these additions, any given dimension or supporting table might join with only a portion of a fact table.

For example, a company might provide a fixed-price dialup Internet service, targeted at consumers. A database is designed to model individual consumers on a fixed-price plan. Later the company discovers that its Internet service offering is popular with small businesses, so it develops marketing and billing systems to better support the small business market. These require additional tables which describe the business customers. Shortly thereafter, the company decides to offer hourly rate plans as an alternative to the fixed price plan. A new table is added to the database which records modem usage for the hourly rate customers.

While any single adjustment to the database structure is simple to understand, after a long sequence of these adjustments the database structure can become quite complex. Often, these structures are not documented, and the data analyst or schema mapper must discover them by trial and error.

However, it is possible to use the properties of set signature to discover tables which are likely to be heterogeneous, and also the joins of the heterogeneous partitions of the table.

1. Given table T,
2. For every field f of table T

(a) Find J, the set of fields with high resemblance to f.

(b) Partition J into $\{J_{T_1}, \ldots, J_{T_m}\}$, where the fields in each J_{T_i} are all and only those in table T_i.

(c) For each J_{T_i},

i. Output all maximal subsets $\{f_1, \ldots f_m\}$ such that $|f \cap f_i \cap f_j|$ is small, $1 \leq i < j \leq m$. and each f_i is drawn from a different J_{T_k}.

Determining that $|f \cap f_i \cap f_j|$ can be done using a simple extension of set signatures (recall Section 2.1) of f, f_i and f_j, i.e. by computing

$$\sum_{k=1,\ldots,N} I[s_k(f) = s_k(f_i) = s_k(f_j)]/N$$

4. BELLMAN

We are developing *Bellman*, a database browser for complex databases. Bellman provides the usual database browser functions such as a query tool and a schema browser. Bellman also collects profiles of the database, and stores these profiles in the database itself. Many of the profiles which Bellman collects are simple, for example the number of tables in a tablespace, the number of rows in a table, the number of distinct and null values of each field. By caching these profiles in the database, they can be interactively displayed to the user. We have found that even this simple functionality greatly simplifies the database exploration process.

Bellman also collects the more complex profiles discussed in the previous section (keys, multiset signatures, q-gram signatures, and q-gram sketches). All fields (e.g., numeric, date, and fixed-length string) are converted to variable-length string fields before profiling. We have implemented a variety of tools (available as GUI windows callable by a button click) which allow the user to query the information in these profiles in an interactive way. For example, one tool allows the user to select a field of a table, and then find all other fields in the database that have a high resemblance. The output is loaded in a table which indicates the field name, the estimated resemblance, and the estimated intersection size. The user can select one of these fields and obtain more information, including the frequency distribution of the field, the estimated join size, the frequency distribution of the values of the fields which participate in the join, and the frequency distribution of the join result. The user can also submit canned queries to the (non-profile) database to determine the actual intersection and join size, and to obtain a sample of values of the field.

Even the complex data mining queries of Section 3 are implemented mostly through SQL queries to the profile data in the database. Because the profiles are small, the response time is fast enough to be interactive.

We implemented Bellman in Java using JDBC to access an Oracle database. Because Bellman is intended to be an easily installed client, we made the decision to perform all of the analysis, including the profile computation, at the client. We found that Java and JDBC are far too slow and buggy for actual use, so we wrote the analysis code in C++ using the OTL [31] library to access the database. OTL is a C++ wrapper which hides the details of database accesses and which can use several connection methods including ODBC and OCI. We obtained good performance running Bellman on a 700 Mhz Pentium III PC and accessing a 8-processor 250 Mhz Solaris server running Oracle. The portable nature of the Bellman code allows us to run it on the server as well.

5. EXPERIMENTS

We ran a collection of experiments to determine whether signatures and sketches can find similar fields, and the size of the signature or sketch required to do so. We also ran several experiments to determine the usefulness of the answers, for which we can give qualitative results.[1]

For our data set, we used a snapshot of a database which describes several aspects of a large data networking service. The full database contains 926 tables and 15,442 fields in four tablespaces. For the experiments described in this paper, we used the three smaller tablespaces containing 267 tables and 3,356 fields. We collected signature and sketch profiles on all fields containing at least 20 distinct values (1,078 fields met this criteria).

