

Searching with Numbers

Rakesh Agrawal, *Fellow, IEEE*, and Ramakrishnan Srikant

Abstract—A large fraction of the useful web is comprised of specification documents that largely consist of (attribute name, numeric value) pairs embedded in text. Examples include product information, classified advertisements, resumes, etc. The approach taken in the past to search these documents by first establishing correspondences between values and their names has achieved limited success because of the difficulty of extracting this information from free text. We propose a new approach that does not require this correspondence to be accurately established. Provided the data has “low reflectivity”, we can do effective search even if the values in the data have not been assigned attribute names and the user has omitted attribute names in the query. We give algorithms and indexing structures for implementing the search. We also show how hints (i.e., imprecise, partial correspondences) from automatic data extraction techniques can be incorporated into our approach for better accuracy on high reflectivity data sets. Finally, we validate our approach by showing that we get high precision in our answers on real data sets from a variety of domains.

Index Terms—Specification documents, search engines, queries, numbers, hints, data extraction, heterogeneous databases.

1 INTRODUCTION

NUMBERS play a central role in modern life. Yet the current search engines treat numbers as strings, ignoring their numeric values. For example, as of this writing, the search for 6798.32 on Google yielded two pages that correctly associate this number with the lunar nutation cycle [18]. However, the search for 6798.320 on Google found no page. The search for 6798.320 on AltaVista, AOL, HotBot, Lycos, MSN, Netscape, Overture, and Yahoo! also did not find any page about the lunar nutation cycle.

A large fraction of the useful web is comprised of what can be called *specification documents*. They largely consist of attribute-value pairs surrounded with text. Examples of such documents include product information, classified advertisements, resumes, etc. For instance, Fig. 1 shows a partial extract of the data sheet for the Cypress CY7C225A PROM. A design engineer should be able to ask a query that looks something like this:

address set-up speed 20 ns power 500 mW CMOS PROM

and get the CY7C225A data sheet. None of the search engines could find this datasheet using the above query (or its variations). We were able to get the data sheet when we provided the exact numeric values for the speed and power attributes because the search engines could then match the string. It is unreasonable to expect that the user will provide exact numeric values when doing such searches. In fact, users typically search for items whose specifications roughly match the values provided in the query.

The approach taken in the past to retrieving the specification documents has been to extract the attribute-value pairs contained in a document and store them in a database. Queries can now be answered using nearest

neighbor techniques [1], [15], [21]. There has been some research on automating the task of data extraction (see surveys in [7], [19]). However, the automatic extraction of attribute-value pairs has a spotty record of success. It is a hard problem, exacerbated by the fact that it is often difficult to identify attribute names and establish correspondence between an attribute name and its value. Very often, different documents refer to the same attribute by different names. We experienced first-hand this problem in building an experimental portal for electronic components, called Pangea. For semiconductor components alone, we obtained more than 200,000 datasheets from nearly 1,000 manufacturers. Our effort to extract parametric information from these datasheets met with only limited success. It is noteworthy that the major content companies in electronics industry employ a large number of people who manually extract the parametric information.

This paper proposes a new approach to retrieving specification documents. In essence, we are suggesting that it is not necessary to establish exact correspondences between attribute names and numeric values in the data. A user query can instead choose to provide only values, without specifying corresponding attribute names. For this approach to work, data must have what we call *low reflectivity*. This property, introduced in this paper, is exhibited by many real world data sets and the extent to which a data set satisfies this property can be computed a priori. For a simple example of a nonreflective data set, assume that the documents contain only two attributes: “memory” and “disk-size.” Further assume that the range of values for memory is 64 to 512 and that for disk-size is 10 to 40. Given a query {20, 128}, the system can correctly retrieve documents that have disk-size and memory values close to 20 and 128, respectively. In this example, the attributes have nonoverlapping domains. However, low reflectivity is not limited to data having such nonoverlapping attributes. If memory and disk-size overlapped, but were correlated such that high memory configurations had high disk-size, the data would still have low reflectivity.

• The authors are with IBM Almaden Research Center, 650 Harry Road, San Jose, CA 95120. E-mail: {ragrawal, srikant}@us.ibm.com.

Manuscript received 15 July, 2002; revised 19 Jan., 2003; accepted 10 Mar., 2003.

For information on obtaining reprints of this article, please send e-mail to: tkde@computer.org, and reference IEEECS Log Number 118463.

-
- CMOS for optimum speed/power
 - High speed
 - 18 ns address set-up
 - 12 ns clock to output
 - Low power
 - 495 mW (commercial)
 - 660 mW (military)
-

Fig. 1. Specifications for Cypress CY7C225A PROM (www.cypress.com/cypress/prodgate/prom/cy7c225a.html)

The target repositories for our techniques are document collections on focused topics. Classification and clustering techniques can often be applied to partition a general repository into a set of topic-specific repositories. Our techniques can also be applied in other applications where providing attribute names in a query is difficult or inconvenient. For example, in a federated database system, the same attribute might be called with different names in different constituent databases [8], [16], [17].

The techniques we propose complement current search technology. One can envision a system in which the ranked results produced using our techniques are combined with the results obtained using words and links in the pages [2], [3], [9], [22]. Techniques such as [10] can be used for combining the results.

The rest of the paper is organized as follows: Section 2 provides the data and query models. We discuss reflectivity in Section 3. We present algorithms for finding matching documents without hints in Sections 4 and with hints in 5. Section 6 gives experimental results showing the accuracy and performance characteristics of our techniques. We conclude with a summary and directions for future work in Section 7.

2 MODEL

We assume that a document is preprocessed to extract numbers appearing in it. For each number, the potential attribute names are optionally extracted by examining the text surrounding the number.¹ This extraction need not be precise. One may associate zero or multiple attribute names with a number depending on the confidence on the quality of extraction.

To simplify exposition, we will initially ignore units normally associated with numeric values. In Section 5.2, we discuss the handling of units within our framework.

At the time of querying, the user may simply provide numbers or, optionally, specify attribute names with numbers. The attribute names provided in a query may not always correspond to the names present in the data since the same attribute may be called with multiple names, e.g., salary, income, pay, etc. We consider nearest neighbor queries where the user is interested in retrieving the top t documents containing values close to the query terms.

1. The specific extraction techniques used are not of concern in this paper. One could, for example, use a regular expression-based extractor [5].

2.1 Database and Queries

Let \mathcal{N} be the universe of numbers and \mathcal{A} the universe of attribute names. The database \mathcal{D} consists of a set of documents. A document $D \in \mathcal{D}$ is defined to be

$$D = \{\langle n_i, H_i \rangle \mid n_i \in \mathcal{N}, H_i \subseteq \mathcal{A}, 1 \leq i \leq m\}, \quad (1)$$

where H_i is the set of attribute names (hints) associated with the number n_i and m is the number of unique (number, attribute names) pairs present in D . Note that it is possible to have $n_i = n_j$ but $H_i \neq H_j$ since more than one attribute name can have the same numeric value. Duplicates ($n_i = n_j$ and $H_i = H_j$) may optionally be admitted by treating D as a multiset. However, we assume that a document does not contain multiple different values for an attribute, i.e., if $H_i = H_j$, then $n_i = n_j$.

If the document does not have hints associated with numbers, D is simply a multiset:

$$D = \{n_i \mid n_i \in \mathcal{N}, 1 \leq i \leq m\}. \quad (2)$$

A search query Q consists of

$$Q = \{\langle q_i, A_i \rangle \mid q_i \in \mathcal{N}, A_i \subseteq \mathcal{A}, 1 \leq i \leq k\}. \quad (3)$$

Here, each (number, attribute names) pair represents a query term and k is the number of query terms present in Q . A_i is the set of attribute names associated with the number q_i . We allow the user to associate a set of attribute names with a number since there may be synonyms. There is an implied conjunction between the query terms. It is possible to have $q_i = q_j$ but $A_i \neq A_j$, i.e., a query may contain more than one occurrence of the same number.

If the query does not have hints associated with the numbers, Q is simply a multiset:

$$Q = \{q_i \mid q_i \in \mathcal{N}, 1 \leq i \leq k\}. \quad (4)$$

2.2 Matching without Attribute Names

The goal for any search is to return documents that are most similar to the query, ordered by their similarity score. The real problem lies in defining similarity. The generally accepted approach is to use some L_p norm as the measure of distance and take similarity to be inverse of this distance.

Consider a query system where the attribute names are not available in both the data and in the query. For a query $Q = \{q_1, \dots, q_k\}$, there must be a true attribute that the user has in mind corresponding to each number q_i in Q . Similarly, for a document $D = \{n_1, \dots, n_m\}$, each number n_i in D has a corresponding true attribute.

We call a document D a *true close match* to a query Q if D contains a set of k numbers $D' = \{n_{j_1}, \dots, n_{j_k}\}$ such that the distance between Q and D' is small (i.e., less than some constant r) and both q_i and n_{j_i} are the values of the same true attribute. We have treated both Q and D as ordered sets in this definition.

We call a document D a *nameless close match* to a query Q if D contains a set of k numbers $D' = \{n_{j_1}, \dots, n_{j_k}\}$ such that the distance between Q and D' is small. Unlike the case of a true close match, the true attributes corresponding to n_{j_i} and q_i need not be the same. Notice that, by requiring a set of k numbers in D , we impose the desired constraint that a

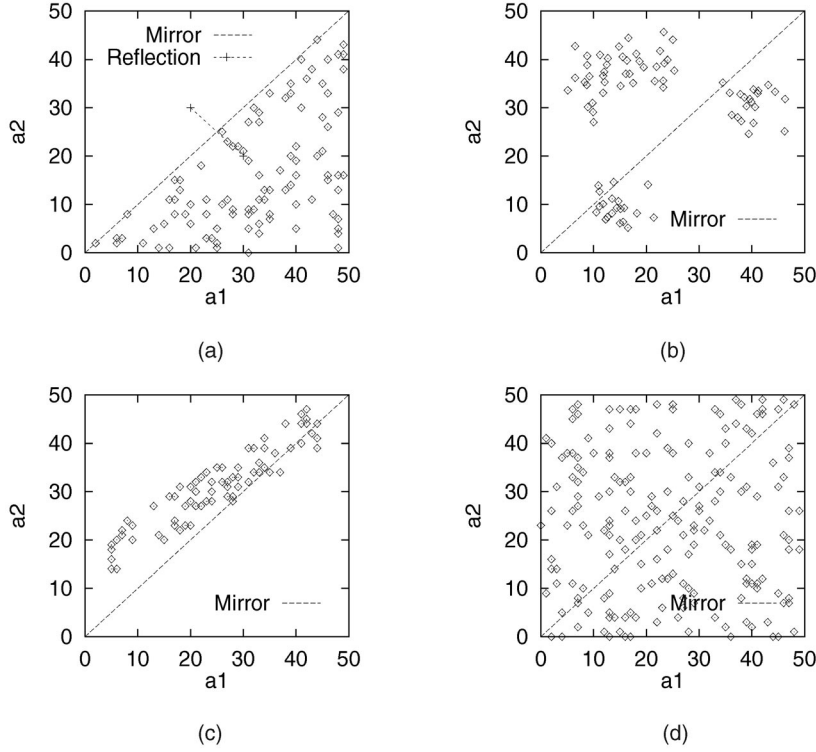


Fig. 2. Examples of nonreflective and reflective data (a) Nonreflective, (b) Low reflectivity, (c) Low reflectivity, (d) High reflectivity.

number instance in the document should not match multiple query terms.

