Semantic Integration in Heterogeneous Databases Using Neural Networks †

Wen-Syan Li Chris Clifton
Department of Electrical Engineering and Computer Science
Northwestern University
Evanston, Illinois, 60208-3118
{acura,clifton}@eecs.nwu.edu

Abstract

One important step in integrating heterogeneous databases is matching equivalent attributes: Determining which fields in two databases refer to the same data. The meaning of information may be embodied within a database model, a conceptual schema, application programs, or data contents. Integration involves extracting semantics, expressing them as metadata, and matching semantically equivalent data elements. We present a procedure using a classifier to categorize attributes according to their field specifications and data values, then train a neural network to recognize similar attributes. In our technique, the knowledge of how to match equivalent data elements is "discovered" from metadata , not "pre-programmed".

1 Introduction

One problem in developing federated databases is semantic integration: determining which fields are equivalent between databases. Attributes (classes of data items) are compared in a pairwise fashion to determine their equivalence. Manually comparing all possible pairs of attributes is an unreasonably large task, especially since most pairs do *not* represent the same information. Simple ad-hoc guesswork, on the other hand, is likely to miss some attributes that should map to the same global attribute. US West reports having 5 terabytes of data managed by 1,000 systems, with customer information alone spread across 200 different databases [DKM+93]. One group at GTE began the integration process with 27,000 data elements, from just 40 of its applications. It required an average of four hours per data element to extract and document matching elements when the task was performed by someone other than the data owner [VH94]. Other GTE integration efforts have found the elements overlapped or nearly matched in their database to be close to 80% [VH94].

[DKM+93] pointed out some important aspects of semantic heterogeneity: Semantics may be embodied within a database model, a conceptual schema, application programs, and the minds of users. Semantic differences among components may be considered inconsistencies or heterogeneity depending on the application, so it is difficult to identify and resolve all the semantic heterogeneity among components. Techniques are required to support the re-use of knowledge gained in the semantic heterogeneity resolution process.

The goal of our research is to develop a semi-automated semantic integration procedure which utilizes the meta-data specification and data contents at the conceptual schema and data content levels. Note that this is the only information reasonably available to an automated tool. Parsing application programs or picking the brains of users is not practical. We want the ability to determine the likelihood of attributes referring to the same real-world class of information from the input data. We also desire to have the ability to

†This material is based upon work supported by the National Science Foundation under Grant No. CCR-9210704.

Permission to copy without fee all or part of this material is granted provided that the copies are not made or distributed for direct commercial advantage, the VLDB copyright notice and the title of the publication and its date appear, and notice is given that copying is by permission of the Very Large Data Base Endowment. To copy otherwise, or to republish, requires a fee and/or special permission from the Endowment.

Proceedings of the 20th VLDB Conference
Santiago, Chile, 1994

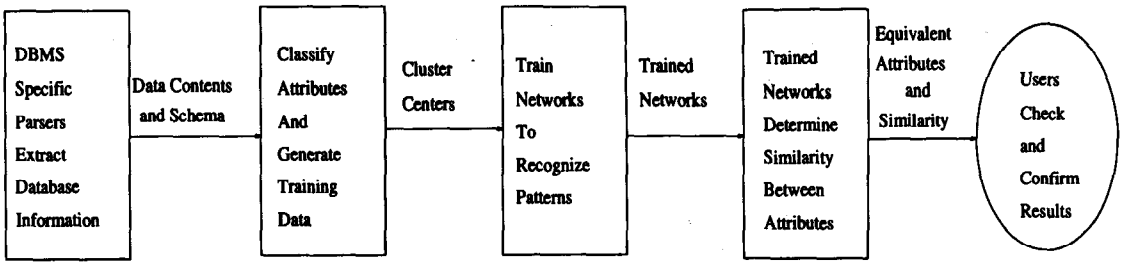

Figure 1: Procedure of Semantic Integration Using Neural Networks

reuse or adapt the knowledge gained in the semantic heterogeneity resolution process to work on similar problems. We present a method where the knowledge of how to determine matching data elements is *discovered*, not *pre-programmed*.

We start with the assumption that attributes in different databases that represent the same real-world concept will have similarities in structure and data values. For example, employee salaries in two databases will probably be numbers greater than 0 (which can be determined from constraints, this is structural similarity). The same can be said for daily gross receipts. However, the range and distribution of data values will be very different for salaries and gross receipts. From this we can determine that two salary fields probably represent the same real-world concept, but gross receipts are something different. Note that the assumption is that there are similarities, not that we know what those similarities are.

In Figure 1 we outline our method. In this process, DBMS specific parsers extract information (schema or data contents) from databases. We then use a classifier that learns how to discriminate among attributes in a single database. The classifier output, cluster centers, is used to train a neural network to recognize categories; this network can then determine similar attributes between databases.

As an example of how this system could be used, imagine that we are planning university admissions. We wish to make sure that we do not admit too many students; to do this we need to check if classes are oversubscribed. We are used to using the Registrar's database which contains the number of students who took a course and the room number (among other information). However, it does not contain room size information. After some checking, we are told that Building and Grounds maintains a database which contains room size and we are given access to this database.

A first attempt is to issue the query (using a multidatabase query language such as [Lit89], or better yet a database browser with multidatabase support):

```
Select sum(c.room#)
From Registrardb.classes c, B+G.rooms r
Where c.room# = r.room# and c.size >= r.size
```

However, we are given the surprising result of 0! We then remember to use the semantic integration tool (using a pre-trained network); in seconds we are given a list of likely matches which shows that *r.seats* matches *c.size* much more closely than r.size. (Building and Grounds considers the size of a room to be the size in square feet). We then reissue the query (using r.seats instead of r.size) and are given the number of classes which filled their rooms. Note that the main user effort involved is finding the appropriate databases; the integration effort is low. The end user is able to distinguish between unreasonable and reasonable answers, and exact results aren't critical. This method allows a user to obtain reasonable answers requiring database integration at low cost.