When computing signatures and sketches, we collected 250 samples. We computed 3-grams of the values of the fields for the q-gram signatures and sketches. Computing all of the profiles except for the keys required less than three hours. Determining the keys of the tables is computationally expensive, using a large number of count distinct queries, and required about 8 hours.

5.1 Estimating Field Intersection Size

For our first experiment, we evaluated the use of set signatures for estimating the size of the intersection between two fields. For a test data set, we collected 68 random pairs of fields with a resemblance of at least 5%. For each pair, we computed the estimated intersection size using signatures of size 50, 100, 150, 200, and 250 samples, and also the exact intersection size.

In Figures 1 and 2 we plot the error in the estimate of the intersection size (relative to the actual intersection size) using 50 and 100 samples in the signature, respectively. The X axis is the actual resemblance. As the charts show, the error in the intersection size estimate increases significantly when the resemblance is small (i.e., 25% or less). Except for three data points, the 50-sample estimate is able to distinguish between small, medium and large intersections, which is good enough for the purpose of finding fields with similar values. The 100-sample estimate is able to distinguish intersection sizes for all samples. Larger sample sizes improve accuracy, but not enough to justify the increased storage or evaluation time cost.

Finding all fields with a large resemblance to a particular field required about 90 seconds using 250-sample signatures on 1,078 profiled fields. We repeated the timing test on a Bellman installation in which all four tablespaces were profiled, and found that the same query requires about 30 seconds using 50-sample signatures on 4,249 profiled fields.

5.2 Estimating Join Sizes

In our next experiment, we evaluate the ability of the multiset signature to estimate the frequency distribution of the values in a field. As these results would be difficult to present visually, we instead use an interesting proxy: estimating the size of a join. If we can estimate the join size accurately, we can certainly estimate the frequency distribution accurately.

[1]The data we used in our experiments is sensitive corporate data which cannot be released to the public.

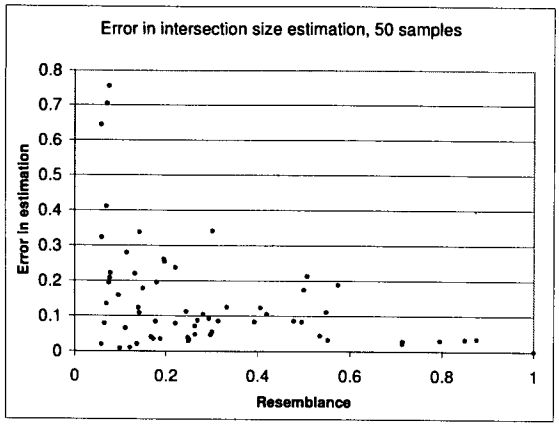

Figure 1: Intersection size estimation, 50 samples.

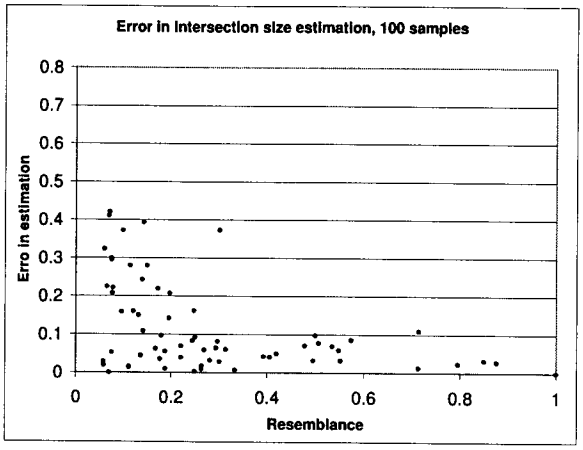

Figure 2: Intersection size estimation, 100 samples.

We reused the same 68 random pairs of fields that were used in the intersection size experiment. For each pair of fields, we estimated their join size using signatures of size 50, 100, 150, 200, and 250 samples, and also computed their exact join size.

In Figures 3 and 4 we plot the error in the estimate of the join size (relative to the actual join size) using 100 and 250 samples in the signature, respectively. The X axis is the actual resemblance.