Conjecture 1 (Realistic Query). *For most queries asked by users, there exists a document $D \in \mathcal{D}$ that is a true close match to the query.*

Conjecture 2 (Implicit Name Match). *In domains where the Realistic Query conjecture holds, if a document D is a nameless close match to a query Q , it is likely that that D is also a true close match to Q .*

Informally, if we get a good match on the numbers between a document and query, then it is likely that we will have correctly matched the attributes as well. Thus, for data sets where this conjecture holds, we can match using only the numbers and still get good accuracy compared to the benchmark where we know the true attributes in both the documents and the query.

In Section 3, we define a property called *reflectivity* that allows us to quantify the extent to which this conjecture holds in a given data set. In fact, if the query distribution for some set of true attributes is the same as the data distribution projected onto that subspace, the likelihood in the above conjecture is the same as the value of nonreflectivity.

3 REFLECTIVITY

Consider the data shown in Fig. 2a for two attributes a_1 and a_2 . The diamonds indicate the points actually present in the data. This data is completely nonreflective: For any point $\langle a_1 = n_i, a_2 = n_j \rangle$ present in the data, its reflection $\langle a_1 = n_j, a_2 = n_i \rangle$ does not exist. Fig. 2b gives an example of clustered data that is almost completely nonreflective. The

correlated data in Fig. 2c is also largely nonreflective. However, the data in Fig. 2d is highly reflective.

The implication of reflectivity is that the queries against low reflectivity data can be answered accurately without knowing the attribute names, provided the realistic query conjecture is true. Hence, although there is complete overlap between the range of values of the two attributes in Figs. 2a, 2b, and 2c, we will get high accuracy on any two-dimensional query. For example, consider the query $\langle 20, 30 \rangle$ on the data in Fig. 2a. This query can only map to $\langle a_1 = 30, a_2 = 20 \rangle$ since there are no points in the region around $\langle a_1 = 20, a_2 = 30 \rangle$.

Queries will often span a subset of the dimensions in the data set and reflectivity will depend on the exact set of dimensions being considered. Consider the query $\langle 20 \rangle$ on the data in Fig. 2a. This query can map to either $\langle a_1 = 20 \rangle$ or $\langle a_2 = 20 \rangle$ and the answer to this query will consist of points for which either a_1 or a_2 is close to 20. Thus, precision on this query will be around 50 percent, in contrast to the close to 100 percent precision that we can get on the query $\langle 20, 30 \rangle$ for the same data set. Similar behavior is exhibited by data in Fig. 2b and 2c: They are highly nonreflective in two dimensions, but quite reflective in either of the one-dimensional projections.

Before formally defining reflectivity, we make the following observations.

- Above, we took a given query Q and checked whether or not the reflections of Q coincided with other data points. However, for similarity queries, we care not only about the exact query values, but points close to the query values. Hence, we should

look at the number of points within distance r of each reflection of Q .

- Rather than taking a query and considering whether there are points close to the reflections of the query, we take a dual viewpoint. For a given query Q , we consider how many reflections of other points are within distance r of Q and compare this number with the number of points that are truly within distance r of Q .

3.1 Definition

We first define reflectivity under the assumption that there are no duplicates, i.e., a document contains the value of an attribute only once. We extend the definition to handle duplicates in Section 3.5.

Let \mathcal{D} be a set of m -dimensional points of cardinality $|\mathcal{D}|$. Let \vec{n}_i denote the coordinates of point x_i . We first define reflectivity over the full m -dimensional space and then give a more general definition for subspaces.

We define the *reflections* of the point x_i to be the set of coordinates obtained by permuting \vec{n}_i (including \vec{n}_i). For example, if x_i were $\langle 1, 2 \rangle$, the reflections of x_i would be $\{\langle 1, 2 \rangle, \langle 2, 1 \rangle\}$.

Let $\theta(\vec{n}_i)$ denote the number of points within distance r of \vec{n}_i (in m -dimensional space). The value of r is so chosen that the average value of $\theta(\vec{n}_i)$ (over all $x_i \in \mathcal{D}$) is close to the number of top answers that users will be interested in. Let $\rho(\vec{n}_i)$ denote the number of points in \mathcal{D} that have at least one reflection within distance r of \vec{n}_i . The reflectivity of \mathcal{D} in m -dimensional space is then defined to be:

$$\text{Reflectivity}(m, r) = 1 - \frac{1}{|\mathcal{D}|} \sum_{x_i \in \mathcal{D}} \frac{\theta(\vec{n}_i)}{\rho(\vec{n}_i)}. \quad (5)$$

Now, consider a k -dimensional subspace S of the space of \mathcal{D} . We define the k -reflections of a point x_i in \mathcal{D} to be the set of coordinates obtained by considering the $k!$ permutations of ${}^m C_k$ combinations of k coordinates chosen from \vec{n}_i . For example, the two-reflections of a three-dimensional point $\langle 1, 2, 3 \rangle$ will be the set $\{\langle 1, 2 \rangle, \langle 2, 1 \rangle, \langle 2, 3 \rangle, \langle 3, 2 \rangle, \langle 1, 3 \rangle, \langle 3, 1 \rangle\}$.

Let \vec{n}_i^S represent the coordinates of point x_i projected onto this subspace. Let $\theta(S, \vec{n}_i^S)$ denote the number of points in \mathcal{D} whose projections onto the subspace S are within distance r of the coordinates \vec{n}_i^S (in the k -dimensional space). As before, the value of r is so chosen that the average value of $\theta(S, \vec{n}_i^S)$ (over all $x_i \in \mathcal{D}$) is close to the number of desired top answers. Let $\rho(S, \vec{n}_i^S)$ denote the number of points in \mathcal{D} that have at least one k -reflection within distance r of the coordinates \vec{n}_i^S (in the k -dimensional space). The reflectivity of the subspace S is defined to be:

$$\text{Reflectivity}(S, r) = 1 - \frac{1}{|\mathcal{D}|} \sum_{x_i \in \mathcal{D}} \frac{\theta(S, \vec{n}_i^S)}{\rho(S, \vec{n}_i^S)}. \quad (6)$$

Finally, let $\hat{\mathcal{S}}_k$ represent the set of k -dimensional subspaces of \mathcal{D} . Let $|\hat{\mathcal{S}}_k| = {}^m C_k$ denote the number of k -dimensional subspaces. Then, the reflectivity of \mathcal{D} over k -dimensional subspaces is defined to be the average of the reflectivity in each subspace:

$$\text{Reflectivity}(k, r) = \frac{1}{|\hat{\mathcal{S}}_k|} \sum_{S \in \hat{\mathcal{S}}_k} \text{Reflectivity}(S, r). \quad (7)$$

Note that:

$$\text{Non-reflectivity} = 1 - \text{Reflectivity}. \quad (8)$$

3.2 Implicit Name Match Conjecture Revisited

Let S be the subspace corresponding to the attribute in a query $Q = \{q_1, \dots, q_k\}$. Let \vec{Q} denote the coordinates corresponding to $\{q_1, \dots, q_k\}$. Then, there are $\theta(S, \vec{Q})$ documents that are true close matches to Q and $\rho(S, \vec{Q})$ documents that are nameless close matches to Q . Hence, for this query, the probability that a document that is a nameless close match will also be a true close match is simply $\theta(S, \vec{Q})/\rho(S, \vec{Q})$.

Let us represent a query distribution for a subspace S by a random sample of queries $\mathcal{Q} = \{Q_1, \dots, Q_N\}$ drawn from that distribution. Then, for a query belonging to this distribution, the probability that a document that is a nameless close match will also be a true close match is

$$\frac{1}{|\mathcal{Q}|} \sum_{Q_i \in \mathcal{Q}} \frac{\theta(S, \vec{Q}_i)}{\rho(S, \vec{Q}_i)}.$$

Finally, if the query distribution for a subspace S is close to the distribution of documents projected onto S , we can treat the set of points as a sample of the query distribution and the probability that a document that is a nameless close match will also be a true close match is simply $\text{nonreflectivity}(S, r)$. Thus reflectivity can serve as a close proxy for expected accuracy.

3.3 Remarks

1. *Nonoverlapping Attributes:* If the attributes of a data set do not overlap, such data is necessarily nonreflective for queries of any length.
2. *Clustering and Correlation:* For a fixed amount of overlap between attributes, clustered and/or correlated data sets are likely to have lower reflectivity than data sets where the attributes are independent. Fig. 2b and 2c support this intuition. In Section 6.4, we study this effect for nine real-world data sets by computing reflectivity both for the original data set and for a modified data set where we destroy any correlation or clustering while keeping the actual values of each attribute fixed. Destroying correlation and clustering increases reflectivity in all nine data sets, often dramatically. Of course, it is easy to come up with counterexamples of correlated or clustered data sets that are quite reflective, as shown in Fig. 3.

3.4 Computing Reflectivity

If the database knows the correct attribute names corresponding to data values, we can use (7) to compute reflectivity. Null values in the data can be handled as follows: Suppose the values of some of the coordinates of an m -dimensional point x_i are null (unknown). We ignore this point in the computation of reflectivity in (5). When computing the reflectivity of a subspace S in (6), the term $\theta(S, \vec{n}_i^S)/\rho(S, \vec{n}_i^S)$ for point x_i is excluded from the summa-

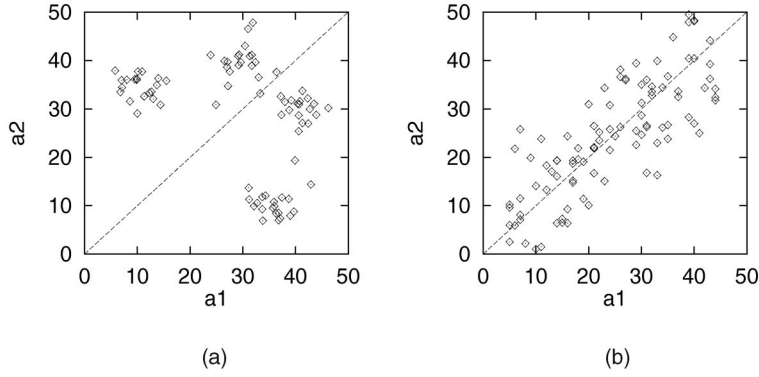


Fig. 3. Counterexamples showing clustered and correlated data sets that are reflective. (a) Clustered & reflective, (b) Correlated & reflective.

tion if a null value is present in the set of coordinates projected onto S . However, x_i may still contribute to the denominator of the above term for some other point x_j if a k -reflection of x_i does not have any null coordinate values and this reflection is within distance r of \vec{n}_j^s .