This paper is organized as follows. We first review existing work in this area. In Section 3 we discuss the semantics available from databases. In Section 4 we describe our technique of using a self-organizing map classifier to categorize attributes and then train a network to recognize input patterns and give degrees of similarity. In Section 5 the experimental results of testing our techniques on three pairs of real databases are presented. Finally, in Section 6 we offer our conclusions.

2 Related Work

A federated architecture for database systems was proposed by McLeod and Heimbigner in [MH80]. A basic conceptual technique for integrating component views into a "superview" was introduced by Motro and Buneman [MB81]. The Multibase project [SBU+81, DH84] by the Computer Corporation of America in the early 80's first built a system for integrating preexisting, heterogeneous, distributed databases. The process of schema generalization and integration, however, still needs the involvement of database designers to find those objects that contain information of the same domain or related data. This becomes a bottleneck for schema integration when the size of the database is large.

One approach for determining the degree of object equivalence, proposed by Storey and Goldstein [SG88], is to compare objects in a pairwise fashion by consult-

ing a lexicon of synonyms. It is assumed that some classes or at least some of their attributes and/or relationships are assigned with meaningful names in a pre-integration phrase. Therefore, the knowledge about the terminological relationship between the names can be used as an indicator of the real world correspondence between the objects. In pre-integration, object equivalence (or degree of similarity) is calculated by comparing the aspects of each object and computing a weighted probability of similarity and dissimilarity. Sheth and Larson [SL90] noted that comparison of the schema objects is difficult unless the related information is represented in a similar form in different schemas.

2.1 Existing Approaches

In [DKM+93] it is noted that semantics are embodied in four places: The database model, conceptual schema, application programs and minds of users. An automatic semantic integration procedure can only make use of information contained in the first two. We further break this into three parts: The names of attributes (obtained from the schema); attribute values and domains (obtained from the data contents); and field specifications (from the schema, or in some cases from automated inspection of the data). We detail these approaches below.

2.1.1 Comparing attribute names

Systems have been developed to automate database integration. One that has addressed the problem of attribute equivalence is MUVIS (Multi-User View Integration System) [HR90]. MUVIS is a knowledge based system for view integration. It assists database designers in representing user views and integrating these views into a global conceptual view. MUVIS determines the degree of similarity and dissimilarity of two objects during a *pre-integration* phrase [1].

The similarity and dissimilarity in MUVIS is primarily based on comparing the *field names* of the attributes. Object equivalence is determined by comparing the aspects of each (such as class names, member names, and attribute names) and computing a weighted value for similarity and dissimilarity. A recommendation is then produced as to how the integration should be performed.

Most automated tools developed to assist designers in establishing object correspondences by comparing attribute names work well for homonyms (same name for different data), as users are shown the false match. However, different objects can have different synonyms that are not easily detected by inspection. This shifts the problem to building the synonym lexicon. Even a synonym lexicon has limitations because it is difficult for database designers to define a field name by using only the words that can be found in a dictionary or abbreviations carrying unambiguous meanings and in some cases, it is difficult to use a single word rather than a phrase to name a field. These reasons make it expensive to build a system of this approach. Sheth and Larson [SL90] also pointed out that completely automatic determination of attribute relationships through searching a synonym lexicon is not possible because it would require that all of the semantics of schema be completely specified. Also, current semantic (or other) data models are not able to capture a real-world state completely and interpretations of real-world state change over time.

[1] Since, in the real world, semantics of terms may vary, the relationship between two attributes is usually fuzzy. Therefore, a degree of similarity and dissimilarity has a strength of [0,1].

2.1.2 Comparing attribute values and domains using data contents

Another approach of determining attribute equivalence is comparing attribute domains. Larson et. al. [LNE89, NB86] and Sheth et. al. [SLCN88] discussed how relationships and entity sets can be integrated primarily based on their domain relationships: EQUAL, CONTAINS, OVERLAP, CONTAINED-IN, and DISJOINT. Determining such relationships can be time consuming and tedious [SL90]. If each schema has 100 entity types, and an average of five attributes per entity type, then 250,000 pairs of attributes must be considered (for each attribute in one schema, a potential relationship with each attribute in other schemas should be considered). Another problem with their approach is poor tolerance of faults. Small amounts of incorrect data may lead the system to draw a wrong conclusion on domain relationships.

In the tool developed to perform schema integration described in [SLCN88], a heuristic algorithm is given to identify pairs of entity types and relationship types that are related by EQUAL, CONTAINS, OVERLAP, and CONTAINED_IN domain relationships. Sheth and Gala [SG89] also argued that this task cannot be automated, and hence we may need to depend on heuristics to identify a small number of attribute pairs that may be potentially related by a relationship other than DISJOINT.

2.1.3 Comparing field specifications

In [NB86] the characteristics of attributes discussed are uniqueness, cardinality, domain, semantic integrity constraints, security constraints, allowable operations, and scale. In our prior work [LC93], we presented a technique which utilizes these field specifications to determine the similarity and dissimilarity of a pair of

attributes. It is assumed that given a database design application, different designers should tend to have similar schema and constraint design because they should have the same technology and knowledge about designing a "good" database. Thus information about attributes; such as length, data types, and constraints; can be used as "discriminators" to determine the likelihood that two attributes are equivalent. The experimental results show that characteristics of schema information are very effective "discriminators". This technique can be used with the other approaches, as a "first step" to eliminate clearly incompatible attributes. This allows the process of comparing attribute domain relationships, which is more computationally expensive, to work on a smaller problem. It can also be used to confirm conclusions reached using other approaches. However, this technique (as well as other related techniques) needs a theoretical basis for developing heuristics for degree of similarity and dissimilarity. Another weakness is that schema information may not be always available. We now discuss the information we use to determine database semantics. In Section 4 we present our integration method.