The charts show that the multiset signature can compute the join size to within a 60% error for most of the sample points when the resemblance between the fields was large (e.g. at least 20%). This level of accuracy is good enough to get an idea of the size of the join result to within an order of magnitude. Increasing the sample size from 100 to 250 samples generally increases the accuracy of the estimate, except for three data points with a resemblance larger than 20% and a very high error. On closer examination, we found that the estimation error was due to a highly skewed distribution of values in these fields.

The skewness of the fields for which the multiset signature join size estimate fails provides us with a solution for avoiding highly inaccurate join size estimates. In addition to the

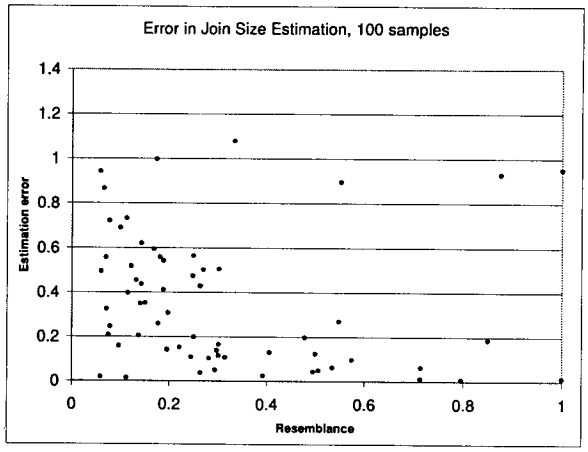

Figure 3: Join size estimation, 100 samples.

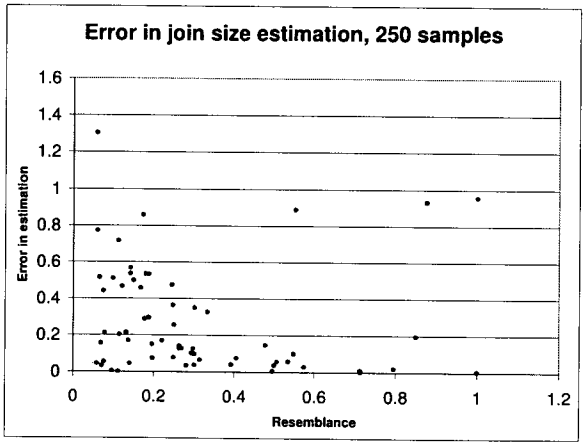

Figure 4: Join size estimation, 250 samples.

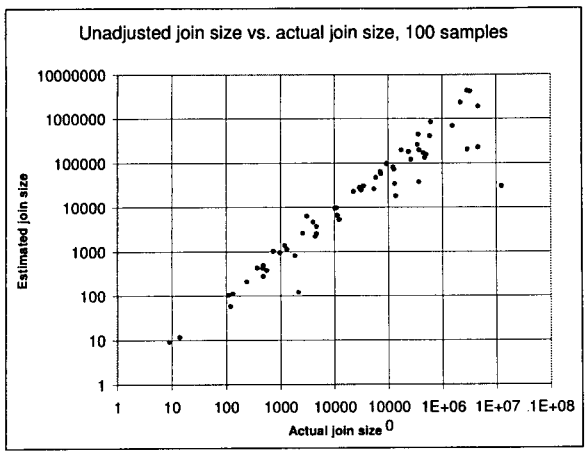

Figure 5: Unadjusted join size estimate, 100 samples.

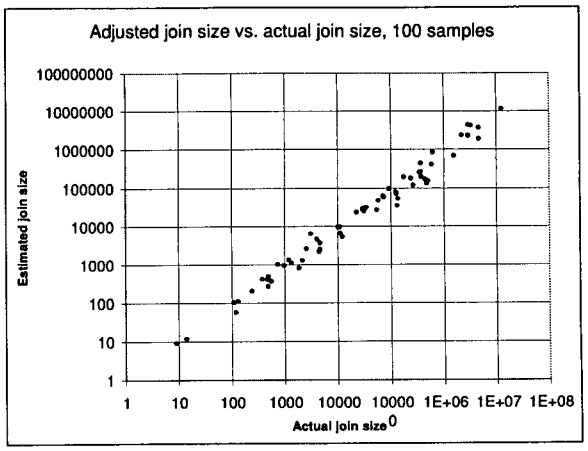

Figure 6: Adjusted join size estimate, 100 samples.

multiset signature, we collect the N most frequent values in the fields, and a count of their occurrence. We use these most-frequent values to compute a lower bound on the join size, namely the size of the join of the most-frequent samples of the fields.