The cost of computing reflectivity can be reduced by doing the summation in (6) for a sample of data points. Similarly, we can do summation over a sample of subspaces in (7).

Consider now the scenario where attribute names have not been assigned to most of the values in the documents. If we can get hints for the attribute name, we can treat the highest-ranked hint for each number as the attribute name and compute reflectivity. We empirically evaluate this idea in Section 6.7. Our results show that this idea tends to be useful if the accuracy of hints is relatively high.

Finally, consider the situation in which we do not even have good hints. The proposed techniques may still work well; we just will not be able to estimate accuracy a priori. If the answers displayed to the user show sufficient summary information that the user's selections are a reliable indicator of the accuracy of the answer, we can use the precision of the answers as a rough estimate of reflectivity.

3.5 Duplicates

In longer documents such as product specifications, the value of an attribute may be repeated, e.g., in a summary as well as in the main body of the document. We now extend our definition of reflectivity to handle such duplicates. In the presence of duplicates, we might not be able to distinguish whether two identical values in a document belong to the same attribute or to two different attributes. Thus, reflections of a point now include not just permutations of coordinates, but also additional reflections arising out of a duplicate value appearing in multiple coordinate positions.

As before, let \mathcal{D} be a set of m -dimensional points and let \vec{n}_i denote the coordinates of point x_i . Let $\Delta(x_i)$ denote the multiset constructed as follows: If the j th value in x_i occurs p times in the document corresponding to x_i , then we add p instances of the j th value to $\Delta(x_i)$. For example, if $x_i = \langle 1, 2, 3 \rangle$ and the first two coordinates occur twice in the document corresponding to x_i , $\Delta(x_i) = \{1, 1, 2, 2, 3\}$.

Consider a k -dimensional subspace S of the space of \mathcal{D} . We now define the k -reflections of a point x_i in \mathcal{D} to be the set of coordinates obtained by considering the $k!$ permutations of $|\Delta(x_i)|C_k$ combinations of k values chosen

from $\Delta(x_i)$. For example, if $\Delta(x_i) = \{1, 1, 2, 2, 3\}$, the two-reflections of x_i would be the set $\{\langle 1, 2 \rangle, \langle 2, 1 \rangle, \langle 2, 3 \rangle, \langle 3, 2 \rangle, \langle 1, 3 \rangle, \langle 3, 1 \rangle, \langle 1, 1 \rangle, \langle 2, 2 \rangle\}$. In this example, the last two reflections are the additional reflections due to the presence of duplicates.

As before, let \vec{n}_i^s represent the coordinates of point x_i projected onto this subspace. Let $\theta(S, \vec{n}_i^s)$ denote the number of points in \mathcal{D} whose projections onto the subspace S are within distance r of the coordinates \vec{n}_i^s (in the k -dimensional space). Let $\rho(S, \vec{n}_i^s)$ denote the number of points in \mathcal{D} that have at least one k -reflection within distance r of the coordinates \vec{n}_i^s (in the k -dimensional space). The reflectivity of the subspace S is defined to be:

$$\text{Reflectivity}(S, r) = 1 - \frac{1}{|\mathcal{D}|} \sum_{x_i \in \mathcal{D}} \frac{\theta(S, \vec{n}_i^s)}{\rho(S, \vec{n}_i^s)}. \quad (9)$$

The other definitions follow directly from the above equation. For instance,

$$\text{Reflectivity}(k, r) = \frac{1}{|\hat{\mathcal{S}}_k|} \sum_{S \in \hat{\mathcal{S}}_k} \text{Reflectivity}(S, r). \quad (10)$$

When computing reflectivity, the only difference is that we use the modified definition of reflections that incorporates duplicates.

The algorithms we describe in the next section work as such on documents with duplicate values.

4 ALGORITHMS

We now give algorithms for finding documents in response to a user query. These algorithms assume that the database, as well as queries, has no attribute names associated with the numbers. Section 5 discusses how to take advantage of this extra information.

Recall that a document D consists of $D = \{n_i \mid n_i \in \mathcal{N}, 1 \leq i \leq m\}$ (2). A search query Q consists of $Q = \{q_i \mid q_i \in \mathcal{N}, 1 \leq i \leq k\}$ (4). Note that both D and Q are multisets. Each value corresponds to an unspecified attribute name.

In computing the distance of query Q from a document D , each q value is matched with exactly one n value. Given a set of query numbers q_1, \dots, q_k and a set of matching document numbers n_{j_1}, \dots, n_{j_k} , the distance function F with the L_p norm ($1 \leq p \leq \infty$) is defined as

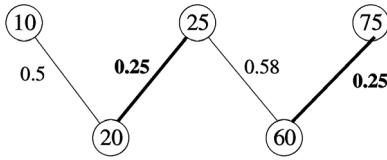


Fig. 4. Bipartite graph

$$F(Q, D) = \left(\sum_{i=1}^k w(q_i, n_{j_i})^p \right)^{1/p}, \quad (11)$$

where $w(q_i, n_{j_i})$ is the distance between q_i and n_{j_i} . We expect that F will typically use relative distance since, otherwise, some of query terms will get disproportionate importance and other query terms will be ignored. For example, $w(q_i, n_{j_i})$ may be defined as $|q_i - n_{j_i}|/|q_i + \epsilon|$.

Maximizing similarity is equivalent to minimizing distance.

4.1 Matching a Document to a Query

Given a set $Q = \{q_1, \dots, q_k\}$ of k numbers and a set $D = \{n_1, \dots, n_m\}$ of m document numbers, we want to select the numbers in D that will lead to the minimum distance. Each number in D is allowed to match with a single number in Q and vice versa.

Construct a weighted bipartite graph G as follows:

- Create k source vertices labeled q_1, \dots, q_k corresponding to k numbers in Q .
- Create m target vertices labeled n_1, \dots, n_m corresponding to m numbers in D . If $m < k$, add $k - m$ target vertices with value ∞ .
- From each source vertex q_i , create an edge to the k closest target vertices in $\{n_1, \dots, n_m\}$.² Assign weight $w(q_i, n_j)^p$ to the edge (q_i, n_j) .

Fig. 4 shows the weighted bipartite graph for $Q = \{20, 60\}$ and $D = \{10, 25, 75\}$, assuming the distance function to be L_1 and $w(q_i, n_j) = |q_i - n_j|/|q_i + \epsilon|$.

Lemma. *The optimum solution to the minimum weight bipartite graph matching problem for the graph G matches each number in Q with a distinct number in D such that we get the lowest value for the distance score $F(Q, D)$.*

We have marked in bold the edges comprising the optimum solution for the graph in Fig. 4. Thus, 20 in Q is matched with 25 in D and 60 with 75 for a total distance score of 0.5.

We can now refer to the rich weighted bipartite graph matching literature (see the survey in [4]) to find the best matching between the numbers in a query and the numbers in a document. We also obtain the distance score at the same time, which is used for ranking the documents. By repeating this process for every document in the database, we have a solution to our problem. In Section 4.2, we present techniques that avoid examining every document.

The best known algorithm for the weighted bipartite matching problem is due to Feder and Motwani [13] and its time complexity is $O(e\sqrt{(k+m)}\log((k+m)^2/e)/\log(k+m))$,

2. For a given source vertex, we only need to create edges to the k closest targets since the other $k - 1$ source vertices can match at most $k - 1$ targets.

where e is the number of edges in the graph. Since $e = k^2$, the complexity is $O(k^2\sqrt{(k+m)}\log((k+m)/k)/\log(k+m))$.

4.2 Limiting the Set of Documents that are Matched

We now address the question of how to limit the number of documents for which we have to compute the distance. This problem turns out to be similar to that of retrieving the top t objects that have highest combined score on k attributes, introduced in [11]. We first describe the score aggregation problem and the threshold algorithm for solving this problem [12], [14], [20].

Score Aggregation Problem. Assume that each object in a database has k scores, one for each of k attributes. For each attribute, there is a sorted list, which lists each object and its score for that attribute, sorted by score (highest score first). There is some monotone aggregation function f for combining the individual scores to obtain an overall score for an object. The problem is to efficiently find the top t objects that have the best overall score.

Threshold Algorithm (TA). There are two modes of access to data. Sorted access obtains the score of an object in one of the sorted lists by proceeding through the list sequentially from the top. Random access obtains the score of an object in a list in one access. The threshold algorithm works as follows [12]:

1. Do sorted access in parallel to each of the k sorted lists L_i . In other words, access the top member of each of the lists under sorted access, then the second member, and so on. As an object D is seen in some list, do random access to other lists to find score s_i of object D in every list L_i . Then, compute the overall score $f(D) = f(s_1, \dots, s_k)$ of object D . If this score is one of the t highest we have seen, then remember D and its score $f(D)$.
2. For each list L_i , let s_i be the score of the last object seen under sorted access. Define the threshold value τ to be $f(s_1, \dots, s_k)$. Halt when t objects have been seen whose overall score is at least equal to τ .
3. Let Y be a set containing the t objects that have been seen with the highest scores. The result is the graded set $\{\langle D, f(D) \rangle \mid D \in Y\}$.

Proposed Adaptation. We now discuss how our problem is similar to the score aggregation problem. We then show how the threshold algorithm can be adapted to our problem.

Assume that the documents have been preprocessed to create data structures to support the following types of accesses.

- **Database Access:** Given a document id, return the multiset of numbers present in the document.
- **Index Access:** Given a number, return the set of documents in which this number is present. Only numbers that appear in at least one document are included in this index. Numbers are kept sorted so that it is easy to determine the nearest left neighbor (smaller number) and nearest right neighbor (larger number) of a number. We can use B-tree [6] for this purpose if the index is too large to fit in memory.

Here is the algorithm, stated in the TA framework. While reading the algorithm, keep in mind that a document with a lower distance score is closer to the query and, hence, better in our setting.

