Approximate Matchings in Scientific Databases*

Abhirup Chatterjee and Arie Segev

Walter A. Haas School of Business
University of California at Berkeley

and

Information and Computing Sciences Division
Lawrence Berkeley Laboratory
Berkeley, CA 94720

Abstract

Organizations often need access to scientific data stored in independently managed heterogeneous databases. In this paper, we analyze the data heterogeneity problem which occurs when the data conveying the same or similar information is represented differently in different databases. We introduce the Matching join to process queries in scientific databases and discuss the three steps to evaluate it.

First we transform the query using the functional dependencies in the database to incorporate additional knowledge. Second, we use rules and weights to compare the attributes. Matching joins can also be used to obtain approximate answers. In the third step, we propose a numeric measure, called the comparison value, c, to estimate the quality of matching and suggest deterministic and probabilistic ways of deriving it. Finally, we analyze the problem of estimating the cutoff value for c that would minimize the cost of errors during the join computation.

1 Introduction

Over the last several years, there has been a growing interest in the research community in large scientific databases (SDBs). Available commercial database softwares are designed for the needs of corporate data processing. However, data in SDBs do not fit well into this Supplier-Supply-Parts paradigm. As a result, commercial database management systems (DBMSs) are inadequate to support scientific and statistical applications [10, 17].

Organizations often need access to scientific data stored in heterogeneous and interdisciplinary databases. But the following factors make the access to heterogeneous databases very difficult [1]:

• these databases are designed and implemented in an uncoordinated way by independent organizations such as government agencies and research laboratories,

• there are no common conventions for naming, describing, formatting, representing and structuring data. Data conveying the same information contained in these databases have different logical and physical representation and even different values,

• scientific applications often need to identify records which are similar but not necessarily identical.

The problems associated with the incompatible representation of data in multiple databases are known as the data heterogeneity problems. Some of these problems are not specific to scientific databases, but they occur more frequently in such a setting. Data heterogeneity occurs when the attributes are defined differently in different databases. For example, data type mismatch, different formats, units, granularity, etc. It could also occur when similarly structured attributes take on different semantics and values in different databases. This could be due to the existence of synonyms, homonyms, incomplete information, mismatched codes and inconsistent updates. A more detailed description of the various sources of data heterogeneity can be found in [3].

Our objective is to provide a generalized model to resolve these data heterogeneity problems in scientific databases. In particular, we focus in this paper on the retrieval of the same or similar data items which are represented differently in the same or different databases. We introduce the Matching join for this purpose and discuss the various steps to evaluate it. In this type of join, tuples are compared and joined if they match, where matching can be defined by the user using a set of rules and/or weights. Matching joins can also be used to obtain approximate answers.

The following is the organization of the paper. Section 2 discusses some of the potential applications of the model. In Section 3, the Matching join framework is introduced. The concept of query transformation is
presented in Section 4. In Section 5, the framework is further extended to include rules and band joins. The approximation aspect of the Matching join is discussed in Section 6. The paper is concluded in Section 7 with a summary and directions for future research.

2 Applications of the Model

In this section, we discuss a number of scientific applications which motivated our research in this area. Most of these application areas deal with medium to large data sets managed by special or general purpose database management systems. Traditionally, the data heterogeneity disputes used to be clerically resolved. However, as many of these disputes occur repeatedly, clerical intervention becomes too costly, unreproducible, error-prone and time consuming to be a viable option. We hope to provide a computerized, general purpose technique to resolve the data heterogeneity problems in a uniform way.

2.1 Astronomy

One of the problems in Astronomy is to determine if an observed object, such as a star, has been identified before. Information about stars are stored in catalogues which are essentially relational tables where each row is an observed object and the columns are its properties, such as brightness, size and position [18]. Cross-identification of stars in different catalogues is of great potential value to the researchers. However, a simple natural join of the two tables, which depends on the exact equality of key-values would be meaningless. The exact values of the properties are usually not known, as the observations are made from different locations at different points in time using different instruments. Thus, a star which appears in the catalogue from one observatory would have slightly different properties when measured elsewhere for another catalogue. The problem is further compounded by the large size of the relations. For example, the Guide Star Catalogue constructed for operating the Hubble Space telescope is about a gigabyte in size and contains over 15 million star positions [9]. In the next section, we introduce a specific example and use it to illustrate the various steps in computing the Matching join.

In general, these problems can be classified into one of two categories: (i) entity matching and (ii) similarity matching problems.

2.2 Entity Matching

Problem such as the one encountered in Astronomy can be categorized as an entity matching problem. It can be defined as the problem of identifying the same entity which is represented differently in different databases. It may happen due to the use of nonstandard identifiers, synonyms, and homonyms in different databases. It can be also caused by temporal and instrument variations and data entry errors. This problem has been also referred to in the database literature as the instance identification [23] or the key equivalence problem [16].