3 Semantics of Databases

We have described some of the information available to an automated semantic integration method. In this section we describe the specific pieces information we use as "discriminators". Note that this is not an exhaustive list; it is simply the information we believe to be readily available and useful. One advantage of our method is that the relative usefulness of these discriminators is discovered automatically; it does not hurt to provide "extra" discriminators that do not provide a good basis for comparing attributes. For input to the classifier, this information is mapped to a vector of values in the range [0,1] [2] , where each item in the vector represents one discriminator. The choice of discriminators (and the software to extract them) need be done only on a *per-DBMS* basis, and the technique presented allows the use of discriminators other than those we discuss. One piece of information we do *not* use is attribute names. This has been well studied, and is complementary to our method. Integrating a comparison of attribute names with this method is an area for future work.

3.1 Field Specifications

The characteristics of field specifications in the schema level we use are: data types, length, and "supplemental data types" such as format specifications (examples are the IBM AS/400 EDTWRD and EDTCDE specifications[3]); and the existence of constraints (primary keys, foreign keys, candidate keys, value and range constraints, disallowing null values, and access restrictions). It is not difficult to extract these characteristics from databases. Many relational databases store this information in tables, allowing SQL queries to extract the information. As another example, the schema definition files of IBM AS/400 use a fixed format. Thus we can develop a set of DBMS-specific parsers to obtain this information. We map some of this information to binary values (e.g. key field or not) and others to a range [0,1] (e.g. field length is mapped using the function $f(length) = 2 * (1/(1 + k^{-length}) - 0.5)$ [4]. Category information such as data types requires special treatment. For example, if we convert data types to a range [0,1] and assign the values 1, 0.5, and 0 to data types date, numeric, and character, then we are saying a date is closer to a numeric field than a character. We do not "pre-judge" such information, but let the classifier determine if this is true. Instead we convert this category input to a vector of binary values (e.g. 1,0,0 for date type, 0,1,0 for numeric type, and 0,0,1 for character type).

In some cases (such as flat-file data) we may not have an accessible schema definition. Many of the above characteristics can be determined by inspecting the data. This need not be a manual process, commercial tools such as DBStar are available that can automatically extract schema information from flat files.

3.2 Data Contents

The data contents of different attributes tend to be different even though their schema designs, such as data type and length, are the same. This is because their data patterns, value distributions, grouping or other characteristics are different. For example, "SSN" and "Account balance" can all be designed as nine-digit numerical fields; they may not be distinguishable based solely on their schema and constraint design characteristics. However, their data contents are different, and therefore their data patterns, value distributions, and averages are all different. Thus, examining data contents, the technique used in the content level, can correct or enhance the accuracy of the outcomes from the dictionary level and the schema level.

Note that this is not the same as the domain anal-

[2] Without normalization, the effect of one node with analog input may be much more significant than the effect of other nodes with binary input as "discriminator".

[3] EDTWRD (edit word) is used to specify for a particular field. EDTCDE (edit code) provides specific formats for numeric fields.

[4] For schema information, we use $k = 1.1$. This provides reasonable discrimination for the range [0,50]. We use the function $f(x) = 1/(1 + k^{-x})$ for normalizing numeric ranges, with $k = 1.01$. Varying k compensates for rounding errors.

ysis of Larson et. al. [LNE89, NB86] and Sheth et. al. [SLCN88]. Domain analysis compares the complete contents of each pair of attributes. We instead perform a one-time analysis of each attribute to obtain a set of characteristics that describe that data. We divide these characteristics into two types: Character and Numeric. Types that do not map into these (such as binary fields or user-defined types) are rare enough that other information should be sufficient to discriminate them from other types.

Data patterns for character fields

1. The ratio of the number of numerical characters to the total number of characters. For example, the ratio of numbers to characters in License Plate for different states are different. This ratio of Last_Name or First_Name should be zero. But for the field Stud_Id whose data type is designed as character (e.g. 999-99-9999), this ratio is 9/11. For the field Address, this ratio should be lower.

2. Ratio of white-space characters to total characters: A Last_Name or First_Name field will contain few white-space characters. An Address field will normally contain some white-space.

3. Statistics on length: In addition to the simple maximum length of a field, we compare the average, variance, and coefficient of variance of the "used" length relative to the maximum length. An ID field will typically use the full field all the time, while a name field will use less and vary more.

Data patterns for numeric fields

For numeric fields, we can use statistical analysis of the data as a discriminator. The statistics we use are:

1. Average (Mean): Average is one of the characteristics for those objects whose data types are number. For example, the Savings Accounts and Checking Accounts will likely have different average balances. The average weights for Ships and Cars should be different also.

2. Variance: The variance is defined as the expectation (or average) of the mean deviations squared. The variance is a statistic for measuring the variability of a set of numbers. It places higher weights on observations which are far away from the mean because it squares the mean deviations.

3. Coefficient of variation (CV): This is the square root of variance (standard deviation) divided by average (mean). CV is a scaled measure of variability. Comparing the CV of two fields can clarify some structural incompatibility such as:

   - Units: Different databases use different units for the same data element (e.g. weight in tons or in kg).
   - Granularity: Data elements representing measurements differ in granularity levels, e.g., sales per month or annual sales.

   Relative measures of variability, like the coefficient of variation, are very useful for comparing numbers of vastly different units or different granularity levels because these measures are based on percentage.
   Even though different databases use measures in different units or in different granularity levels for the same data item, the coefficients of variation of this data item should be similar. A coefficient of variation of a data item can be used as an effective indicator for finding the fields which are structural incompatible in units or granularity levels.

4. Grouping: In many cases, the data of a field can be sorted into several groups. For example, Zip Code (first five digits), Phone_Number (area code or first three digits), Social Security Number (first three digits), City Name, and State Name.