1. Form k conceptual lists, one for each query term q_i as follows: For every q_i , create an ordered list of numbers n_i^1, n_i^2, \dots such that $w(q_i, n_i^j) \leq w(q_i, n_i^{j+1})$. (Recall that $w(q_i, n_i^j)$ is the distance between q_i and n_i^j .) Associate the score $s_i^j = w(q_i, n_i^j)$ with every document returned by index access on n_i^j . The list L_i for q_i is now defined to consist of documents obtained from index look up on terms n_i^1, n_i^2, \dots sorted in ascending value of score (lowest score first). Note that these lists are not physically materialized, but the `next()` operation on these lists is well-defined and can be efficiently implemented using the index access described above.
2. Do a round-robin access to each of the k sorted lists L_i . As a document D is seen in some list, do a database access for this document and match it with the query using the algorithm from Section 4.1. The distance score returned by the matching algorithm gives the overall score of the document.
3. Let n'_i be the number in the index that we last looked at for query term q_i . Define the threshold value τ to be the distance $(\sum_{i=1}^k w(q_i, n'_i)^p)^{1/p}$ from (11). Halt when t documents have been seen whose distance from Q is less than or equal to τ . At this point, for any document that has not been seen in the index, the closest number to each query term q_i must be at least as far from q_i as n'_i and, hence, the distance between the document and the query must be at least as high as τ .

Note that, unlike the original threshold algorithm, the s_i scores in the adaptation are lower bounds on the distance, not necessarily the actual distance. In other words, when we match a document D to a query, the number that ends up being matched with q_i may be further away from q_i than indicated by the score for D in the sorted list for q_i . The reason is that, during matching, a number in D can only match one query term, but we do not track this constraint during index access (to avoid the bookkeeping overhead). Thus, if a single number in D is the closest number to two different query terms, one of the two scores for D will be a lower bound for the actual distance. This does not affect the correctness of the halting criteria in Step 3 since the threshold value τ is a lower bound on the distance of a document that we have not yet seen and we stop when we have seen t documents whose distance is lower than τ .

5 USING ATTRIBUTE NAMES AND UNITS

5.1 Attribute Names

We now describe how to use hints about attribute names to aid matching. Let H_i denote the set of attribute names associated with the number n_i in a document. As before, let A_i denote the set of attribute names associated with q_i in a query. We extend the distance function from (11) to incorporate hints as follows:

$$F(Q, D) = \left(\sum_{i=1}^k (w(q_i, n_i)^p + B \times v(A_i, H_i)^p) \right)^{1/p}. \quad (12)$$

The parameter B balances the importance between the match on the numbers and the match on the hints. In general, the higher the accuracy of the hints and the higher the reflectivity of the data, the higher should be the value of B .

Recall that the function $w(q_i, n_j)$ determines the distance between a query number and a document number. Analogously, $v(A_i, H_j)$ is a function that determines the distance between the set of attribute names associated with a query number and the set of attribute names associated with a document number. We use the following distance function v in our experiments:

$$v(A_i, H_j) = \begin{cases} 0 & \text{if } A_i \cap H_j \neq \phi \\ 0 & \text{if } A_i = \phi \\ 1 & \text{otherwise.} \end{cases} \quad (13)$$

This function penalizes a match only if q_i and n_j are likely to belong to different attributes. Hence, if any of the attribute names in the query matches any of the hints or if there is no attribute name specified in the query, the distance is zero. Otherwise, the distance is 1.

Our techniques are independent of the specific form of the function v . A more sophisticated function form tuned to the specific data extractor being used (e.g., by incorporating belief values generated by the data extractor) may yield better results. However, as we will see, our experiments indicate that even this simple function can be quite effective.

Determining the weight B . A good value for B is important for successfully using hints. Suppose the Website provides enough summary with each answer that the user is likely to click on relevant answers. By tracking user clicks, we can get a set of queries for which we know the true answers with high probability. Treat this set as a tune set and evaluate the performance of the matching algorithm for different values of B . Then, pick the value that gives the highest accuracy. If a set of values give similar accuracy, pick the median value in the set. Note that the best value of B is likely to be different for different query sizes and, hence, we will need a tune set per query size.

Constructing the Bipartite Graph. As before, we map the matching problem to weighted bipartite graph matching. The only difference is that we now assign $w(q_i, n_j)^p + B \times v(A_i, H_j)^p$ as the weight of the edge (q_i, n_j) .

Limiting the Set of Matches. The algorithm proposed in Section 4.2 can be used as is since we only need the s_i^j score to be a lower bound on the true distance. That is, we can use $s_i^j = w(q_i, n_i^j)$ to create the sorted list L_i for q_i , ignoring the match on attribute names.

However, the algorithm will be considerably more efficient if we modify it as follows: First, create an index to support an additional type of access, *hint index access*: Given a number n_i together with an attribute name a_i , return the set of documents in which n_i is present and the set of hints for n_i includes a_i . The original index can be (conceptually) treated as a special case of this index, where the attribute name $a_i = \phi$.

Now, in Step 1, for each query term q_i , create an ordered list $\langle n_i^1, a_i^1 \rangle, \langle n_i^2, a_i^2 \rangle, \dots$ and associate the score $s_i^j = w(q_i, n_i^j)^p + B \times v(A_i, \{a_i^j\})^p$ with the entry $\langle n_i^j, a_i^j \rangle$, such that $s_i^j \leq s_i^{j+1}$. We can do this efficiently by using hint index access for each attribute name in the set of hints A_i associated with the query term q_i and also for the empty attribute name ϕ . In Step 3, the only change is that we now define the threshold value τ to be $(\sum_{i=1}^k (w(q_i, n_i^j)^p + B \times v(A_i, \{a_i^j\})^p))^{1/p}$ from (12), where $\langle n_i^j, a_i^j \rangle$ is the entry in the index that we last looked at for query term q_i .

5.2 Units

If the unit names are available in addition to attribute names, we extend the distance function from (12) by adding a term $B^u \times u(A_i^u, H_{j_i}^u)^p$ for whether the units match:

$$F(Q, D) = \left(\sum_{i=1}^k (w(q_i, n_{j_i})^p + B \times v(A_i, H_{j_i})^p + B^u \times u(A_i^u, H_{j_i}^u)^p) \right)^{1/p}. \quad (14)$$

A_i^u denotes the set of unit names for q_i , $H_{j_i}^u$ is the set of unit names for n_{j_i} , and $u(A_i^u, H_{j_i}^u)$ is the distance function between the two sets of unit names. The function $u(A_i^u, H_{j_i}^u)$ is defined in a manner similar to $v(A_i, H_{j_i})$. The parameter B^u is analogous to B and can be similarly computed.

An additional complication arises due to different units (e.g., MHz and GHz) being assigned to the values of the same attribute. In some cases, this problem can be approached by converting the corresponding quantities into a “standard” form.

6 EXPERIMENTS

In this section, we report the results of experiments we performed to study the accuracy and performance characteristics of the proposed techniques. We used synthetic as well as real data sets in this study. Synthetic data sets are amenable to controlled experiments, while real data sets are used to check that the results are indeed applicable in the real world.

6.1 Accuracy Metric

Given a database \mathcal{D} in which the correct correspondences between the numbers and the attributes are known, a query Q consisting of numbers and full information about the attributes corresponding to those numbers, we could generate the “perfect” answer of top t matching documents $M_P(t) = \{D_{p_1}, \dots, D_{p_t}\}$ for a given distance function F . If the information about the attribute is not available or only partially available in the form of hints, we would generate a different set of top t matching documents $M(t) = \{D_1, \dots, D_t\}$.

Precision(t) is defined to be the percentage of documents in $M_P(t)$ that are also present in $M(t)$:

$$\text{Precision}(t) = \frac{|M_P(t) \cap M(t)|}{|M(t)|} \times 100. \quad (15)$$

We set t to 10 in our experiments, corresponding to the number of hits typically displayed on the first page by search engines, and refer to Precision(10) as simply Precision.

6.2 Data Sets

6.2.1 Synthetic Data

A document is modeled as a set of m numbers, corresponding to m known attributes. We generated three types of synthetic data:

1. *Independent*: Each attribute is independent of the other attributes.
2. *Correlated*: Attribute a_{i+1} is correlated with attribute a_i .
3. *Clustered*: Data is clustered around a few cluster centers.

We expect Independent to be more reflective and Clustered and Correlated to be less reflective. Therefore, we should get higher precision on Clustered and Correlated and lower precision on Independent.

Fig. 5 shows the pseudocode for generating the three types of synthetic data. We use $s[]$ to control the amount of reflectivity. The value of $s[j]$ is initially set to $R \times j$, where R is a parameter that controls to the amount of overlap between the range of values of various attributes, which in turn influences reflectivity. If we wanted to generate a data set with almost no overlap between any of the attributes, R would be set to a high value, e.g., 10. On the other hand, if we wanted very high overlap between the attributes, R would be set to 0. In our experiments, we manually tuned R such that the one-dimensional nonreflectivity is roughly 80 percent for each of the three data sets (with the default number of attributes). The values in $s[]$ are then randomly permuted so that, for the correlated data set, there is no connection between which attributes are correlated and which attributes are adjacent in terms of values.

Fig. 6 shows the settings used in the synthetic data experiments. The Range column gives the range of values used in various experiments. The Default column provides the default value of the corresponding parameter. Query size refers to the number of terms in the query.

To generate a query of size k , we first pick a document at random. Next, for Independent and Clustered, we randomly pick k attributes from the document and, for Correlated, we randomly pick a set of k consecutive attributes. We then remove this document from the data set and compute the precision. (If we kept the document in the data set, our scores would be artificially boosted.) We average the results over 1,000 queries for each combination of parameters and query size. We use L_1 to compute distance between a query and a document in all our experiments.

6.2.2 Real Data

Fig. 7 gives information about the nine real data sets used in our experiments. The first four data sets came from the Pangea data on electronic components for different categories of electronic parts. The last five data sets are from the University of California-Irvine Repository of Machine Learning. A record here corresponds to a document.


```

//  $N$ : number of documents
//  $m$ : number of attributes
//  $D_i$ :  $i^{\text{th}}$  document
//  $D_i(j)$ : value of the attribute  $j$  in document  $D_i$ 
//  $R$ : controls the amount of overlap
//  $s[\cdot]$ :  $s[j]$  set to  $R \times j$  and then  $s[\cdot]$  is permuted
// Gauss(): Gaussian with  $\mu = 0$  and  $\sigma = 1$ 

```

Independent:

- 1) for $i := 1$ to N
- 2) for $j := 1$ to m
- 3) $D_i(j) := \text{Gauss}() + s[j]$

Correlated:

- 1) for $i := 1$ to N
- 2) $D_i(1) := \text{Gauss}()$
- 3) for $j := 2$ to m
- 4) $D_i(j) := (D_i(j-1) + \text{Gauss}()) \times 0.7$
- 5) for $j := 1$ to m begin
- 6) $D_i(j) := D_i(j) + s[j]$

Clustered:

- ```

// C : number of clusters
1) for $i := 1$ to C
2) for $j := 1$ to m
3) $D_i(j) := \text{Gauss}() + s[j]$
4) for $i := C+1$ to N
5) $\text{cid} := (i \bmod C) + 1$
6) for $j := 1$ to m
7) $D_i(j) := D_{\text{cid}}(j) + 0.2 \times \text{Gauss}()$

```

Fig. 5. Synthetic data generation.