We computed the adjustment and applied it to the data points in the 100-samples join size estimate. In figures 5, we plot the unadjusted 100-sample join size estimate against the actual join size on a log-log plot. The high-error estimates are the data points significantly below the trend line. We plot the adjusted estimates in 6. Although there are still several high-error estimates even after the adjustment, in every case we are able to correctly estimate the order of magnitude of the join size.

We note that the a count of the most frequent items can be computed quickly and in small space using the methods described in [18]. Furthermore, we do not need to store the actual values of the most frequent values, instead we can store their hash values. Based on our results, the join size of two fields (and therefore their frequency distributions) can be accurately estimated using multiset signatures with 100 samples plus an additional 10 most-frequent samples.

5.3 Q-gram Signatures

In our next experiment, we evaluate the accuracy of q-gram signatures. We randomly selected 67 pairs of fields with a q-gram resemblance of at least 15%. We estimated the q-gram resemblance using q-gram signatures with 50, 100, 150, 200, and 250 samples, and also computed the exact resemblance between the pairs of fields.

When finding similar fields using q-grams, we are interested in a similarity measure rather than a numeric value such as the intersection size. In Figures 7 and 8, we plot the estimated versus the actual resemblance using 50-sample q-gram signatures and 150-sample q-gram signatures, respectively. Even in the 50-sample case and even for low resemblance, the resemblance can be accurately estimated. The maximum estimation error (compared to the actual resemblance) is 47% in the 50-sample experiment, and 21% in the 150-sample experiment. Accurately estimating q-gram resemblance is easier than estimating intersection sizes because we are directly estimating the resemblance, and be-

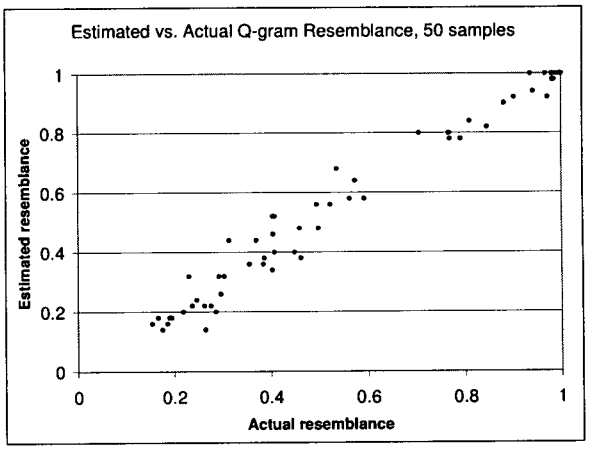

Figure 7: Q-gram resemblance estimation, 50 samples.

cause the universe of possible values is much smaller. We can accurately tell the difference between large, medium, and small resemblance using a 50-sample q-gram signature.

5.4 Q-gram Sketches

Finally, we evaluate the accuracy of q-gram sketches. For this experiment, we use the same set of 67 pairs of fields that we used in the q-gram signature experiment. We estimate the L_2 distance between the normalized q-gram occurrence vectors using q-gram sketches with 50, 100, 150, 200, and 250 samples, and also computed the actual L_2 distance.

The estimated q-gram vector distance is plotted against the actual q-gram vector distance in Figure 9 for 50-sample estimates, and in Figure 10 for 150-sample estimates. Although the 150-sample estimates have a much lower error than the 50-sample estimates, the 50-sample estimates can readily distinguish between small, medium, and large distances. We conclude that only 50 samples are required for a q-gram vector sketch.

