ELSEVIER

Data & Knowledge Engineering 33 (2000) 49–84

DATA & KNOWLEDGE ENGINEERING

www.elsevier.com/locate/datak

SEMINT: A tool for identifying attribute correspondences in heterogeneous databases using neural networks ☆

Wen-Syan Li [a,*,1], Chris Clifton [b,2]

[a] *C&C Research Laboratories, NEC USA, Inc., 110 Rio Robles, MS SJ100 San, Jose, CA 95134, USA*
[b] *The MITRE Corporation, M/S K308, 202 Burlington Road, Bedford, MA 01730-1420, USA*

Received 23 February 1999; received in revised form 23 September 1999; accepted 11 November 1999

Abstract

One step in interoperating among heterogeneous databases is semantic integration: Identifying relationships between attributes or classes in different database schemas. SEMantic INTegrator (SEMINT) is a tool based on neural networks to assist in identifying attribute correspondences in heterogeneous databases. SEMINT supports access to a variety of database systems and utilizes both schema information and data contents to produce rules for matching corresponding attributes automatically. This paper provides theoretical background and implementation details of SEMINT. Experimental results from large and complex real databases are presented. We discuss the effectiveness of SEMINT and our experiences with attribute correspondence identification in various environments.

Keywords: Heterogeneous databases; Database integration; Attribute correspondence identification; Neural networks

1. Introduction

Advances in information technology have lead to an explosion in the number of databases accessible to a user. Although these databases are generally designed to be self-contained, serving a limited application domain, the ability to integrate data from multiple databases enables many new applications. For example, corporate decision-makers have realized that the huge amount of information stored in information systems is not only operational data; but an asset that can, when integrated, be used to support decision-making. However, many of these databases are heterogeneous in a variety of aspects because they were initially developed individually for departmental purposes without consideration for information sharing and collaboration.

☆ This material is based upon work supported by the National Science Foundation under Grant No. CCR-9210704.
* Corresponding author. Tel.: +1-408-943-3008; fax: +1-408-943-3099.
E-mail addresses: wen@ccrl.sj.nec.com (W.-S. Li), clifton@mitre.org (C. Clifton).
[1] The work described in this paper was performed when the first author was obtaining his Ph.D. at Northwestern University, Department of EECS.
[2] The views and opinions in this paper are those of the author and do not reflect MITREs work position.

0169-023X/00/$ - see front matter
PII: S0169-023X(99)00044-0

Enterprises may also have various kinds of databases due to company mergers or the introduction of new database technology. By integrating these databases, the differences of DBMS, language and data models can be suppressed; users can use a single high-level query language (or interface) to access multiple database systems [20]; and integrated views of enterprise-wise information can be provided.

Early work in heterogeneous database integration focused on procedures to merge individual schemas into a single global conceptual schema, such as described in [20,37]. The amount of knowledge required about local schemas, how to identify and resolve heterogeneity among the local schemas, and how changes to local schemas can be reflected by corresponding changes in the global schema are major problems with this approach because of the complexity of a global schema.

The federated database approach [18,23] resolves some of the problems associated with a global schema addressed above. Federated databases only require *partial* integration. A federated database integrates a collection of local database systems by supporting interoperability between pairs or collections of the local databases rather than through a complete global schema.

Multidatabase language approach, such as MSQL [8,17], is another attempt to resolve some of the problems associated with a global schema. With these systems, no global schema (not even a partial one) is maintained. This approach puts the integration responsibility on users by providing functionality beyond standard SQL to allow users to specify integration information as part of the query. This is a heavy burden to place on users. We instead view the multidatabase language as an intermediate language, to be used for query processing after schema integration issues have been resolved. This approach is also used in [45], that uses an attribute correspondence table to convert queries into a multidatabase-like intermediate form.

1.1. Problem statement

In either approach to database integration, the steps of database integration include extracting semantics, transforming formats, identifying attribute correspondence, resolving and modeling heterogeneity, multidatabase query processing, and data integration. In order to answer queries in multidatabase systems three distinct processes need to be performed by the user, database administrator, and/or system as shown in Fig. 1. The Schema Integration process includes a possible schema transformation step, followed by correspondence identification, and an object integration and mapping construction step [23]. In Query Processing step, global queries are reformulated into sub-queries, the sub-queries are executed at the local sites, and their results are assembled at a final site. The Data Integration process is complimentary to Query Processing step, i.e., it determines how the results from different local databases should be merged, or presented, at the final site.