The occurrence of entity matching problems are not limited to Astronomy. Similar problems are widely encountered in social sciences, such as, comparing health records of workers exposed to asbestos or radiation; estimating survival statistics (e.g., for cancer patients), and in the development of disease incidence files (e.g., ischemic heart disease) [2]. These applications require matching of individual's records maintained by different organizations over different periods of time. A standard join is inadequate here as the identifiers that are locally assigned by hospitals and employers are not standardized to help entity identification across databases. Even when standard identifiers, such as social security numbers (SSNs) are assigned, they are frequently not reported or reported wrong. A study showed that more than 6 million individuals have two or more SSNs, while 6 percent of all death certificates do not report any SSN [18].

2.3 Similarity Matching

Similarity matching represents a different class of problems, where the objective is to identify distinct records that exhibit similar characteristics [19]. Matching of identifiers may not be very informative for these problems as the records being matched do not refer to the same entity. An approximate (as opposed to an exact) match becomes appropriate here as distinct records are often similar but rarely identical.

Applications of similarity matching can be found in chemistry and related biological sciences, such as biochemistry and pharmacology [12]. Searching for similar chemicals, location of rare functional groups, comparison of bond lengths/strength, identification of critical fragments, detection of cliques are but a few applications where the matching techniques can be used. A similarity match becomes applicable here as retrieval of chemicals with similar properties is often as useful as searching for an exact match. Moreover, exact matches frequently do not exist.

An interesting application of approximate matching occurs in Space Physics, where interplanetary magnetic field components or solar wind velocities are measured at regular intervals, often by two or more satellites. The result is stored in a table which has a column for each quantity measured with the time of measurement as the primary key [18]. The physicists have to compare these observations to obtain the time dependence of measurements and correlations. However, the time when the observations are made by different satellites are not necessarily synchronized. Consequently, an exact match (or, an equi-join) becomes ineffective in such cases.

These are a few examples of query types that cannot be modeled satisfactorily as exact joins because of the heterogeneity in the databases. A detailed list of other matching problems can be found in [4, 19]. In the following sections, we discuss a special type of join, the Matching join, that can be used to process the above queries.

3 Matching Join Framework

In this section, we introduce the concept of Matching join and show its appropriateness in heterogeneous SDBs. We focus primarily on the (interdatabase) equi-join both because there are interesting theoretical issues associated with this type of join, and the fact that it occurs naturally in many settings.
In conventional database terminology, equi-joins are always exact as opposed to approximate. The join attributes are required to be exactly equal for two tuples to join. However as illustrated in the last section, the exact equality of the join attributes is often too strict a condition for scientific applications. This is because, the users often lack sufficient information to meaningfully pose an exact query. In a heterogeneous environment, the problem is further compounded by the non-standard data representations which, if not accounted for, can lead to wrong conclusions when constructing the join.

The Matching join model can be used to address both the heterogeneity as well as the approximation problems. Our objective is to identify tuple pairs which fully or partially (or approximately) satisfy the join conditions when the join attribute values are heterogeneous and/or not exactly equal. Thus, the two primary uses of this model can be summarized as follows:

1. to resolve the heterogeneity problem, i.e., identify and join records that are represented differently in different databases, and
2. to provide approximate answers when the users lack sufficient knowledge to pose an exact query.

In the Matching join framework, the equality constraint is relaxed and replaced by equivalence. The idea is to match attributes that represent the same entity or concept, rather than the same value. This is formally expressed in the following way. Consider two relations, \( r(R) \) and \( r(S) \) where, \( R \) and \( S \) denote the schemas for the two relations respectively. (We assume that the relations are from two different databases.) Let the tuples of \( r(R) \) be \( r_1, i = 1, \ldots, K \). Similarly, let the tuples of \( r(S) \) be denoted by \( s_i, i = 1, \ldots, L \). Assume there are \( n \) user-defined equi-join conditions of the form:

\[
r_i(a_k) = s_j(a_k) \quad \forall k = 1, \ldots, n
\]

where \( a_k \in R, S \forall k = 1, \ldots, n \). Let \( M \) be a set containing the accurate result of the join query, that is, the join result given no heterogeneity. Then in a heterogeneous environment, \( M \) can be expressed as follows, where the symbol "\( \equiv \)" is being used to indicate equivalence:

\[
M = r(R) \Join r(S) = r(R) \times r(S)
\]

such that \( r_i(a_k) \equiv s_j(a_k) \quad \forall k = 1, \ldots, n \)

It may appear from this formulation, that we are considering only a narrow band of joins, where the identical attributes in two relations are equi-joined. That is not necessarily the case. The model can be used to compare any pairs of attributes as long as the information as to how to compare them is available. The equivalence operator indicates that a mere comparison of values may not be sufficient.