We note that q-gram vector distance and q-gram resemblance are rather different and complementary measures. In Figure 11, we plot the actual q-gram vector distance against

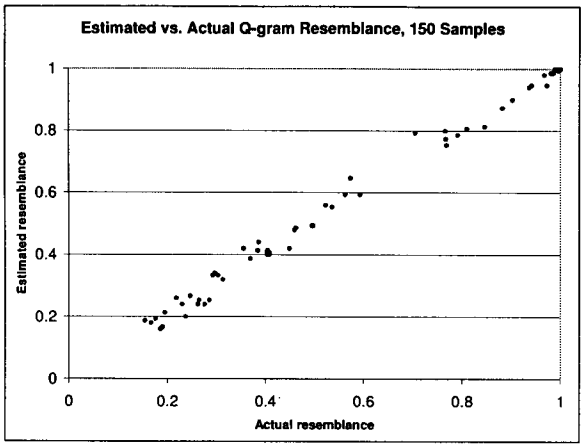

Figure 8: Q-gram resemblance estimation, 150 samples.

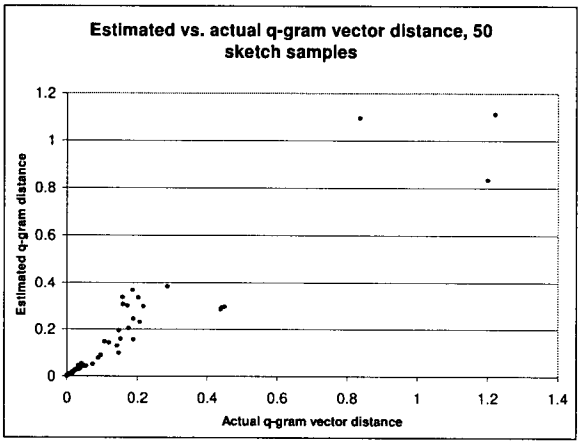

Figure 9: Q-gram vector distance estimation, 50 samples.

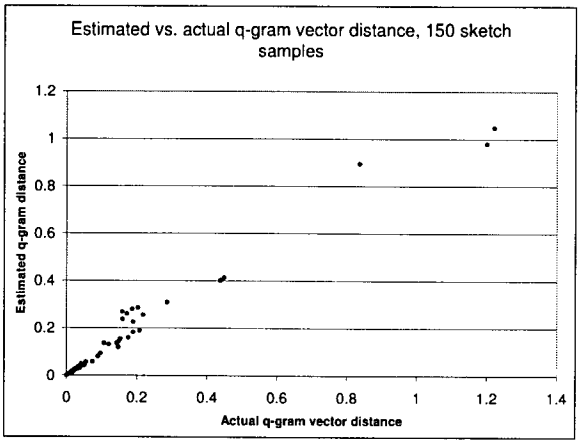

Figure 10: Q-gram vector distance, 150 samples.

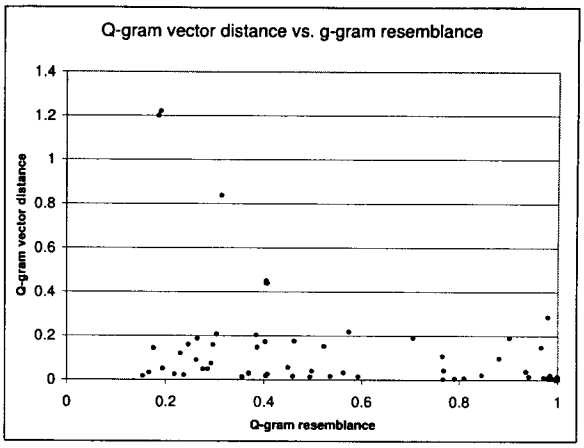

Figure 11: Q-gram vector distance vs. q-gram resemblance.

the actual q-gram resemblance for the 67 pairs of fields used in the previous experiment. Although high-resemblance pairs are not high-distance pairs, otherwise there is little correlation between resemblance and distance.

5.5 Qualitative Experiments

In the previous experiments, we have established that small summaries are sufficient for the purpose of exploring the structure of a large and complex database. However, the question of whether the summaries actually help in the exploration process still needs to be answered.