4 Semantic Integration Method

The discriminators provide a good deal of information to characterize attributes. However, it is difficult to determine just which discriminators will be helpful, and which will be little more than "noise". Programmed computing is best used in those situations where the processing to be accomplished can be defined in terms of a known procedure or a known set of rules. Clearly, there is no perfect procedure or known set of rules which solves the problems of determining the semantic equivalence of attributes in heterogeneous databases since the relationships between attributes are fuzzy and availability of database information varies.
Neural networks have emerged as a powerful pattern recognition technique. Neural networks can learn the similarities among data directly from instances of data and empirically infer solutions from data without prior knowledge of regularities. We use them as a bridge over the gap between individual examples and general relationships. The advantages of using neural networks for determining attribute equivalence over methods with fixed rules are

1. Neural networks can perform a task such as classification and generalization without being given rules since neural networks are trained, not programmed. They are therefore easier to adapt to new problems and can infer relationships unrecognized by the programmers.

2. The weights assigned can be adjusted dynamically according to the input data. The weights can also be readjusted according to new input data.

3. Neural networks can generalize because of their ability to respond correctly to data not used in training. Generalization is important since the input data; the names, the field specifications, and the values of attributes in heterogeneous databases; is often "noisy" and incomplete.

First, the available information from an individual database (discussed in Section 3) is used as input data for a self-organizing map algorithm to categorize attributes. Second, the output of classifier (tagged with a target result) is used as the training data for a category learning and recognition algorithm. The trained recognition algorithm then determines the similarity between attribute pairs in different databases. We will first present an overview of self-organization and category learning and recognition algorithms. The complete procedure is given in detail in Section 4.3.

4.1 Self-Organizing Map Algorithm

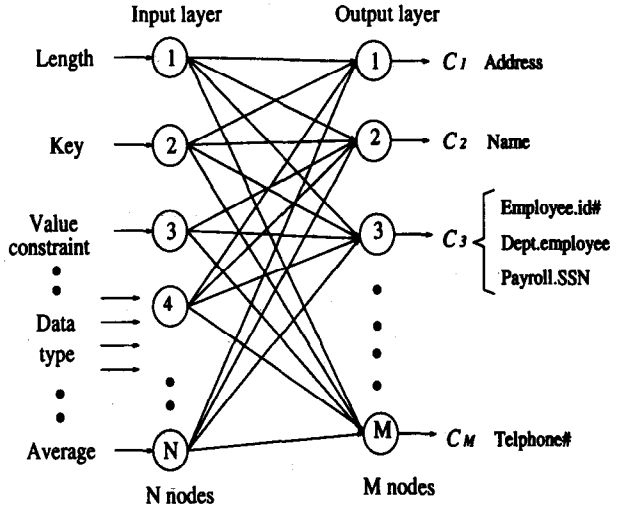

Figure 2: Self-Organizing Map Network Architecture

The self-organizing map algorithm [Koh87] is an *unsupervised* learning algorithm. It categorizes input patterns into M groups, where M is set by users. In the semantic integration process we want to control how similar the patterns are in a category, not how many categories are formed. Therefore, we have adapted it so that users can determining how fine these categories are by setting the radius of clusters (threshold) and create a new cluster when a new pattern is not close to any existing cluster.

It serves as a clustering algorithm to classify patterns, represented by arbitrary sequences of both analog input (e.g. field length) and binary input (e.g. key field or not), into different categories. Figure 2 shows a typical self-organizing map network architecture, with one input layer of N nodes on the left side and one output layer of M nodes on the right side. N is the number of discriminators compared and M is the number of the categories established. Ideally M is the number of distinct attributes. This is the only "human effort" needed in the categorization process. A "first guess" for M can be determined for some DBMS's using primary key/foreign key relationships: $M = total\_attributes - foreign\_key\_attributes$. It views patterns as points in N-dimensional feature space. Patterns that are similar in some respect cluster closer to one another in the N-dimensional space than whose patterns that are dissimilar. For the details of the algorithm, please see [Koh87].

For example, in Figure 2, we may use field specifications such as length, key field or not, value constraint, data type, and statistics of data contents such as average as input characteristics of patterns. According to these input characteristics, the classifier clusters patterns into M clusters. "Employee.id#", "Dept.employee", and "Payroll.SSN" are clustered into one category since their input characteristics (and real world meanings) are close to each other.

Figure 3 is the classifier output for an IBM AS/400 field reference file. The values of cluster center weights on the right stand for data type [5], length, value

Category	Pattern numbers	Cluster center weights
1	0 6 7	1 0.133 0.0 0 0 0.5
2	1 8	1 0.750 0.0 0 0 0.5
3	2 23 24	0 0.100 0.0 0 0 1.0
4	3 4	0 0.067 0.0 0 0 0.0
5	5 9 10	1 0.017 0.2 0 0 0.5
6	11 12 13 14 15 16 17 18 19 20	1 0.833 0.0 0 0 0.5
7	21 22 40	0 0.167 0.0 0 0 0.5
8	25 26 27 28 29 30 31 32 33 34 35 36 37 38	1 0.250 0.0 0 0 0.5
9	39	1 0.050 0.0 0 0 0.5

Figure 3: Output of Classifier for an AS/400 Field Reference File with Threshold of 0.1

[5] AS/400 only supports decimal/non-decimal as data types; we map this to a binary discriminator.

constraint, range constraint, foreign key constraint, and EDTCDE/EDTWRD specification. The classifier placed 41 attributes (pattern numbers from 0 to 40 as shown in the middle column) into 9 categories (numbers from 1 to 9) according to their schema information. For example, the patterns (attributes) 1 and 8 both record the telemarketing activity descriptions, so they are clustered together as category 2. In Figure 4, the weights of the cluster centers have been tagged with target results (category numbers) into training data. This is the training data for the category learning and recognition algorithm.

```
1 0.133 0.0 0 0 0.5 -->  Cluster center 1
1 0 0 0 0 0 0 0 0   -->  Target result
1 0.750 0.0 0 0 0.5 -->  Cluster center 2
0 1 0 0 0 0 0 0 0   -->  Target result
    .........
    .........

1 0.050 0.0 0 0 0.5 -->  Cluster center 9
0 0 0 0 0 0 0 0 1   -->  Target result
```