The purpose of the Matching join is to evaluate \( M \), the result of the query, as precisely as possible. In this paper, we propose the following three step procedure for this purpose:

**Query transformation**: This constitutes the first step in the evaluation of the Matching join. Its objective is to resolve the heterogeneity problem by supplementing (or substituting, as the case may be) the user supplied join conditions with additional ones so that the ambiguity in the matching process is minimized. Once the appropriate conditions are selected, they have to be evaluated. This leads us to the next step in Matching join evaluation, the attribute comparison.

**Attribute comparison**: Evaluation of join conditions typically involves comparison of attributes. If these attributes are heterogeneous, the corresponding join conditions tend to get fairly complex. In order to model such conditions and resolve the heterogeneity problem, we support rules in our framework.

These features are usually adequate during an exact match. However, depending on the nature of the query and the information available to the user, sometimes it becomes necessary to use approximations, which constitutes the third step in our framework. We define a measure called the *comparison value* for this purpose.

**Comparison value estimation and result computation**: Since not all tuples satisfy the join conditions to the same degree, the comparison value, \( c \) is needed to distinguish the tuples. The appropriateness of joining tuples \( r_i \) and \( s_j \) is numerically denoted by \( c_{ij} \); a higher value indicating a *better* join. We provide a deterministic and a probabilistic scheme to evaluate the comparison value later in the paper. We also illustrate how to determine a minimum cost cutoff value for \( c \), such that only the tuples whose \( c \) value is greater than the cutoff are retained as the result of the query.

The issues related to each of these stages are analyzed in detail in the following sections.

4 Query Transformation

Query transformation is the first step in evaluating the Matching join. When the join attributes are heterogeneous, they cannot be directly used in the evaluation of the query. Query transformation is a scheme to substitute or supplement the heterogeneous join condition by a set of other appropriate conditions.

4.1 Basic Scheme

We denote the attributes that are useful for the evaluation of the join query as the *relevant attributes*. So, the attributes that are associated with the user defined join conditions, i.e., \( a_k \forall k = 1, \ldots, n \) in our model, are by default relevant to the query. We shall denote these as the *primary attributes*. If the primary attributes are not reliable enough or are heterogeneous, they may be substituted by other attributes that are relevant to the join condition. A non-primary attribute that is relevant to a join condition is known as a *secondary attribute*. The join conditions involving primary attributes are called the *primary conditions* and those involving the secondary attributes are called the *secondary conditions*.

In this paper, we argue that to be able to substitute a primary condition by a set of secondary ones,
the secondary attributes ought to be functionally depen-
dent on the primary attribute. In other words, all attributes that are
determined by a primary attribute are potential secondary attributes. The logic
behind this approach is fairly simple. An attribute contributes to the
definition of an object, not the other way around. When two records share
the same identifier, we infer that they refer to the same real world
instance. For example in Astronomy, if the spatial coordinates are identical
in two records, one concludes that the records describe the same star.

Here, we apply the same reasoning. There are typically
more than one attribute that describes an
entity in a database. Most of these would match
when the two records describe the same (or similar) real
world instance(s), even if the primary attributes are
heterogeneous. Simply stated, the same attribute
describing the same (or similar) entity instance(s) in
different databases should contain the same (or similar)
value(s). Similarly, if the two records refer to two dis-
similar real world instances, the secondary attributes
are less likely to match. Thus, by considering the
secondary attributes, the entities they describe can
be identified with a reasonable accuracy. So the sec-
todary conditions can be used to substitute a primary
condition involving a heterogeneous attribute in a join
query.

Of course, the secondary attribute values may not
necessarily be unique. For instance, it is possible for
several records to have the same brightness. This may
give rise to ambiguous situations, specially during en-
tity identification. The probability of such occurrences
decrease with the number and informativeness of the
relevant attributes, but rarely will it be zero. We con-
sider this aspect of matching errors at length later in
the paper.

The idea of using non-key attributes to help identify
or-inalus was originally mentioned in [23, 20]. A similar
approach was recently suggested by Lim et al, who
proposed a set of attributes, called the extended key,
to identify the tuples [14]. But the attributes that
can be used for entity identification were not speci-
fied in these papers. Several authors have addressed
the heterogeneity problem in literature assuming ho-
5merogeneous primary attributes, and have allowed only
the non-primary attribute to be heterogeneous [7]. In
our model, we allow even the primary attributes to
be heterogeneous. It may appear that we are making
the reverse assumption of homogeneous secondary at-
tributes. That however, is not the case. We of course
take the homogeneous attributes into consideration if
and when they are available1. But the model is not
dependent on the existence of these attributes. Using
inference rules, which have been discussed in detail in
[5], we derive secondary attributes which are homoge-
neous, even though the base attributes were hetero-
genous to begin with. Thus, the derived attributes
play a major role in instance identification and pro-
vide considerable strength to our model.