The query generation procedure used with the real data is identical to the procedure we described for the Independent and Clustered data. All results are the average of 1,000 queries.

**6.3 Relationship of Precision to Reflectivity**

Fig. 8a shows the precision on the three synthetic data sets as the query size is increased. We compute answers to the same query under two settings: (1) attribute names are known for both data and query and (2) attribute names are ignored in data as well as query. We then use (15) to compute precision. Fig. 8b plots nonreflectivity versus subspace dimensionality. Nonreflectivity is computed assuming attribute names are known in the data. Both precision and nonreflectivity have been plotted as a percentage. These figures show that precision at a given query size closely tracks nonreflectivity for the corresponding subspace dimensionality.

As anticipated, both Clustered and Correlated gave higher precision than Independent. Clustered gave higher precision than Correlated for higher query dimensions. To understand the behavior of Clustered with increasing subspace dimensionality, recall our example from Fig. 2b. This data set has very low reflectivity in two dimensions, but high reflectivity in one dimension. Similarly, in our synthetic data, the clusters gradually merge as we drop dimensions, leading to an increase in reflectivity for lower dimensions. By the time we reach two dimensions, the

| Parameter                    | Default | Range               |
|------------------------------|---------|---------------------|
| Number of Attributes ( $m$ ) | 20      | 5 to 50             |
| Number of Documents ( $N$ )  | 10K     | 1K to 800K          |
| Query Size ( $k$ )           | 5       | 1 to 10             |
| 1-dim. non-reflectivity      | 80%     | 100% to 10%         |
| Data Type                    |         | Ind., Corr., Clust. |

Fig. 6. Parameters for synthetic data experiments.

clusters have completely disappeared and Clustered starts behaving like Correlated and Independent.

Fig. 9a shows the precision as a function of the number of documents. Fig. 10a shows the precision as a function of the number of data attributes. We again note from corresponding (b) figures that the precision closely tracks nonreflectivity.

**6.4 Precision on Real Data**

Fig. 11 shows the precision and nonreflectivity as a percentage in the same graph for the nine real data sets. The x-axis for precision and nonreflectivity is query size and subspace dimensionality respectively. We again see that nonreflectivity can provide a good estimate of the expected precision. We also see that real data sets can be quite nonreflective and hence our techniques can be effectively used on them.

**Effect of Clustering and Correlation.** In Section 3, we stated our intuition that clustering and correlation would typically increase nonreflectivity. To verify this conjecture, we study the effect of destroying any clustering or correlation effects in a data set by randomly permuting the values for every attribute. (We permute the values of an attribute across documents, not across different attributes.) If the attributes were originally independent, this permutation will not affect reflectivity, since permutation does not affect the distribution of values for each attribute. Hence the difference in nonreflectivity between the original data and the randomized data can be attributed to clustering and correlation. The line “Randomized Non-Refl.” (in Fig. 11) shows the nonreflectivity on the randomized data sets. For all but the Credit data set, randomized nonreflectivity is substantially lower than nonreflectivity.

**6.5 Duplicates**

We explore the effect of duplicates on reflectivity (Section 3.5) by creating two new versions of each real data set. The “1

| Source | Dataset        | Number of Records | Maximum Number of Attributes | Average Number of Attributes |
|--------|----------------|-------------------|------------------------------|------------------------------|
| Panga  | DRAM           | 3,866             | 10                           | 6.8                          |
|        | LCD            | 1,733             | 12                           | 9.4                          |
|        | Microprocessor | 1,133             | 12                           | 5.3                          |
|        | Transistor     | 22,273            | 24                           | 13.3                         |
| UCI    | Automobile     | 205               | 16                           | 15.7                         |
|        | Housing        | 506               | 14                           | 14                           |
|        | Glass          | 214               | 10                           | 10                           |
|        | Wine           | 179               | 14                           | 14                           |
|        | Credit         | 666               | 6                            | 6                            |

Fig. 7. Real data sets.

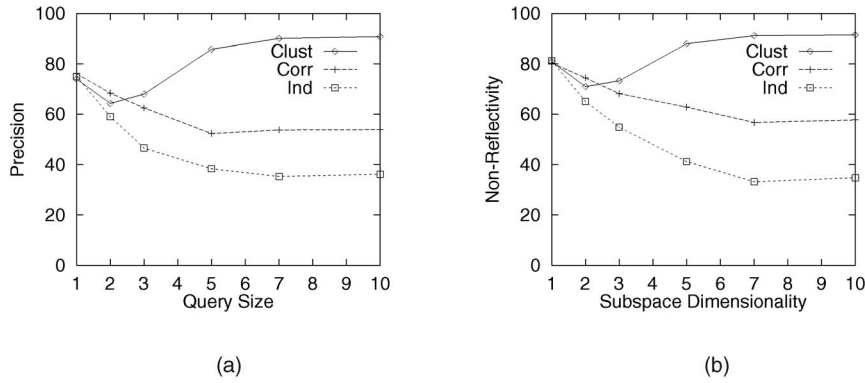


Fig. 8. Precision and nonreflectivity versus query size (#documents = 10K, #attributes = 20). (a) Precision (b) Non-reflectivity.

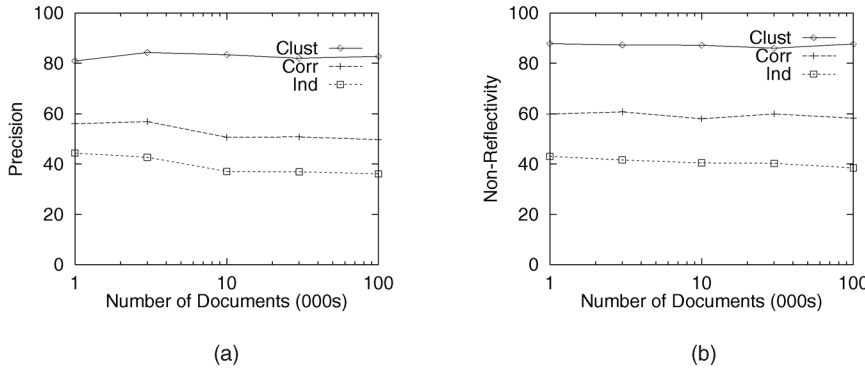


Fig. 9. Precision and nonreflectivity versus number of documents (query size = 5, #attributes = 20). (a) Precision, (b) Non-reflectivity.

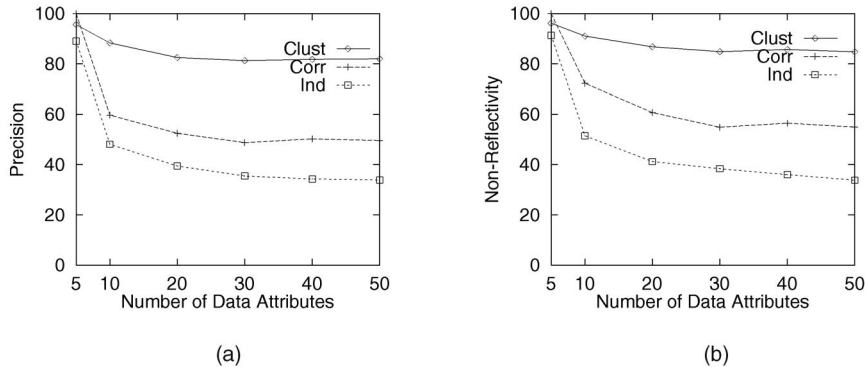


Fig. 10. Precision and nonreflectivity versus number of attributes (query size = 5, #attributes = 10K). (a) Precision, (b) Non-reflectivity.

Dup" version has each value in a document repeated once. The "5 Dup" version has each value repeated five times. We show results for three data sets: DRAM, Auto and Wine. DRAM and Auto have high nonreflectivity, and Wine has lower nonreflectivity.

Figs. 12 and 13 show the effect of duplicates on nonreflectivity and precision, respectively. The precision again follows the nonreflectivity. When the subspace dimensionality (or, equivalently, query size) is small, duplicates have almost no impact on nonreflectivity. Even for larger subspace dimensionalities of 5 to 10, the impact is still quite moderate.

3. We can also do this analysis in terms of reflectivity, and partition points based on whether different coordinate values are equal (after projecting to the subspace). We have given the analysis in terms of queries since the latter is easier to explain.

To understand why duplicates have so little impact, we partitioned the queries into two types: (1) those in which all the query values are unique and (2) those in which the same value occurs multiple times (corresponding to different attributes).<sup>3</sup> Fig. 14a shows that, when the DRAM data does not have duplicates, these two query types give almost the same precision. However, in the version of DRAM with duplicates, the queries in which a value does not recur maintain almost the same precision, while those with recurring values show a large drop in precision, as shown in Fig. 14b. Thus, most of the loss in precision comes from those queries that have recurring values (or, more generally, that have two or more values close to each other). The reason is that the additional reflections created by the presence of duplicates have the property that they have two or more identical coordinates (recall the example in Section 3.5). These reflections will often match queries with recurring values