Sheth and Kashyap [33] argued that identifying semantically related objects and then resolving the schematic differences is the *fundamental* question in any approach to database system interoperability. In all three major approaches of database integration, identifying attribute correspondences is the essential step before any query can be answered. The dominant computer paradigm will involve a large number of information sources that are heterogeneous in semantics and format. Some large organizations have huge databases in terms of the number of attributes and the number of rows. One example is the tooling database at Boeing Computer Services (BCS), that contains hundreds of attributes and millions of rows. Many of these databases were designed

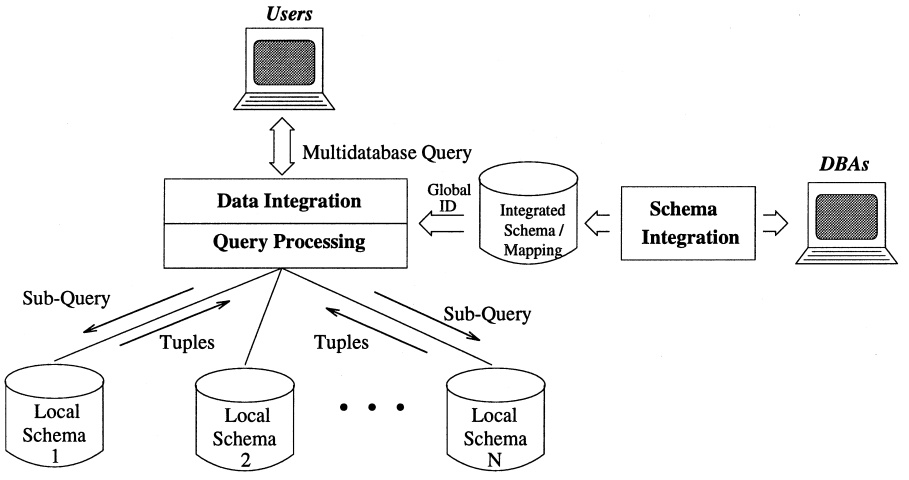

Fig. 1. Multidatabase query processing.

by different people for various purposes; as a result, the formats of data and its semantics presentation are not expected to be standardized.

US West reports having five terabytes of data managed by 1000 systems, with customer information on 200 databases [5]. Manually comparing all possible pairs of attributes is an unreasonably large task. One group at GTE began the integration process with 27,000 data elements, from 40 of its applications. It required an average of 4 h per data element to extract and document matching elements when the task was performed by someone other than the data owner [39]. In such complicated computing environments and with such large numbers of databases and attributes, understanding the semantics of each individual database and integrating these heterogeneous databases are extremely difficult tasks.

We focus on the problem of identifying corresponding attributes in different DBMSs that reflect the same real-world class of information. Currently, there is little technology available to 'automate' this process. Integrating semantically heterogeneous databases can be costly since it is an intellectual process. Drew et al. [5] pointed out many 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 reuse of knowledge gained in the semantic heterogeneity resolution process.

The goal of our research is to develop a semi-automated [3] semantic integration procedure that utilizes the metadata available in the database systems to identify attribute correspondences. We see information easily available to an automated tool as including attribute names, schema design, constraints and data contents. Parsing application programs is not practical, and 'picking the brains of users' would be too interactive for our goal of a semi-automated system. We want the

[3] We feel that a fully automated procedure is not realistic with current technologies.

ability to automatically determine the likelihood of attributes referring to the same real-world class of information from the metadata. There are many existing systems where how to determine the attribute correspondences is 'pre-programmed'. We feel this type of procedure is ad hoc. Since these databases are heterogeneous, the 'formula' works for one database integration problem may not work for others. We also desire to have the ability to reuse or adapt the knowledge gained in the semantic heterogeneity resolution process to work on similar problems. Another essential feature is automation. Automatic process of extracting and transforming data semantics becomes important as the volume and variation of accessible data increases.

1.2. Semantic integration in SEMINT

We present a method that utilizes metadata available in databases for attribute correspondence identification. Attributes in different databases that represent the same real-world concept will likely have similarities in schema designs, constraints, and data value patterns; these similarities allow us to identify correspondences. For example, employee salaries in two databases will probably be similar in both schema design and values; on the other hand, employee salaries are different from addresses. We see three levels of metadata that can be automatically extracted from databases and used to determine attribute correspondences: attribute names (the dictionary level), schema information (the field specification level), and data contents and statistics (the data content level).

Most existing work focuses on using attribute names to determine attribute correspondences. However, synonyms occur when objects with different names represent the same concepts, and homonyms occur when the names are the same but different concepts are represented. From GMs efforts in integration [24], attribute names were not sufficient for semantic integration; only a few obvious matches were found. However, similarities in schema information were found to be useful. For example, it was discovered in one case that attributes of type char(14) were equivalent to those of char(15) (with an appended blank).

Our semantic integration procedure utilizes schema design information (schema and constraints) and data contents (data patterns and statistics) as clues to the semantics of the data. We use neural networks to *learn* how this metadata characterizes the semantics of the attributes *in a particular domain*. This gives us a procedure that automatically incorporates domain knowledge from metadata extracted directly from the database. This approach can cooperate with other approaches based on naming techniques and ontological engineering to take advantage of whatever information is available. In our method, the knowledge of how to determine matching data elements is *discovered* from the metadata directly, not *pre-programmed*.

In the example shown in Fig. 2, we want to integrate **Faculty** and **Student** databases. SEMantic INTegrator (SEMINT) first uses DBMS specific parsers to extract metadata (schema design, constraints and data content statistics) from these two databases. The metadata forms 'signatures' (patterns) describing the attributes in the **Faculty** and **Student** databases. These attribute signatures are used as training data for neural networks to recognize these signatures (thus, attributes as well). The trained neural network can then identify corresponding attributes, based on the metadata (signatures) of attributes in, **Faculty** and **Student** databases and point out their similarity. In our approach (see Fig. 2) the metadata extraction is automated; and how to match corresponding attributes and determine their similarity is 'learned' during the training process directly from the metadata.