---

4.2 Relevant Attributes

In order to identify secondary attributes, we take
into consideration the functional dependencies (FDs)
in the database. We use the standard notation of
X → Y to indicate attribute X determines attribute
Y or Y is functionally dependent on X. Here X is a
determinant.

Our model is based on the fact that if two records
have the same value of X, then the corresponding
value of Y must be the same for both records. If
X constitutes the primary attribute then Y can be
considered to be a secondary attribute to X. Since
more than one instance of X can determine the same
Y value, we try to obtain uniqueness by considering
as many (functionally dependent) secondary attributes
as possible. If we have two FDs, X → Y and
X → W, Y, Z, and x₁, x₂ ∈ X, w ∈ W, y ∈ Y and
z ∈ Z, then the following conjecture holds:

\[
Pr(\exists x₁ ≠ x₂ | (x₁, w, y, z) ∧ (x₂, w, y, z) ∈ DB) ≤ Pr(\exists x₁ ≠ x₂ | (x₁, y) ∧ (x₂, y) ∈ DB)
\]

where DB represents the database. If two records
are truly distinct, then it is far more likely to be de-
tected by comparing a large number of their attributes
than by considering only a few ones. In other words,
the chance of accurate identification increases with
the number of attributes to consider. We are likely to get
the strict inequality if W, Y and Z are mutually in-
dependent and the equality if the W and Z are fully
dependent on Y, either directly or transitively².

Assume that X is the join attribute. Our objective
is to use as many attributes (including derived at-
tributes) as possible, which are mutually independent
and are determined by X. The following summarizes
our transformation principle:

If any primary attribute is a determinant D₁
and if the FD: D → A₁, A₂, ..., Aₙ, \( n \geq 1 \) holds, then the primary join condition
\( r₁(D₁) ≡ s₁(D₁) \) can be substituted by n sec-
todary conditions of the form:

\[ r₁(A₁) ≡ s₁(A₁) \land r₁(A₂) ≡ s₁(A₂) \land \ldots \land r₁(Aₙ) ≡ s₁(Aₙ) \]

In general, less the mutual correlation between
the secondary attributes, more informative they become in
identifying X. In reality, the secondary attributes
are unlikely to be mutually independent. It is also
unlikely that they would be fully correlated. In most
cases, inclusion of a relevant attribute is likely to in-
crease the probability of accurate identification by a
positive amount, though sometimes it may not be by
much. The weight to be assigned to a relevant at-
ttribute depends to a large extent on its ability to in-
crease the probability of identification. We illustrate
the query transformation using the following example.

Consider the problem of star catalogue matching
introduced earlier in the paper. Suppose that the cat-
alogues being compared, A and B, have the following scheme (the primary keys are in bold):

1 One can identify the homogeneous attributes in different
databases using the techniques suggested in literature, such as
[13, 20]. We assume this information will be available.

2 By direct dependence, we mean W = f(Y), where as a
transitive dependence means, Z = g(W) = g(f(Y)) = h(Y).

---

651
right ascension | declination | brightness | size |
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

To identify stars present in both catalogues, the following SQL query can be used:

```sql
SELECT all
FROM A, B
WHERE A[right ascension] = B[right ascension]
AND A[declination] = B[declination];
```

Since the recordings are made at different observatories, the join attributes are likely to be heterogeneous. In such a situation, the primary join conditions have to be substituted using secondary attributes. Assume that the following FD holds:

\[
\text{right ascension, declination} \rightarrow \{\text{brightness, size}\}
\]

Then, the above query can be transformed to the following:

```sql
SELECT all
FROM A, B
WHERE A[brightness] = B[brightness]
AND A[size] = B[size];
```

It should be pointed out that query transformation is more of a semantic tool rather than a syntactic one. For the system to meaningfully transform a condition of the form \( r_1[A] \equiv s_1[B] \), \( A \) and \( B \) ought to have the same semantics. It is not sufficient that they simply be determinants. Thus, for comparisons such as, those between employee numbers and machine numbers, no meaningful transformation may be found even though both attributes may be determinants. Of course, the user can substitute or supplement any condition at any point in time. The list of potential secondary attributes for each determinant can be stored in the metadata. These can be presented for consideration, when requested by the user, or if some of the primary join attributes are incompatible.