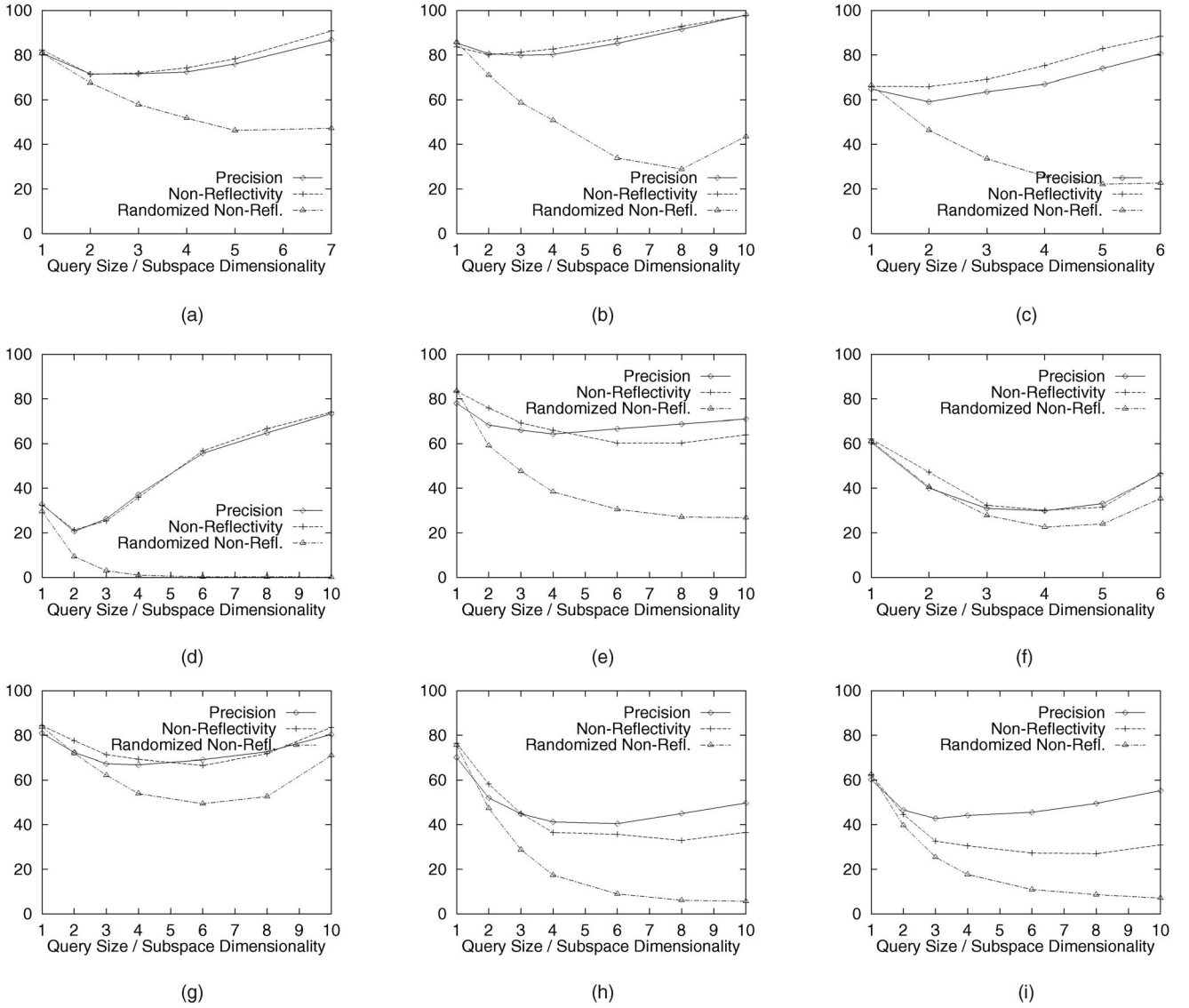
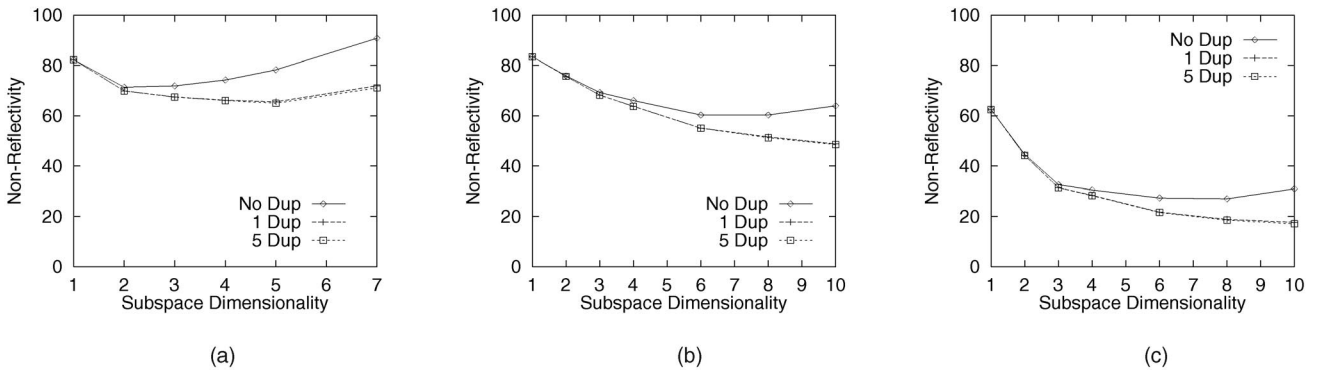


Fig. 11. Precision on real data sets. (a) DRAM, (b) LCD, (c) Microprocessor, (d) Transistor, (e) Automobile, (f) Credit, (g) Glass, (h) Housing, (i)



causing loss in precision, but rarely match queries that do not have two close values.

It is worth noting in Fig. 14b that, for queries with recurring values the precision increased with increasing query size. It is because the amount of information in the remaining query values, makes it more likely that the

correct document will be matched. One might now wonder as to why we get a larger drop in precision for larger query sizes in Fig. 13a. The reason is that, in the query mix, the fraction of queries with recurring values goes up with an increase in query size. For DRAM, the fraction increased

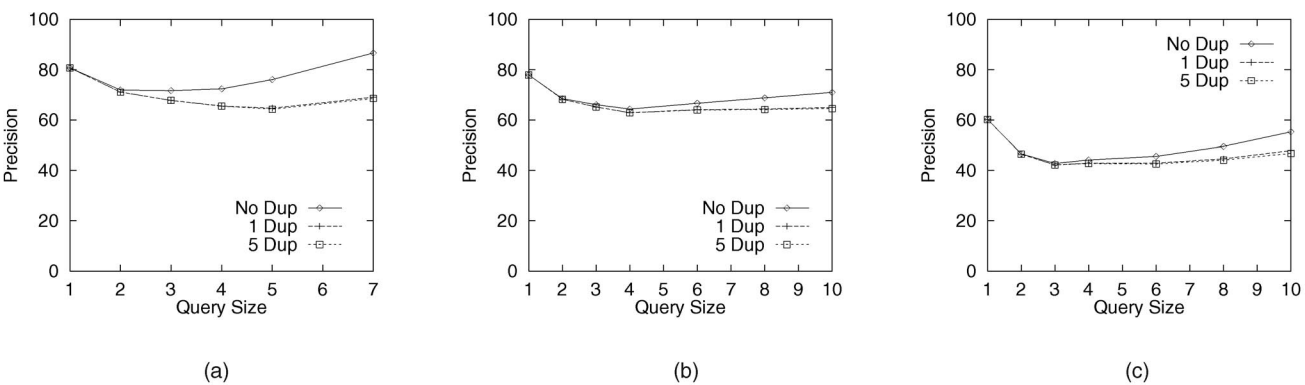
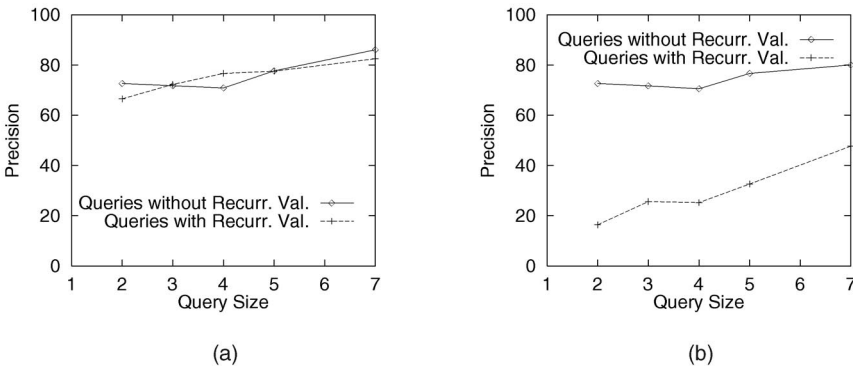


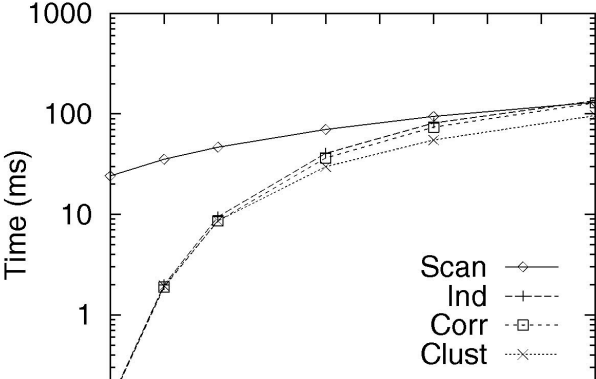
Fig. 13. Effect of duplicates on precision. (a) DRAM, (b) Auto, (c) Wine.



from around 3 percent at query size 2 to around 40 percent at query size 10. Thus, although individual queries are making less error, there are more queries finding more errors as the query size is increased. Hence, the overall loss in precision increases.

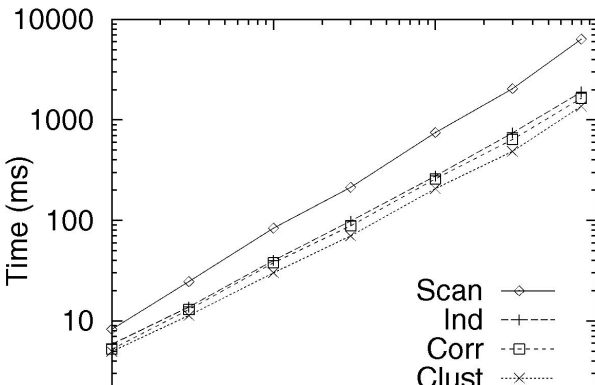
6.6 E

We now consider a set of distance queries on the 933 MB dataset. The server was w/



relative impact of various factors is likely to be the same. Both index and documents were kept completely in DRAM, limiting us to around 800,000 documents. Fig. 15 shows the execution times with and without

Fig. 15. (a) No duplicates, (b) with duplicates.