The data set we use for experiments is confidential corporate data, so we cannot discuss any particular query results. However, we can discuss in general the usefulness of the tools.

5.5.1 Using Multiset Resemblance

A version of Bellman which has been released to several users collects and stores multiset signatures of all fields with at least 20 distinct values (a user-adjustable parameter). Bellman contains a tool for querying the multiset signatures to find all fields with a minimum (estimated) resemblance to a given field. The result is displayed to the user in a table which includes columns indicating the estimated resemblance, estimated intersection size, estimated join frequency dependencies, and estimated join size.

The released version of Bellman has been used to explore the four tablespace, 926 table data set, a portion of which was used for the experimental data (actually, the problem of exploring this data set was the motivation for developing Bellman). Bellman's ability to quickly and interactively find high-resemblance fields has been a distinct valuable feature, and has led to several unexpected discoveries.

There are a variety of other methods for finding joinable fields. One method is to perform the join, but doing so is unacceptably slow. Another method is to look for fields with the same name. In our explorations, we have found that joinable fields rarely have the same name (and often do not even have similar names).

A more sophisticated method is to collect a variety of features of the field, such as the data type of the field, the length of the field, the nature of the characters in the field

(letters, numbers, punctuation). This method is used by systems which profile databases, typically as preparation for migrating a legacy database [12, 34, 28]. Fields which have similar features are judged to be similar. However, this type of profiling is susceptible to false positives and false negatives. The reason for the false positives is evident (e.g. two fields might contain an non-overlapping set of telephone numbers). When the database contains fifteen thousand fields, eliminating as many false positives as is possible is critical. False negatives can occur when a field is polluted with unexpected values, e.g. a field which is supposed to contain a 10 digit key might contain entries[2] such as "none", "no key", "Hamid Ahmadi", etc. The field seems to contain alphanumeric data, and therefore is not presented as a match to a field with 10 digit entries.

Some systems will attempt to classify the nature of the data, e.g. name, address, telephone number, etc. [43, 5]. This type of more selective feature can help to eliminate false positives, but it still suffers from the problems of the feature vector approach. In addition, the method is fragile – we always seem to encounter a new data format in our explorations. However we note that both of these types of profiling are useful adjuncts to the profiling techniques discussed in this paper.

5.5.2 *Using Q-gram Similarity*

The q-gram summaries (both signatures and sketches) find fields which are "textually similar" to a target field – that is, typical field values look similar to the human eye. A q-gram similarity query often returns a long list of similar fields (as is likely to happen when 1,000+ fields are profiled), but ranked by similarity. The fields in the result list are often semantically related to the target field. For example, a query to find fields similar to a City field returned many other fields containing city names, but also fields containing addresses and first and last names of people. A query on an field containing IP addresses returned other IP address fields, but also some fields containing fixed point numbers with three digits of precision after the decimal.

By pruning high value-resemblance fields from the result of a q-gram similarity query and providing a data exploration GUI, we can usually make serendipitous discoveries. For example, some of the fields found to be textually similar to the City were labeled Country. On closer examination, these fields actually contained *county* names (and many county and city names are identical). The q-gram similarity query on an IP address field returned a field containing class-C prefixes of the IP addresses (i.e., the first 3 octets of the 4-octet IP addresses).

We tried these q-gram similarity queries, and several others, using both q-gram signatures and q-gram sketches. As can be expected, we found the results of the query on the q-gram sketches to be highly targeted, while the query on the q-gram signatures to be much less so. Almost all of the highly ranked results returned by the q-gram sketch query were highly relevant, but the query missed several interesting fields returned by the q-gram signature. We conclude that these tools are complementary, and both are useful in a browser.

6. CONCLUSIONS

In this paper, we have explored the use of *min hash* signatures and *sketches* to summarize the values of a field, and have made a novel use of these summaries for data cleaning and structural data mining purposes.

Signatures and sketches have several attractive properties (e.g., summability, ability to detect subsets) which make them an appropriate building block for data mining queries on the structure of a large and complex database. Many such queries are possible, we propose three and give algorithms for evaluating them.