Figure 4: Training Data for Back-Propagation Network for an AS/400 Field Reference File

4.2 Category Learning and Recognition Algorithm

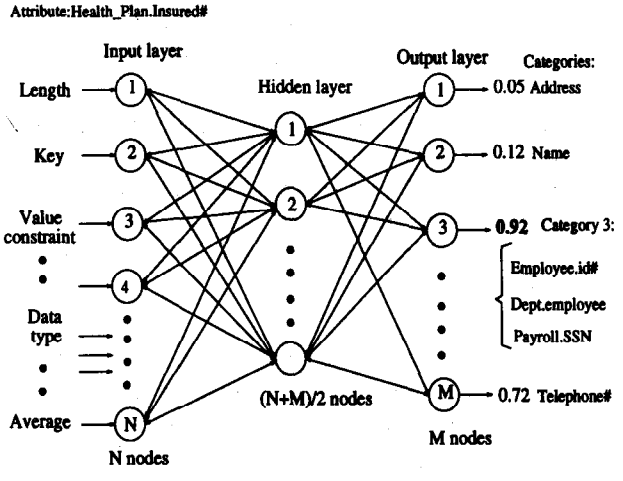

Figure 5: Back-Propagation Network Architecture

The back-propagation learning algorithm is a *supervised* learning algorithm, in which target results are provided. It has been used for various tasks such as pattern recognition, control, and classification. Here we use it as the training algorithm to train a network to recognize input patterns and give degrees of similarity. Figure 5 shows a three-layer neural network [6] for recognizing M categories of patterns. The structure of the network is designed as follows: There are N nodes in the input layer on the left, each of which represents a discriminator. The hidden layer consists of (N+M)/2 [7] nodes in the middle. The output layer (on the right side) is composed of M nodes (M categories). The tagged data generated by the classifier (Figure 4) is used as training data.

During training, the network changes the weights of connections between nodes so that each node in the output layer generates its target result (corresponding category number). The forward propagation (generating output), error calculation (computing the difference between the actual output and target output), and backward propagation (changing weights based on the errors calculated) continue until the errors in the output layer are less than the threshold.

For the AS/400 field reference file training data shown in Figure 4, we train the network do the following: when we present "1 0.133 0.0 0 0 0.5" (cluster center weights of category 1), the network outputs "1 0 0 0 0 0 0 0 0", the target result, which indicates category 1. When we present "1 0.750 0.0 0 0 0.5" (cluster center weights of category 2), the network outputs "0 1 0 0 0 0 0 0 0", which indicates category 2. After training, the network encodes data by matching each input pattern to the closest output node and giving the similarity between the input pattern (of another database) and each category (we use to train the network).

As an example take the result of the classifier in Figure 2 that clustered "Employee.id#", "Dept.employee", and "Payroll.SSN" into one category. The weights of these cluster centers are then tagged to train the network in Figure 5. After the back-propagation network is trained, we present it with a new pattern of N characteristics, attribute "health_Plan.Insured#". This network determines the similarity between the input pattern and each of the M categories. In Figure 5, the network shows that the input pattern "Insured#" is closest to the category 3 (id numbers) (similarity=0.92), and then category M (telephone#) (similarity=0.72). It also shows the input pattern is not similar to either the category 1 (Address), or category 2 (Name) since the similarity is low (0.05 and 0.12).

4.3 Semantic Integration Procedure

Figure 6 shows the diagram of our semantic integration procedure. Note the only human input is to give the threshold for the classifiers (once per database) and to examine and confirm the output results (similar pairs of attributes and the similarity between them).

[6] The computing time will increase as more layers are added.

[7] The number of nodes in the hidden layer can be arbitrary. However, (N+M)/2 nodes gave the shortest training time in our experiments.

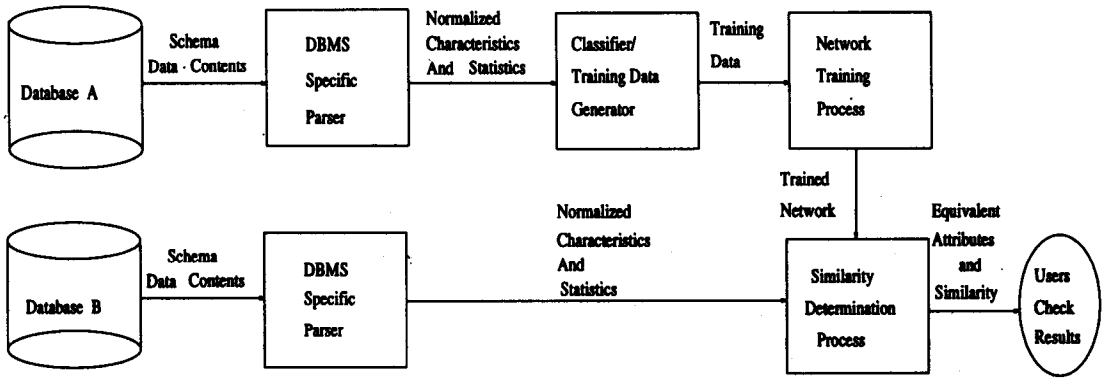

Figure 6: Semantic Integration Procedure

Step 1: Use DBMS specific parsers [8] to get information from the databases to be integrated. The system transforms this information into a common format. The output of these parsers include schema information, statistics of data values, and types of characteristics available. This is done once per database.