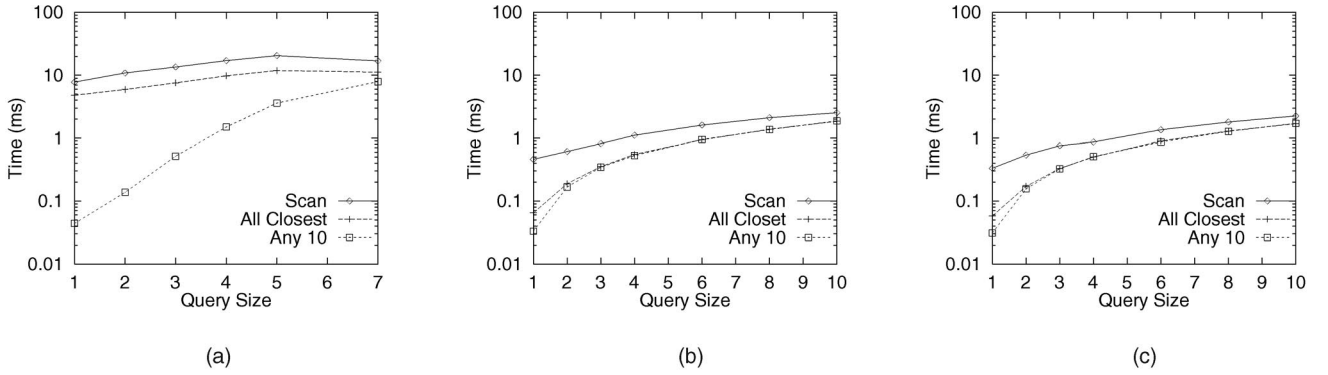


Fig. 16. Effectiveness of indexing: real data. (a) DRAM, (b) Auto, (c) Wine.

**Scalability** Fig. 15b shows the scalability of the indexing as we increase the number of documents from 1,000 to almost one million. At query size five, the number of documents matched goes up by a factor of about 250 times as we increase the number of documents by 800 times and the number of index entries scanned goes up around 200 times. Hence, while the fraction of the index scanned and the percentage of documents checked both decrease slightly, the absolute time goes up almost linearly with the number of documents. At 800,000 documents, we take slightly more than one second for query size five and around 0.03 seconds for query size two.

Fig. 15c shows that the execution time remains flat as we increase the number of attributes. In our synthetic data sets, adding new attributes does not change the average amount of overlap between attributes. Hence, the number of documents matched or index entries scanned is not affected by adding more attributes. However, matching takes a little longer and, hence, the overall time goes up slightly.

**Real Data Sets.** Fig. 16 shows results on three of the real data sets. For DRAM, there were often a large number of documents at a distance of zero from the query  $Q$  for smaller query sizes. Hence, whether we stop as soon as we get 10 documents with zero distance or whether we get all documents at zero distance (and then present a random sample) makes a dramatic difference in the running time. The “All Closest” line shows the former case and the “Any 10” line the latter. For the other two data sets, this is not a significant issue and, hence, these two lines are very close to each other. Again, we can conclude that indexing is quite effective for smaller query sizes.

## 6.7 Using Attribute Names as Hints

We generated data for this set of experiments by starting with the real data sets and then augmenting them with hints. Our procedure for generating hints takes two parameters: AvgNumHints and ProbTrue. AvgNumHints is the average number of hints per data value. Each data value is assigned at least one hint, hence the minimum value of AvgNumHints is 1. If AvgNumHints is greater than 1, then the number of hints for a specific data value is determined using a Poisson distribution. ProbTrue is the probability that the set of hints for a data value includes the true attribute name. Fig. 17 gives the pseudocode for generating hints for a given data value.

In general, increasing AvgNumHints should result in increasing ProbTrue. However, since the exact relationship is data and data extractor dependent, in any given experiment, we fix one of the values and vary the other. The query consists of numbers together with the corresponding true attribute name.

We synthesized nine data sets from the real data sets described earlier and experimented with all of them. However, we show the graphs for only three representative data sets: DRAM, Auto, and Wine. DRAM and Auto have high nonreflectivity and Wine lower nonreflectivity.

**Weighting the Match on Hints.** Fig. 18 shows the effect of changing the weight given to the match on the attribute names (the parameter  $B$  in (12)). AvgNumHints was set to 1 so that every data value had exactly one hint assigned to it. The different curves show precision for different values of ProbTrue. If the hints are accurate (high value of ProbTrue), then precision rapidly goes up with increasing  $B$ . For DRAM, the precision goes up for values of  $B$  much smaller than 0.01 and, hence, Fig. 18a does not show this increase. For queries on DRAM, there are often different attributes with the same numeric value as the query value and even a small value of  $B$  is sufficient to pick between them. If the hints are not very accurate, precision goes down with increasing  $B$ . One should not conclude that  $B$  should be  $\infty$  if hints are accurate and 0 otherwise. For example, in the case of wine with ProbTrue between 0.6 and 0.8, the precision initially goes up with  $B$  and then drops back down. Therefore, it is important to choose a good value of  $B$ .

In the rest of the experiments, we assume we have a tune set of 100 queries (per query size) for which we know the best match. We compute Precision over this tune set for

```
// ProbTrue: Prob. of true attribute name being in the set of hints
// Rand(0,1): Uniform Random number between 0 and 1
n := 1 + Poisson(AvgNumHints-1)
 \mathcal{A}_f := \mathcal{A} - true attribute name
if (Rand(0,1) ≤ ProbTrue)
 Output true attribute name and $n-1$ names randomly
 selected from \mathcal{A}_f
else
 Output n attribute names randomly selected from \mathcal{A}_f
```

Fig. 17. Generating hints.

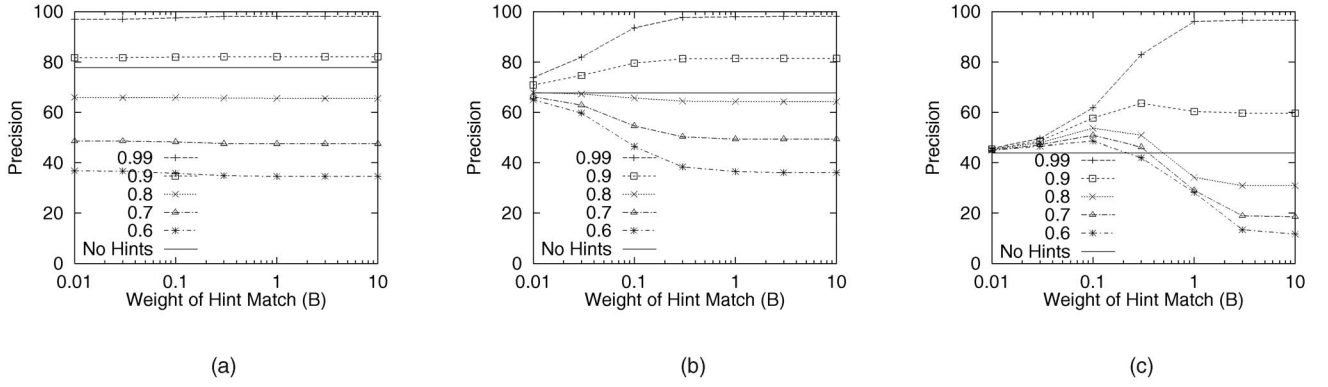


Fig. 18. Effect of balance between number match and hint match. (a) DRAM (query size 2), (b) Auto (query size 2), (c) Wine (query size 5)

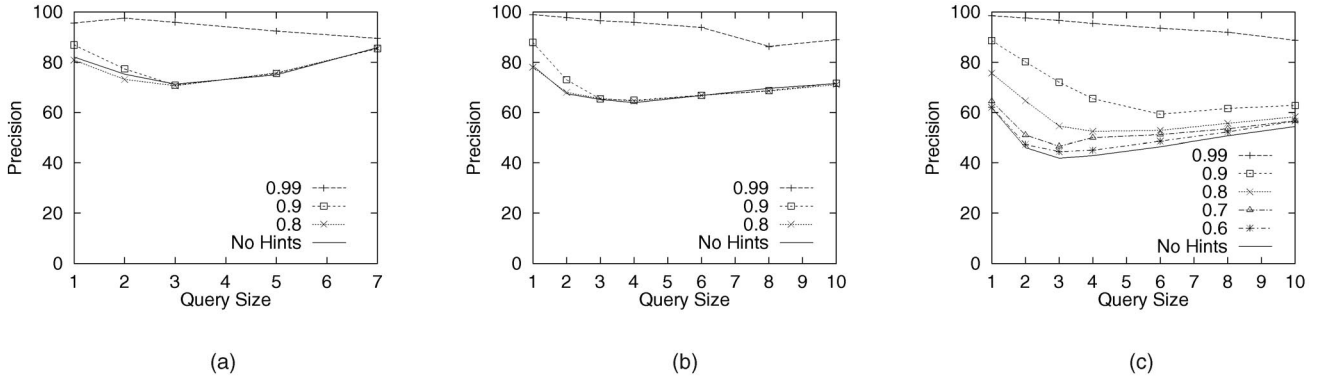


Fig. 19. Effectiveness of hints. (a) DRAM, (b) Auto, (c) Wine

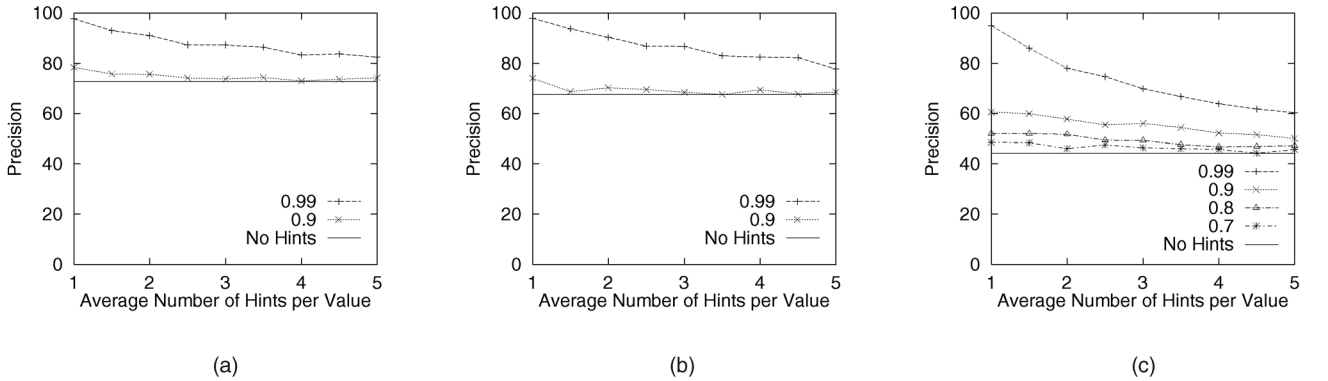


Fig. 20. Number of hints. (a) DRAM (query size 2), (b) Auto (query size 2), (c) Wine (query size 5).

different values of  $B$  (specifically, we used  $\{0.01, 0.03, 0.1, 0.3, 1, 3, 10\}$ ). We then choose the value that gives the highest precision. As we mentioned in Section 5, this method can be used as long as the Website provides enough summary with each answer so that the user is likely to click on relevant answers.