The profiles we collect of the database should be small, for efficient storage and fast query response time, but large enough to give accurate answers. We evaluated the tradeoff between summary size and accuracy for set signatures, multiset signatures, q-gram signatures, and q-gram vector sketches. We found that fifty samples per summarized field was sufficient to accurately determine intersection size, q-gram resemblance, and q-gram vector distance. One hundred samples plus the hashes and counts of the ten most frequent values suffice to estimate join sizes accurately.

We have developed Bellman to browse large and complex databases. Bellman profiles the database, caches the results, and provides interactive responses to the user. Users of Bellman have found the ability to discover fields with high resemblance to be extremely useful in the data exploration process.

7. REFERENCES

[1] D. Achlioptas. Database-friendly random projections. In *PODS*, 2001.

[2] N. Alon, P. Gibbons, and M. S. Y. Matias and. Tracking join and self-join sizes in limited storage. In *PODS*, pages 10–20, 1999.

[3] C. Batani, M. Lenzerini, and S. Navathe. A comparative analysis of methodologies for database schema integration. *Computing Surveys*, 18(4):323–364, 1986.

[4] D. Bitton, J. Millman, and S. Torgerson. A feasibility study and performance study of dependency inference. In *Proc. Intl. Conf. Data Engineering*, pages 635–641, 1989.

[5] V. Borkar, K. Deshmukh, and S. Sarawagi. Automatically extracting structure from free text addresses. *Data Engineering Bulletin*, 23(4):27–32, 2000.

[6] A. Broder. On the resemblance and containment of documents. In *IEEE Compression and Complexity of Sequences '97*, pages 21–29, 1997.

[7] Z. Chen, F. Korn, N. Koudas, and S. Muthukrishnan. Selectivity estimation for boolean queries. In *PODS*, pages 216–225, 2000.

[8] C. Clifton, E. Housman, and A. Rosenthal. Experience with a combined approach to attribute-matching across heterogeneous databases. In *IEEE Workshop on Data Semantics*, 1997.

[9] E. Cohen. Size-estimation framework with applications to transitive closure and reachability. *JCSS*, 55(3):441–453, 1997.

[10] A. Doan, P. Domingos, and A. Halevy. Reconciling schemas of disparate data sources: A machine-learning approach. In *Proc. SIGMOD Conf.*, pages 509–520, 2001.

[2]We have encountered many such examples.

[11] A. Doan, P. Domingos, and A. Levy. Learning source description for data integration. In *Proc. Intl. Workshop on The Web and Databases*, 2000.

[12] Evoke Software. http://www.evokesoftware.com/.

[13] Evoke Software. Data profiling and mapping, the essential first step in data migration and integration projects. http://www.evokesoftware.com/pdf/wtpprDPM.pdf, 2000.

[14] H. Galhardas, D. Florescu, D. Shasha, and E. Simon. AJAX: An extensible data cleaning tool. In *Proc. SIGMOD Conf.*, page 590, 2000.

[15] S. Ganguly, P. Gibbons, Y. Matias, and A. Silberschatz. Bifocal sampling for skew-resistant join size estimation. In *SIGMOD*, pages 271–281, 1996.

[16] M. Garofalakis and P. Gibbons. Approximate query processing: Taming the terabytes. VLDB 2001 tutorial.

[17] P. Gibbons. Distinct sampling for highly-accurate answers to distinct values queries and event reports. In *VLDB*, pages 541–550, 2001.

[18] P. Gibbons and Y. Matias. Synopsis data structures for massive data sets. In *Symp. on Discrete Algorithms*, pages 909–910, 1999.

[19] A. Gilbert, Y. Kotidis, S. Muthukrishnan, and M. Strauss. Surfing wavelets on streams: One-pass summaries for approximate aggregate queries. In *VLDB*, pages 79–88, 2001.

[20] L. Gravano, P. Ipeirotis, H. Jagadish, N. Koudas, S. Muthukrishnan, and D. Srivastava. Approximate string joins in a database (almost) for free. In *Proc. Intl. Conf. VLDB*, 2001.

[21] M. Hernandez, R. Miller, and L. Haas. Schema mapping as query discovery. In *Proc. SIGMOD Conf.*, page 607, 2001.