### 4.3 Substitute vs Supplement

We mentioned earlier that secondary conditions can be used to supplement or substitute the primary ones. In the example, we substituted the primary conditions. In general, when the primary attributes are heterogeneous, substitution may be the only option. If we had supplemented the primary condition in the last example, the transformed query would have looked like the following:

```sql
SELECT all
FROM A, B
WHERE A[right ascension] = B[right ascension]
AND A[brightness] = B[brightness]
AND A[size] = B[size];
```

For most situations, supplementing the primary conditions is a better strategy. If for example, the primary attribute(s) are homogeneous but contains many errors which could cause false mismatches and spurious matches, the user may still want to retain the primary condition but also include the secondary ones with proper weights.

Another situation where it is prudent to supplement is when the primary attribute(s) are expected to be homogeneous, but the possibility of homonyms cannot be ruled out. In many cases the user may be unaware of the homonyms which may be few and far between. In such cases, the user could retain the primary condition and supplement it with secondary ones with properly adjusted weights. If the user is aware of the existence of large number of homonyms, e.g., different organizations using the same set of identification numbers to identify different sets of individuals, substitution may be the better choice.

### 4.4 Perfect Transformation

This is a special case where it can be guaranteed that query transformation does not result in any loss of information. We mentioned earlier that the secondary attributes are usually not unique and joining on them could result in some spurious matches and omissions. Both these can be avoided however, if the transformation is perfect.

Suppose the FD: \( D \leftrightarrow A_1, A_2, \ldots, A_n \), \( n \geq 1 \) holds in the DB. Then, attribute \( A_i \), \( 1 \leq i \leq n \), is a perfect substitute for \( D \) if \( A_i \) determines \( D \), or a FD: \( A_i \rightarrow D \) also holds in DB.

In effect, we have obtained a FD: \( D \leftrightarrow A_i \), where both attributes uniquely determine each other. If the condition involving \( A_i \) is used to substitute the one involving \( D \), it will not result in any loss of information. That is why, perfect transformation is always the most desired transformation.

### 5 Attribute Comparison

The second step in the evaluation of the Matching join is the actual comparison of the attributes. These attributes could be primary or secondary depending on whether any query transformation was performed. We mentioned earlier that in order for two attributes to be compared, they need to be either homogeneous, or else a mechanism to compare them has to be devised. We first consider the homogeneous case and then analyze the non-homogeneous attributes which require a comparison mechanism. We would like to point out that in order to execute this step, it is not necessary to perform query transformation. The strategies outlined in this section can be applied directly to the primary conditions, if it is so desired.

#### 5.1 Homogeneous Attributes

Two attributes are considered homogeneous when they have the same semantics and it is meaningful to compare their values directly. In order to compare homogeneous attributes, we suggest two techniques: (i) equi-join and (ii) band join.

##### 5.1.1 Equality

The equi-join operator needs no introduction. This is the simplest possible operator to compare two attributes and is applicable in many settings. It is appropriate for situations where an exact match of the re-
event attributes is desired. The Matching join model contains the equi-join as one of its components.

5.1.2 Band Joins

The reason behind introducing the Matching join was to be able to relax the equality constraints. In a band join, two attributes are considered matched if their values are within a certain range, or band. Thus, a band join condition between two relations, \( r(R) \) and \( r(S) \) on attribute \( a \) can be expressed as: \( R[a] - c_1 \leq S[a] \leq R[a] + c_2 \). A tuple \( r \in r(R) \) joins with tuple \( s \in r(S) \) only if \( r[a] \) appears within a band of size \( c_1 + c_2 \) about \( s[a] \). Analysis of band join algorithms can be found in [21, 8].

Clearly, equality is a special case of band join, where the band length is equal to zero. Band joins can be considered as a special case of theta join, since a band join condition can be separated into two theta-joins, where both the inequalities involve the same pair of attributes. Since the band can be adjusted to account for instrument error and variations, the band join is very useful in finding records with similar properties in scientific and statistical applications. For example in Astronomy [9], brightness of two stars in two different catalogues, 1 & 2 say, are considered equivalent if their difference is within a band of size equal to three times the estimated error:

\[
\Delta(m_{a1} - m_{a2}) < 3E_{m_{a}}
\]

where the estimated error is derived from the knowledge of standard errors, \( \sigma \) with a minimum of 1 mag:

\[
E_{m_{a}} = \max(1^{m_{a}}, \sigma_1, \sigma_2)
\]

5.2.1 Derived Attributes

One way of converting two heterogeneous attributes to a compatible format is by using the concept of views [6]. The converted version of the attributes are called the derived attributes, and the conversion rules used in the derivation, the inference rules [9]. Ideally, one would like to replace the original values by the derived ones in order to attain uniformity. But typically this would not be possible as the databases are likely to be autonomous and the users, unwilling or incapable of changing their "ways". But the derived attributes can be stored in the database like views and can be used for interdatabase queries.