Step 2: The system creates a self-organizing map network with N nodes at the input layer (as shown in Figure 2). Use information of database A as input for the self-organizing network just created. This network classifies the attributes in database A to M categories. M is not pre-determined; the number of categories (clusters) created depends on the threshold set by the system trainer (e.g. the DBA of that database). Note that M (with sufficient distinguishing data) should be the number of distinct attributes in the databases, that is the attributes that do not have a foreign key relationship to other attributes in the database. The output for this step (as shown in Figure 3) is the number of categories (M) and the weights of cluster centers (M vectors of N weights). The weights of the cluster centers are then tagged as training data (as shown in Figure 4) to train the network created in the step 3.

Step 3: The system creates a three-layer back-propagation learning network with N nodes at the input layer, M nodes at the output layer, and (N+M)/2 nodes at the hidden layer (as shown in Figure 5). During the training, the network changes its weights so that each node in the output layer represents a cluster. Note that the steps 1-3 must be performed only once per database. We can then integrate with multiple DB's without retraining.

[8] We are currently developing parsers which extract information from Ingres, Oracle, and IBM AS/400 databases. The schema specification and data content statistics used in this paper were extracted manually.

Step 4: The input for the network trained in the step 3 is the attribute information of another database (database B). The network then gives the similarity between the input attribute of database B and each category of database A.

Step 5: System users check and confirm the output results of the trained network. The output results include lists of similar attributes and the similarity between them. This is the only human input required on subsequent integrations with other databases.

The are several reasons for using the classifier as the first step of this semantic integration system:

1. Ease of training. Ideally M can be determined from the DBMS using foreign key relationships. Classification lowers the number of nodes in the back propagation network output layer. It reduces the computational complexity as well as the training time.

2. After the attributes of database A are classified into M categories, the attributes of database B are compared with these cluster centers instead of each attribute of database A, which is less computationally expensive.

3. Networks can not be trained with a training set in which there are two identical answers to a question and one is correct while another is not (more discussion in section 5.4). This happens when the discriminating information does not distinguish two attributes (for example, employee.id# and payroll.SSN, which represent the same information). The classifier detects cases where this is true, and groups them to one cluster. Otherwise, the network can not be trained to point out they are different.

5 Experimental Results

We tested this technique using three pairs of existing databases. One pair were transaction data of British pound call options and put options from March 2, 1983 to June 27, 1985. The available information for these databases is only data contents. However, we extracted some schema specification information from the contents. The second pair were IBM AS/400 databases; we used the field reference files to obtain schema information (the data contents were not available.) The third pair were Sybase databases, for which we also had only schema information. [9] We would like to test integration of databases from different DBMS's, however we have not been able to obtain suitable test databases. We tested our system on integrating one IBM AS/400 database with another Sybase database (which contain no common information). The average similarity is 0.006 and the maximal similarity is 0.021. The result shows that attributes of these two databases are totally different (as expected). However, we do not know if similar attributes in different DBMS's would be detected; we need additional data to test this.

5.1 Transaction Data of British Pound Call Option and Put Option

The option transaction data used in this experiment was the British pound call option and put option price quotes at the Philadelphia Exchange between March 2, 1983 and June 27, 1985. There are nine attributes in each database (Figure 7). The schema information we used includes data types, the number of decimals, field length, and existence of null value. The numerical statistics we used were maximum value (MAX), minimum value (MIN), coefficient of variance (CV), standard deviation (SD), and average. Figure 7 shows a sample output of similarity between an attribute, call option spot bid price, and each attribute of the put option database. As we can see, the call option spot bid price is similar to both put option spot bid price (similarity = 0.953933) and spot ask price (similarity = 0.949726). Spot bid prices and attribute spot ask prices are actually very similar, generally moving in "lock step".

The difference between using schema information and using data contents to determine degrees of similarity is that the data contents may change over time. In the British pound option databases we used in this experiment, the spot bid prices, ask prices, and volumes traded changed over time. Besides, there were some abnormal trades during the first few weeks of data (the British pound option trade started at the Philadelphia

[9] These two pairs of databases were used in our prior work [LC93]. The results here are substantially better than the previous work .

```
File: Call_Option          File: Put_Option
Attribute: Spot_bid_price

 The normalized input:     The output similarity:
----------------------    -------------------------
   Char type: 0.000000   *Spot_bid_price:   0.953933
Numeric type: 1.000000   *Spot_ask_price:   0.949726
   Date type: 0.000000
   Time type: 0.000000    Expiration_price:0.047791
     Length: 0.681892     Option_price:    0.015392
    Decimal: 0.400000     Volume:          0.001423
       Null: 0.000000     Transaction_time:0.000383
    Average: 0.503130     Transaction_date:0.000229
        Max: 0.503993     Maturity_date:   0.000017
        Min: 0.502583     Symbol:          0.000001
         SD: 0.500305
         CV: 0.524235
```

Figure 7: The Similarity between Call Option Spot Bid Price and all categories of Put Option Database

Exchange in March 1983). This phenomenon holds true in many databases. When a database is just established, the data may be transferred from other databases or input manually. The amount of data inserted is higher than normal amount. Also, faulty data may happen more often because of the huge amount of manually inserted initial data or the process of transferring data from one database to another database.

We are pleased by the result that our system can discriminate "expiration_price" from "bid_price" and "ask_price". Without the statistics of data contents, we would not able to do it since their schema specifications are the same. With the statistics of data contents, we discriminate "expiration_price" from "bid_price" and "ask_price" by their different averages, MAX, MIN, coefficient of variance (CV), and standard deviation (SD). The average price of "expiration_price" is lower and "expiration_price" is more stable than "spot_bid_price" and "spot_ask_price". Table 1 shows the average, maximum, and minimum similarities for both the attributes we determined are similar (should be merged) such as "call expiration_price" and "put expiration_price", and those that aren't equivalent (such as "transaction date" and "maturity date"). In this table we have classified the (arguably equivalent) bid and ask prices into the same cluster; given this there is a clear break between the two groups. In

Table 1: Statistics of Similarity for British Pound Option Transaction Databases

Equivalent pairs	Avg	Max	Min
Degree of similarity:	0.96921	0.98869	0.93299
Non-equivalent pairs	**Avg**	**Max**	**Min**
Degree of similarity:	0.00623	0.05989	0.0000

this experiment, if we simply trust that all attributes with similarity less than 0.1 are different, and all with a *single match* with similarity greater than 0.9 are the same, we would be left with only the bid and ask prices to manually inspect.