[22] M. Hernandez and S. Stolfo. Real-world data is dirty: data cleansing and the merge/purge problem. *Data Mining and Knowledge Discovery*, 2(1):9–37, 1998.

[23] Y. Huhtala, J. Karkkainen, P. Porkka, and H. Toivonen. Efficient discovery of functional dependencies and approximate dependencies using partitions. In *Proc. IEEE Intl. Conf. on Data Engineering*, pages 392–401, 1998.

[24] P. Indyk, N. Koudas, and S. Muthukrishnan. Trends in massive time series data sets using sketches. In *VLDB*, pages 363–372, 2000.

[25] Y. Ioannidis and V. Poosala. Histogram-based solutions to diverse database estimation problems. *Data Engineering Bulletin*, 18(3):10–18, 1995.

[26] W. Johnson and J. Lindenstrauss. Extensions of lipschitz mappings into a hilbert space. In *Conference in Modern Analysis and Probability*, pages 189–206, 1984.

[27] V. Kashyap and A. Seth. Semantic and schematic similarities between database objects: A context-based approach. *VLDB Journal*, 5(4):276–304, 1996.

[28] Knowledge Driver. http://www.knowledgedriver.com/.

[29] D. Knuth. *The Art of Computer Programming Vol. 3, Sorting and Searching.* Addison Wesley, 1973.

[30] S. Kramer and B. Pfahringer. Efficient search of string parital determinations. In *Proc. Intl. Conf. on Knowledge Discover and Data Mining*, pages 371–378, 1996.

[31] S. Kuchin. Oracle, odbc and db2-cli template library programmer's guide. http://www.geocities.com/skuchin/otl/home.htm.

[32] M. Lee, H. Lu, T. Ling, and Y. Ko. Cleansing data for mining and warehousing. In *Proc. 10th DEXA*, 1999.

[33] W. Li and S. Clifton. SEMINT: A tool for identifying attribute correspondances in hetergenous datbases using neural networks. *Data and Knowledge Engineering*, 33(1):49–84, 2000.

[34] Metagenix Inc. http://www.metagenix.com/home.asp.

[35] R. Miller, L. Haas, and M. Hernandez. Schema mapping as query discovery. In *Proc. 26th VLDB Conf.*, pages 77–88, 2000.

[36] R. Miller, L. Haas, and M. Hernandez. Schema mapping as query discovery. In *Proc. Intl. Conf. VLDB*, pages 77–88, 2000.

[37] A. Monge. The field matching problem: Algorithms and applications. *IEEE Data Engineering Bulletin*, 23(4):14–20, 2000.

[38] A. Monge and P. Elkan. The field matching problem: Algorithms and applications. In *Proc. Intl. Conf. Knowledge Discovery and Data Mining*, 1996.

[39] A. Monge and P. Elkan. An efficient domain-independent algorithm for detecting approximately duplicate database records. In *Proc. SIGMOD Workshop on Research Issues in Data Mining and Knowledge Discovery*, 1997.

[40] S. Muthukrishnan. Estimating number of rare values on data streams, 2001. Submitted for publication.

[41] C. Parent and S. Spaccepietra. Issues and approaches of database integration. *Comm. of the ACM*, 41(5):166–178, 1998.

[42] E. Rahm and H. Do. Data cleaning: Problems and current approaches. *IEEE Data Engineering Bulletin*, 23(4):1–11, 2000.

[43] V. Raman and J. Hellerstein. Potters wheel: An interactive data cleaning system. In *Proc. VLDB*, 2001.

[44] I. Savnik and P. Flach. Bottom-up induction of functional dependencies from relations. In *Proc. AAAI Knowledge Discovery in Databases*, pages 174–185, 1993.

[45] J. Schlimmer. Efficiently inducing determinations: A complete and systematic search algorithm that uses pruning. In *Proc. AAAI Knowledge Discovery in Databases*, pages 284–290, 1993.

[46] P. Vassiliadias, Z. Vagena, S. Skiadopoulos, N. Karayannidis, and T. Sellis. ARKTOS: A tool for data cleaning and transformation in data warehouse environments. *Data Engineering Bulletin*, 23(4):43–48, 2000.