Once computed, the derived attributes behave like other ordinary attributes. Weights can be assigned to them, based on their informativeness and the users' confidence in the rules deriving them. Also, they can be compared using equality or band join, depending on the user needs. If they are relevant to a query, the derived attributes can be used to generate the secondary join conditions.

As a simple example of derived attributes, consider the problem of unit mismatch. Temperature is expressed in several different units depending on the device used for measurement. In order to compare these readings, they have to be converted to a common unit. This would require a set of rules which would specify the unit conversion principles. Using these rules, values in desired units can be derived. If the derived values are physically stored, then care needs to be taken in updating them whenever the base values are changed. In general, numerous heterogeneity issues like inconsistent codes, units, granularity and data type mismatch can be addressed by deriving appropriate attributes.

5.2.2 Rule-Based Joins

Rule-based joins are by far the most general and powerful tool to compare two attributes, primary or secondary, derived or user defined. Here, the basic idea is that two attributes are considered matched if they satisfy certain rules. We denote these rules as the Join rules. These could vary from simple join conditions expressed in the Where clause of SQL to arbitrarily complex rules expressed in a general purpose programming language.

As an example consider the following rule: *star positions in two records are equivalent if they are within 3 arc seconds of the great circle distance between them or their separation is less than 3 times the estimated error.* Due to its complexity, a language such as C will be necessary to implement it in a database. In general, any comparison scheme, however complicated, can be represented by a set of Join rules. Thus, the band and the equi-joins become special cases of Join rules. In addition, these rules have the ability to handle exceptions. Using rules, one can take into account the fact that names like Richard and Dick, though different, may actually refer to the same individual.

Join rules can be implemented in a next-generation DBMS, such as PostgreSQL, where any collection of database or C commands can be packaged together and defined as a rule. Information about the rules needs to be inserted in the system catalogues and rule locks, placed on the appropriate tuples [22]. The advantage of this implementation is that it makes the Join rules transparent to the user. The rules will provide the correct answer whenever the tuples need to be matched. We believe that using a combination of database commands and C functions any Join rule can be implemented in a DBMS.

6 Comparison Value Estimation

The third step in the Matching join evaluation is the comparison value estimation. As mentioned earlier,
the comparison value plays a central role during the computation of approximate answers. In this section we first discuss how the weights can be used during the join. We then present the definition of comparison value and demonstrate how weights can contribute to its computation.

Once the comparison value is available, one is confronted with the final decision of whether or not to include the corresponding tuple in the join result. We analyze this decision problem both from a deterministic as well as from a probabilistic perspective. Finally, we present a computational short cut that can be used to compute Matching joins in an efficient way.

6.1 Assignment of Weights

The idea of using weights in a Matching join was mentioned earlier in the paper. Here we briefly describe the use of weights with Matching join conditions.

A join query often consists of several join conditions, all of which the user would like to see satisfied. However, not all the conditions are equally important, nor are the data equally reliable and informative. Such information cannot be captured in a query using standard SQL.

In a Matching join, we allow weights to be assigned to the join conditions so that the crucial ones receive higher priority. This approach is useful even if the join is not an equi-join but a general theta-join. The advantage of using weights is that it allows more important conditions to take precedence over less important ones. Also, weights allow one to execute the join without requiring the tuples to satisfy all the join conditions.

This is particularly useful for many bibliographic and scientific applications, where one often searches for similar records. Two books or two sets of experimental results may be similar but are unlikely to be identical. The user could provide a set of conditions, all of which he/she would like to see satisfied. But there may not be any record in the database that satisfies all those conditions. So the join result could be a null set, unless the conditions are prioritized using weights and some of them (presumably the less important ones) are allowed to remain unsatisfied.

If there are multiple theta-join conditions or selection conditions, weights can be applied to prioritize them as well. Weights can also be used to compensate for data quality. If the proportion of error in an attribute is high, it should be substituted or at least supplemented by other conditions with proper weights, depending on the situation. In general, if an attribute is known to be error-prone it receives a lower weight than a error free attribute of comparable informativeness.

Weights to be assigned to various join conditions are best specified by the user. Although we do not make specific recommendations in this regard, we would like to point out a couple of options the user might want to consider. Firstly, there is the issue of negative weights. It is usually common to equate a match to a positive weight, but the situation is not obvious, when two attributes do not match or a condition is not satisfied. It is conceivable that one might want to assign a negative weight to a mismatch. In some cases, a mismatch of two attributes could be very informative and should receive as much (negative) weight as their match.

Secondly, there is the issue of statistical dependence between the attributes. Often there are situations, where matching of one pair of attributes, e.g., job title, makes the matching of another pair more likely, e.g., salary range. In this case, a mismatch conveys more information than a match. Similarly, the reverse situation can occur as well, where the matching of one pair of attributes could make the matching of another pair much less likely. In such cases, the informativeness of the second attribute decreases in the presence of the first one, and the weights have to be adjusted accordingly.