5.2 AS/400 Databases

The second pair of sample databases were provided by M.I.S. Systems, Chicago, Illinois. They are two field reference files running on IBM AS/400. The characteristics available for us to use are data type, length, range and value constraints, and EDTCDE and EDTWRD specifications [10]. There are 10 fields in a database that record general marketing activity information and 41 fields in a database that record telemarketing activity information. Thus, there are 410 pairs of fields being compared.

Here we first used the classifier to classify the database with 41 fields into 9 categories (as shown in Figure 3) by setting the threshold to 0.1. .9 is the number of distinct categories (M) in the database (e.g. one category with attributes recording dates to call customers and one category with attributes recording the types of telemarketing activities). We then used the back-propagation learning algorithm to train a three-layer network with 6 nodes in the input layer, 7 nodes in the hidden layer, and 9 nodes in the output layer, to recognize these categories. After the network was trained, the database with 10 attributes was presented to this trained network. The network then determined the similarity between these 10 fields and the 9 categories. Thus, there are only 90 comparisons needed instead of 410 comparisons without using a classifier.

Table 2 shows the average similarity for equivalent attributes is 0.98905 while the average similarity for non-equivalent is 0.00161. The network is very effective in recognizing equivalent attributes and discriminating non-equivalent attributes. These results are much better than the results in [LC93]. This is because the probability values of similarity are not pre-programmed, but are determined *for that database* by training.

Table 2: Statistics of Similarity for AS/400 Databases

Equivalent pairs	Avg	Max	Min
Degree of similarity:	0.98905	0.99591	0.98360
Non-equivalent pairs	**Avg**	**Max**	**Min**
Degree of similarity:	0.00161	0.00801	0.00000

[10] The IBM AS/400 databases have all the design information described in Section 3 available on the system. However, some of this information was not available to us.

5.3 Sybase databases

The third pair of databases were Sybase databases provided by Reuters Information Systems in Oak Brook, Illinois. The available characteristics of these databases include data type, length, key, value and range constraints, file access restrictions, foreign key constraints, and null value constraints. There are 7 attributes in the database that record project document information and 6 attributes in the database record project meeting information. Thus, there are 42 pairs of fields being compared. In the pattern recognition network, there are 13 nodes in the input layer, 10 nodes in the hidden layer, 6 nodes (6 categories, 1 attribute in each category since they present different meanings) in the output layer. Again, the network is very effective in both discriminating pairs of non-equivalent attributes (average similarity = 0.00924) and matching equivalent pairs of attributes (average similarity = 0.93254). In this experiment, the trained neural network shows its ability of generalization to correctly response to patterns not used to trained it. Note that one pair of equivalent attributes is given similarity of 0.87616 while another pair of equivalent is given 0.98892 (the second row in Table 3). This pair of equivalent attributes of similarity of 0.87616 represent the same real-world meaning even though their schema specifications are slightly different. The network is still able to identify this pair of equivalent attributes.

The maximal similarity for non-equivalent pairs is 0.07240; that is, some similarity between some pairs of non-equivalent attributes. The gap between minimal similarity of equivalent attributes (0.87616) and maximal similarity of non-equivalent pair (0.07240) indicates the effectiveness of our technique.

Table 3: Statistics of Similarity for Sybase Databases

Equivalent pairs	Avg	Max	Min
Degree of similarity:	0.93254	0.98892	0.87616
Non-equivalent pairs	**Avg**	**Max**	**Min**
Degree of similarity:	0.00924	0.07240	0.00000

5.4 System Performance

This system is implemented in C; our tests were run on an IBM RS/6000 model 220. CPU time needed for the classification or recognition processes finished in under 0.1 second. The training in our experiments always took less than 7 seconds of CPU time when we set the thresholds to 0.01. The training time can be reduced as larger thresholds are used. For the British pound option transaction databases, the training time

Table 4: System Performance and Human Effort Needed

Process	CPU time	Human effort
Extract information from databases	Varies	Little (once per DBMS)
Classify attributes into categories	< 0.1 second	Little (give threshold, once per database)
Train networks	< 7.0 seconds	None
Determine similarity	< 0.1 second	Some (check results)

is 2.2 seconds CPU time (704 epochs [11]) with a classifier. Without a classifier to group similar attributes into a single cluster, the training time needed would be much longer (11642 epochs and 33.6 seconds CPU time). The reason is that networks can not be trained with a training set in which there are two different expected output for the same input pattern (two different attributes with the same characteristics). This is also the reason why training takes much longer using two very similar input patterns.

6 Conclusions and Future Work

In this paper we show a system in which the human effort needed for semantic integration is minimized. We use a self-organizing classifier algorithm to categorize attributes and then use a back-propagation learning algorithm to train a network to recognize input patterns and determine similarity between attributes. The experimental results show the effectiveness and efficiency of our technique.

From Table 4, we learn that the only potentially slow process is extracting information from databases, which can be done as a "batch" process once per DBMS. The rest of the method is fast enough to be done interactively. After the information is extracted, system users can "interact" with the system (provide the threshold during the classification process and check and confirm the results: similar attributes identified by the system) to integrate databases. Based on the three experiments we presented, a user could simply follow the rule "if the similarity is less than 0.1, the attributes are different; if there is one cluster with similarity greater than 0.8 then the attribute matches that cluster". The only "manual inspection" needed would be for the spot bid price and spot ask price in the British pound option databases, and even here following the closest match would give the proper result. Further experiments are needed to determine exactly what this rule should be, and how often users will need to inspect the results. To this end we have made this tool available (as C source code) via anonymous ftp from eecs.nwu.edu in directory /pub/semint. We are developing parsers for the IBM AS/400, Ingres, and flat files; we will develop more as time and resources allow. We would appreciate feedback on the effectiveness of this tool.