**Effectiveness of Hints.** Fig. 19 shows the gains in precision for different values of ProbTrue when AvgNum Hints is set to 1. Notice that, for data sets with high nonreflectivity (DRAM and Auto), the hints have to be extremely accurate (ProbTrue  $> 0.9$ ) to add significant value. However, for data sets with low nonreflectivity (Wine), even hints with lower levels of accuracy can increase precision.

**Number of Hints.** In the previous set of experiments, we found that ProbTrue (the probability that the set of hints will include the true attribute name) had to be quite high to increase precision. In many domains, such high values of ProbTrue may not be achievable without having multiple hints per attribute, where only one of the hints is correct. Fig. 20 shows precision for different values of the average number of hints (AvgNumHints). For each line, we have fixed the value of ProbTrue.

As expected, for a fixed value of ProbTrue, precision decreases as the number of hints increases. However, we expect ProbTrue to increase with the number of hints. Consider Fig. 20c, and assume we have a data extractor that gives ProbTrue of 0.7 when AvgNumHints = 1, ProbTrue of 0.9 when AvgNumHints = 1.5, and ProbTrue of 0.99 when

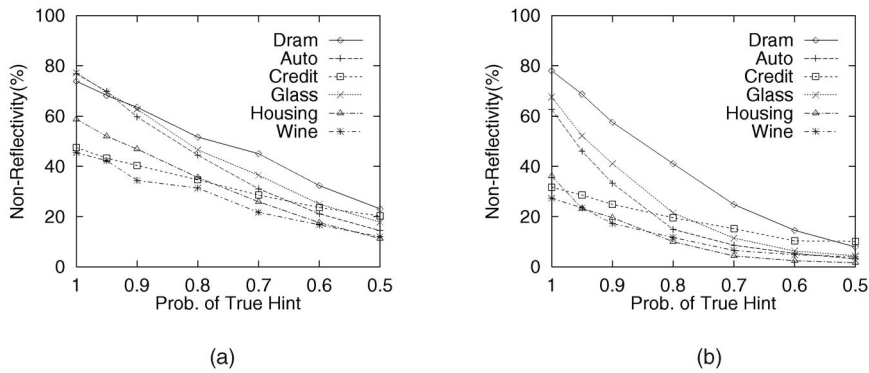


Fig. 21. Estimating reflectivity using hints. (a) Subspace Dim. 2, (b) Subspace Dim. 5.

$\text{AvgNumHints} \geq 3$ . In this scenario, choosing  $\text{AvgNumHints} = 3$  will result in the highest precision. Thus, having multiple hints can improve precision if there is an accompanying increase in  $\text{ProbTrue}$ .

**Estimating Reflectivity Using Hints.** Finally, we explore how well we can estimate nonreflectivity when we do not know true attributes but only have hints about them. We generate data with  $\text{AvgNumHints} = 1$  and different values of  $\text{ProbTrue}$ , treat each hint as if it were the true attribute name, and compute reflectivity. Fig. 21 shows the results. The values on the y-axis, with  $\text{ProbTrue} = 1$ , are the true values of nonreflectivity. The nonreflectivity estimates are lower with hints than with true attribute names. However, for small subspace dimensions, the drop-off is sufficiently gradual that the estimate is still useful when the hints are reasonably accurate. For larger subspace dimensions, the drop-off is sometimes quite steep. Hence, if the estimate for nonreflectivity has a low value, we cannot be sure that the true value of nonreflectivity is indeed low or whether it is an artifact of the inaccuracy of hints. But, if we get a high value, we can be confident that the true value of nonreflectivity is also high.

## 7 CONCLUSIONS

Many Web text repositories consist of documents in which it is difficult to establish exact correspondences between attribute names and their numeric values. To enable nearest-neighbor queries over such repositories, we explored a rather audacious approach of using only sets of numbers in the search, ignoring any attribute names. We showed that matching a document with a query in this setting corresponds to the weighted bipartite graph matching problem. We also showed how to limit the number of documents that we have to match with a given query.

We identified a data property, called reflectivity, that can tell a priori how well our approach will work against a data set. High nonreflectivity in data assures that our techniques will have high precision. Our experiments showed that real data sets can exhibit high nonreflectivity and our techniques yielded high precision against these data sets. Using synthetic data, we also showed the scalability and resilience of our techniques in terms of the number of documents, number of attributes in the data, size of queries, and types of data.

We also showed how we can use imprecise attribute names as hints to improve precision. Hints are particularly helpful when the data has low nonreflectivity. However, for data sets with high nonreflectivity, hints have to be extremely accurate to improve precision beyond what we get by matching numbers.

In the future, we plan to extend this work in three directions. First, we would like to handle queries in which value ranges are provided. Second, some extraction procedures associate confidence values with the attribute names assigned to a number. We wish to explore how to take advantage of this additional information. Finally, we would like to better understand the interaction between correlated and clustered data and reflectivity.

## ACKNOWLEDGEMENT

The authors would like to thank Ron Fagin for discussions on the threshold algorithm. A preliminary version of this paper appeared in the *Proceedings of the 2002 World Wide Web Conference*, Honolulu, Hawaii, May 2002.

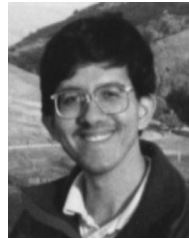
## REFERENCES

- [1] S. Berchtold, C. Bohm, D.A. Keim, F. Krebs, and H.-P. Kriegel, "On Optimizing Nearest Neighbor Queries in High-Dimensional Data Spaces," *Proc. Eighth Int'l Conf. Database Theory*, Jan. 2001.
- [2] A. Broder and M. Henzinger, "Information Retrieval on the Web: Tools and Algorithmic Issues," *Foundations of Computer Science* invited tutorial, 1998.
- [3] S. Chakrabarti, "Data Mining for Hypertext: A Tutorial Survey," *SIGKDD Explorations*, 1, 2000.
- [4] B. Cherkassky, A. Goldberg, P. Martin, J. Setubal, and J. Stolfi, "Augment or Push? A Computational Study of Bipartite Matching and Unit Capacity Flow Algorithms," Technical Report 97-127, NEC Research Inst., 1997.
- [5] J. Cho and S. Rajagopalan, "A Fast Regular Expression Indexing Engine," *Proc. Int'l Conf. Data Eng.*, Feb. 2002.
- [6] D. Comer, "The Ubiquitous B-tree," *ACM Computing, Surveys*, vol. 11, no. 2, pp. 121-138, June 1979.
- [7] A. Crespo, J. Jannink, E. Neuhold, M. Rys, and R. Studer, "A Survey of Semi-Automatic Extraction and Transformation," <http://www-db.stanford.edu/crespo/publications/>, 1994.
- [8] U. Dayal and H.-Y. Hwang, "View Definition and Generalization for Database Integration in a Multidatabase System," *IEEE Trans. Software Eng.*, vol. 10, no. 6, pp. 628-645, 1984.
- [9] S. Deerwester, S. Dumais, G. Furnas, T. Landauer, and R. Harshman, "Indexing by Latent Semantic Analysis," *J. Am. Soc. Information Science*, vol. 41, no. 6, pp. 391-407, 1990.

- [10] C. Dwork, M. Naor, R. Kumar, and D. Sivakumar, "Rank Aggregation Methods for the Web," *Proc. 10th Int'l World Wide Web Conf.*, May 2001.
- [11] R. Fagin, "Combining Fuzzy Information from Multiple Systems (extended abstract)," *Proc. Symp. Principles of Database Systems*, 1996.
- [12] R. Fagin, A. Lotem, and M. Naor, "Optimal Aggregation Algorithms for Middleware," *Proc. Symp. Principles of Database Systems*, 2001.
- [13] T. Feder and R. Motwani, "Clique Partitions, Graph Compression and Speeding-Up Algorithms," *J. Computer and System Sciences*, vol. 51, pp. 261-272, 1995.
- [14] U. Guntzer, W.-T. Balke, and W. Kiessling, "Optimizing Multi-Feature Queries for Image Databases," *Proc. 26th Int'l Conf. Very Large Databases*, 2000.
- [15] P. Indyk and R. Motwani, "Approximate Nearest Neighbors: Towards Removing the Curse of Dimensionality," *Proc. ACM Symp. Theory of Computing*, pp. 604-613, 1998.
- [16] V. Kashyap and A.P. Sheth, "Semantic and Schematic Similarities between Database Objects: A Context-Based Approach," *VLDB J.*, vol. 5, no. 4, pp. 276-304, 1996.
- [17] W. Kim and J. Seo, "Classifying Schematic and Data Heterogeneity in Multidatabase Systems," *Computer*, vol. 24, no. 12, pp. 12-18, Dec. 1991.
- [18] *Physics and Astronomy of the Moon*, Z. Kopal, ed. Academic Press, 1962.
- [19] I. Muslea, "Extraction Patterns for Information Extraction Tasks: A Survey," *Proc. AAAI-99 Workshop Machine Learning for Information Extraction*, 1999.
- [20] S. Nepal and M.V. Ramakrishna, "Query Processing Issues in Image (Multimedia) Databases," *Proc. 15th Int'l Conf. Data Eng.*, 1999.
- [21] N. Roussopoulos, S. Kelley, and F. Vincent, "Nearest Neighbor Queries," *Proc. 1995 ACM SIGMOD Int'l Conf. Management of Data*, pp. 71-79, 1995.
- [22] G. Salton, *Automatic Text Processing: The Transformation, Analysis, and Retrieval of Information by Computer*. New York, Addison-Wesley, 1989.



and applications. He has published more than 100 research papers and he has been granted 45 patents. He is the recipient of the ACM-SIGKDD First Innovation Award as well as the ACM-SIGMOD 2000 Innovations Award. He is a fellow of the IEEE.



He was named an IBM Research Division Master Inventor in 1999 and has also received two Outstanding Technical Achievement Awards for his contributions to Intelligent Miner.

**Rakesh Agrawal** is an IBM fellow whose current research interests include privacy technologies for data systems, Web technologies, data mining, and OLAP. He leads the Quest project at the IBM Almaden Research Center, which pioneered key data mining concepts and technologies. IBM's commercial data mining product, Intelligent Miner, grew out of this work. His technical contributions have influenced several commercial and academic products, prototypes

**Ramakrishnan Srikant** is a research staff member at the IBM Almaden Research Center. His research interests include privacy technologies for data systems, data mining, and Web technologies. He was a key architect for IBM's commercial data mining product, Intelligent Miner. He has published more than 30 research papers and been granted 12 patents. He was the Program co-chair of the 2001 ACM SIGKDD International Conference on Knowledge Discovery

► **For more information on this or any computing topic, please visit our Digital Library at <http://computer.org/publications/dlib>.**