6.2 The Formulation

The comparison value, $c_{ij}$, is a numeric estimate of how well tuples $r_i \in r(R)$ and $s_j \in r(S)$ satisfy the join conditions. In other words, it is a measure of the appropriateness of joining $r_i$ with $s_j$, given the join conditions.

In order to define the comparison value function, we use the following notation. Let a comparison vector for a tuple $t_{ij} = (r_i, s_j)$ be defined as: $\gamma(t_{ij}) = \{\gamma_1(t_{ij}), \gamma_2(t_{ij}), \ldots, \gamma_n(t_{ij})\}$, where the number of components of $\gamma(t_{ij})$ is equal to the number of join conditions in the transformed query. (If the transformation step is by-passed, $n$ is simply the number of primary conditions.) (We denote vectors in bold type.) As described earlier in the paper, these conditions could be any combination of equi-join or band join conditions or employ more complex rules. This is captured by the equivalence operator in each individual component.

Each component of the vector contributes a weight depending on whether the conditions corresponding to that component is satisfied by $t_{ij}$. If the weights were omitted during the earlier stages, we shall assume that all the join conditions have equal priority and assign a binary weight, where 1 signifies a match and a zero, otherwise. Thus, we have:

$$\gamma_1(t_{ij}) = w_1 \text{ if } r_i[a_1] \equiv s_j[a_1]$$
$$= w'_1 \text{ otherwise}$$
$$\gamma_2(t_{ij}) = w_2 \text{ if } r_i[a_2] \equiv s_j[a_2]$$
$$= w'_2 \text{ otherwise}$$
$$\vdots$$
$$\gamma_n(t_{ij}) = w_n \text{ if } r_i[a_n] \equiv s_j[a_n]$$
$$= w'_n \text{ otherwise}$$

where, $w_k > w'_k$, $\forall i = 1, \ldots, n$.

The weights contributed by individual components can be combined in a number of different ways, such as, simple sum, normalized sum, squared sum, etc. If $h$ represents this function, then the result of comparing tuples $r_i$ and $s_j$ can be expressed as:

$$v_{ij} = h[\gamma(t_{ij})]$$

454
A cutoff value is usually set to eliminate the lesser amount of information available. Consequently, this problem constitutes a match. This decision depends on the type of M. It may be that Type - II. A type-II error (or false alarm) occurs when \( t \not\in \mathcal{M} \) but \( t \in \text{Result} \). This means, an unmatched tuple has been included in the Result.

Tuples are retrieved in responses to queries. There are costs associated with the inclusion of unmatched tuples and the exclusion of matched ones. Our objective is to derive an optimum cutoff value that would minimize the expected costs of these errors.

Let \( \alpha_{II} \) be the cost of including an unmatched tuple in the Result, which is incurred every time a type-II error is committed. Similarly, let the cost of omitting a matched tuple from the Result be \( \alpha_{I} \). This is the cost of making a type-I error. Furthermore, let \( \beta \) be the desired cutoff value and let the probability density function associated with the comparison value be \( f(c) \). (In the rest of the analysis, we assume \( c \) to be a probability.)

When the comparison value is \( c \), the cost of making a type-I error, \( C_I \), is given by:

\[
C_I = \alpha_1 c
\]

An omission occurs only if the comparison value is less than the cutoff. So the expected cost of type-I error can be computed as:

\[
E[C_I] = \int_{-\infty}^{\beta} \alpha_1 cf(c)\,dc
\]

Similarly, the cost of making a type-II error is given by:

\[
C_{II} = \alpha_{II} (1 - c)
\]

This cost is incurred only if the comparison value is greater than the cutoff. So the expected cost of type-II error can be computed as:

\[
E[C_{II}] = \int_{\beta}^{\infty} \alpha_{II} (1 - c)f(c)\,dc
\]

In order to evaluate these costs, we require the knowledge about the distribution of \( c \). Let us consider the simple case, when \( c \) is uniformly distributed, that is,

\[
f(c) = \begin{cases} 
1, & \text{if } 0 < c < 1 \\
0, & \text{otherwise}
\end{cases}
\]

Then, the total expected cost of error, \( E[C] \) is given by:

\[
E[C] = E[C_I] + E[C_{II}]
\]

\[
= \int_{0}^{\beta} \alpha_1 c\,dc + \int_{\beta}^{1} \alpha_{II} (1 - c)\,dc
\]

\[
= C_I \beta^2 / 2 + C_{II} \left\{ (1 - \beta) - 1 - \beta^2 / 2 \right\}
\]

In order to minimize the total expected cost, we differentiate the above expression with respect to \( \beta \) and equate the result to zero. This yields:

\[
C_I \beta - C_{II} + C_{II} \beta = 0
\]
which implies,

$$\beta = \frac{C_{II}}{C_I + C_{II}}$$

Thus by setting $\beta$ equal to the above value, the cost of error can be minimized. (The second differential, $C_I + C_{II}$ is positive, since costs cannot be negative. So this is truly a minima.) Similar expressions can be derived for other distributions of $c$.