Our method does have one problem when dealing with multiple types of Database Management Systems. In order to ease the presentation of the method we have glossed over this, however we must note that the back-propagation algorithm requires that input patterns have the same discriminators used in the training. We may not be able to obtain the same list of discriminators from two different DBMSs (for example, the data types supported may be different). In this case, we must train a new network using the intersection of the available discriminators. However, the intersection/filtering process can be automated, and the expected result of the classifier (M) is the same regardless of the discriminators provided. Thus no extra human effort is needed to train multiple networks (provided the intersection of the discriminators is sufficient to distinguish the attributes in a single database). We are investigating alternatives to back propagation that will allow us to use a single network regardless of the discriminators available.

We have concentrated on matching attributes (the **heterogeneity of organizational models** problem of [Wie93]). We are also investigating use of this technique in uncertain data query and join operations by attacking some of the other levels of heterogeneity described in [Wie93]. We want to extend these research results to more general areas such as intelligent integration of information (I3). The I3 objective requires the establishment of a conceptual information system architecture, so that many participants and technologies can contribute [Wie93]. Automation becomes important as the volume of accessible data increases. The ability to "discover" heuristics in our technique makes it favorable for automation. Our technique can also be combined with [DeM89] to support operations over partially matched domain and uncertain data queries [TC93] in heterogeneous databases.

[11] One cycle of forward-propagation, backward-propagation, and weight-update is called one epoch.

References

[DeM89] Linda G. DeMichiel. Performing operations over mismatched domains. In *Proceedings in the 5th International Confer-*

ence on Data Engineering, pages 36–45. IEEE, May 1989.

[DH84] Umeshwar Dayal and Hai-Yann Hwang. View definition and generalization for database integration in multidatabase system. *Transactions on the Software Engineering*, SE-10(6):628–644, November 1984.

[DKM+93] P. Drew, R. King, D. McLeod, M. Rusinkiewicz, and A. Silberschatz. Report of the workshop on semantic heterogeneity and interoperation in multidatabase systems. *SIGMOD Record*, pages 47–56, September 1993.

[HR90] Stephen Hayne and Sudha Ram. Multi-user view integration system (MUVIS): An expert system for view integration. In *Proceedings in the 6th International Conference on Data Engineering*, pages 402–409. IEEE, February 1990.

[Koh87] Teuvo Kohonen. Adaptive, associative, and self-organizing functions in neural computing. *Applied Optics*, 26:4910–4918, 1987.

[LC93] Wen-Syan Li and Chris Clifton. Using field specification to determine attribute equivalence in heterogeneous databases. In *Third International Workshop on Research Issues on Data Engineering: INTEROPERABILITY IN MULTIDATABASE SYSTEMS*, pages 174–177, Vienna, Austria, April 18-20 1993. IEEE.

[Lit89] Witold Litwin. *MSQL: A Multidatabase Language.* Elsevier Science Publishing, 1989.

[LNE89] James A. Larson, Shamkant B. Navathe, and Ramez Elmasri. A theory of attribute equivalence in database with application to schema integration. *Transaction on Software Engineering*, 15(4):449–463, April 1989.

[MB81] Amihai Motro and Peter Buneman. Constructing superviews. In *Proceeding of the SIGMOD International Conference on Management of Data*, pages 56–64, Ann Arbor, Michigan, April 1981. ACM.

[MH80] Dennis McLeod and Dennis Heimbigner. A federated architecture for database system. In *Proceedings of the National Computer Conference*, pages 283–289, Anaheim, CA, May 1980. AFIPS.

[NB86] S. Navathe and Peter Buneman. Integrating user views in database design. *Computers*, 19(1):50–62, January 1986.

[SBU+81] J. M. Smith, P. A. Bernstein, U.Dayal, N.Goodman, T. Landers, T. Lin, and E.Wang. Multibase - integrating heterogeneous distributed database systems. In *Proceeding of the National Computer Conference*, pages 487–499. AFIPS, 1981.

[SG88] V. C. Storey and R. C. Goldstein. Creating user views in database design. *Transactions on Database Systems*, pages 305–338, September 1988.

[SG89] Amit Sheth and Sunit K. Gala. Attribute relationships: An impediment in automating schema integration. In *Proceedings of the NSF Workshop on Heterogeneous Database Systems*, Evanston, IL, December 1989.

[SL90] Amit Sheth and James Larson. Federated database systems for managing distributed heterogeneous, and autonomous databases. *Computer Surveys*, 22(3):183–236, September 1990.

[SLCN88] Amit Sheth, James Larson, A. Cornelio, and S. B. Navathe. A tool for integrating conceptual schemas and user views. In *Proceedings of the 4th International Conference on Data Engineering*, Los Angeles, CA, February 1988. IEEE.

[TC93] Pauray S.M. Tsai and Arbee L.P. Chen. Querying uncertain data in heterogeneous databases. In *Third International Workshop on Research Issues on Data Engineering: INTEROPERABILITY IN MULTIDATABASE SYSTEMS*, pages 161–168, Vienna, Austria, April 18-20 1993. IEEE.

[VH94] Vincent Ventrone and Sandra Heiler. Some advice for dealing with semantic heterogeneity in federated database systems. Submitted to *International Journal of Computer-Aided Engineering*, 1994.

[Wie93] Gio Wiederhold. Intelligent integration of information. *SIGMOD Record*, pages 434–437, May 1993.