### 6.5 A Processing Short-Cut

Before concluding this section, we would like to suggest a processing short-cut that takes advantage of the special Matching join framework. Matching join employs $n$ conditions, some of which may involve complex rules or band joins. It is conceivable that evaluating all these conditions may require a lot of processing. By applying the short-cut discussed below, we might be able to avoid evaluating some of these conditions and thereby reduce the processing cost.

Our basic approach is the following. Each condition contributes a certain weight towards the comparison value. So a simple approach is to evaluate those conditions that contribute the largest weights without requiring a lot of processing. We know from our previous analysis that the cutoff value is instrumental in deciding if a tuple should be included in the result. Let the value $\nu$ corresponding to $\beta$ be denoted as $\overline{v}$. Thus, for the deterministic case, $\overline{v}$ is the minimum contribution in weights a tuple must have to be included in the Result. For the probabilistic case, it is minimum value of $v$ for which

$$\Pr \{ t \in M | v \} \geq \beta$$

holds. (We are assuming that the function $\Pr\{v\}$ is monotonically nondecreasing in $v$, with $\Pr\{v = v_2\} \geq \Pr\{v = v_1\}$ if $v_2 > v_1$.)

Let us order the $n$ conditions and renumber them such that the condition contributing the largest weight but requiring the least processing is at the top of the order and the condition contributing the least weight and requiring the most processing is at the bottom. (This is similar to the predicate ranking scheme suggested in [11])

$$w_i = h_p(t_{ij}) = \sum_{j \in i} \gamma_i(t_{ij}).$$

Then for each tuple pair $t_{ij}$, $v$ can be evaluated using the following algorithm:

```
Step 0. $v := 0$
max_weight := $\sum_{i=1}^n w_i$

Step 1. $i := 1$

Step 2. If condition $i$ is satisfied

then $v := v + w_i$
else $v := v + w_i$

max_weight := max_weight - $w_i$

Step 3. If $v \geq \overline{v}$

then stop; $t \in Result$
else if $v + max_weight < \overline{v}$

then stop; $t \notin Result$
else $i := i + 1$

go to Step 2
```

In this algorithm, we not only check after evaluating each condition $^3$ if $v$ has exceeded the cutoff, we also check if there is any point in continuing the process. If the remaining conditions do not have adequate weight to make $v$ greater than $\overline{v}$, there is no point in evaluating those conditions. This enables us in reducing the number of conditions evaluated per tuple and hence, the join processing cost.

### 7 Conclusions

Organizations often need access to scientific data stored in independently managed heterogeneous databases. Data in these databases are collected over a period of time, often in an uncoordinated way without a consideration that one day they may be shared. Accessing and manipulating data in a heterogeneous environment is consequently rendered very difficult.

In this paper, we analyzed the problem caused by data heterogeneity in SDBs. This problem occurs when the data conveying the same or similar information is represented differently in different databases. Scientific and statistical applications also require approximate matching of entities which are similar but not necessarily identical.

We introduced the Matching join to process queries in scientific databases. Query transformation was used to substitute or supplement the join conditions when the join attributes were heterogeneous. We took advantage of the functional dependencies in the database to incorporate additional knowledge. We also extended the Matching join by including weights to prioritize the join conditions, and rules, to map the join attributes to a mutually compatible format before they were compared.

We proposed a numeric measure, the comparison value, to estimate the degree to which a particular tuple satisfies the query conditions, and suggested ways of deriving it. We first examined the simple deterministic case. Then, to capture the uncertainty associated with the matching process, we extended the model to include probabilistic reasoning. We analyzed the problem of estimating the cutoff value that would minimize the cost of errors. We also proposed a short-cut that can be used to reduce the join processing cost.

The area of SDBs offers interesting and challenging opportunities in database management research. In particular, we are looking at the following areas for further analysis:

- Extending the model to cover other sources of data heterogeneity. The extended model will take into account the informativeness of the attributes and suggest weights accordingly.
- Extending the results of this paper to optimize more complicated queries, like combinations of band joins and approximate pattern matching.
- Implement the Matching join using a relational DBMS like Postgres.

$^3$Actually, this test is not needed for the first $k$ conditions, where $k = \min\{ w_1, \ldots, w_i \} \geq \overline{v}$, $i \leq n$. 

468
References