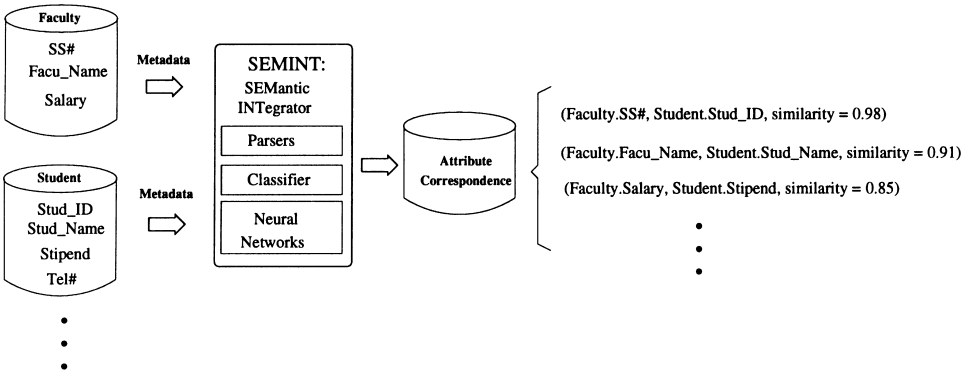

Fig. 2. Overview of semantic integration in SEMINT.

1.3. Paper organization

The rest of this paper is organized as follows. In Section 2, we review existing work in this area. Section 3 presents details of our attribute correspondence identification method and the design of SEMINT, including a detailed discussion of the metadata that can be automatically extracted from databases, and how we use it to identify attribute correspondences. Section 4 describes some experimental results on very large and complex real databases and discusses the effectiveness of SEMINT based on these experiences in various environments. We also highlight some applications where SEMINT is used as a component. Finally, we conclude this paper with a discussion of areas for future work.

2. Related work

In [11] Larson et al. discussed metadata characteristics and theory of attribute equivalence to schema integration. We see three approaches to determining attribute correspondences based on types of metadata used: Comparing attribute names (at the dictionary level), comparing field specifications (at the schema level), and comparing attribute values and patterns (at the data content level).

2.1. Comparing attribute names

This approach assumes that the same attribute may be represented by synonyms in different databases. Systems have been developed to automate database integration. One that has addressed the problem of attribute correspondences is Multi-User View Integration System (MUVIS) [9]. MUVIS is a knowledge-based system for view integration in an object-oriented development environment. It assists database designers in representing user views and integrating these views into a global conceptual view. The similarity and dissimilarity in MUVIS is primarily based on *attribute names*.

Clement Yu et al. [44] argued that semantics of majority of attributes can be captured by consulting pre-established concepts that are analogous to standardized keyword index hierarchies. Then, the aggregation among attributes can be determined.

A. Sheth and V. Kashyap [33] introduced the concept of semantic proximity to specify degrees of semantic similarity among objects based on their real world semantics. Heuristics to identify possible semantic similarity between objects that have various types of schematic difference and data inconsistency were also proposed.

Existing systems using this approach work by consulting a synonym lexicon, such as WordNet [19]. However, consulting a synonym lexicon has limitations since database designers tend to use abbreviations for attribute names (e.g., Emp for Employee, SSN or SS# for Social Security Number, etc). Additionally, in some cases, attribute names are actually abbreviations of phrases (e.g., DOB for date of birth and W_Phone for work phone number). These types of relationships cannot be easily identified by consulting a synonym lexicon. Another problem with this method is *homonyms*, where the names are the same but different concepts are represented.

The DELTA project at MITRE [1] has developed a tool based on string matching to assist in integration. This goes beyond simple attribute names, and looks at the entire data dictionary (including comments and field descriptions). However, this still faces many of the problems discussed above.

2.2. Comparing field specifications at the schema level

Approach of comparison of attribute specification using design information has also been proposed. In [21], the characteristics of attributes discussed are uniqueness, cardinality, domain, static semantic integrity constraints, dynamic semantic integrity constraints, security constraints, allowable operations and scale. However, [21] focused on the theory of attribute equivalence to schema integration rather than utilizing these characteristics to determine attribute correspondences.

Li and Clifton [13] presented a technique that utilizes field specifications to determine the similarity and dissimilarity of a pair of attributes. This technique (over simply comparing field names or using attribute domain relationships) include:

1. the effect of synonyms and homonyms [21] will be different (the problem is with structural synonyms and homonyms, rather than dictionary synonyms and homonyms); and
2. it provides a solution for the time-consuming problem of determining the attribute (domain) relations in [11,31,32,34,35] by eliminating most of the non-equivalent attribute pairs; simply using schema information.

The technique described in [13] does not intend to completely replace searching through a synonym lexicon, but helps to determine attribute correspondences when no conclusion can be made simply by searching a synonym dictionary. It can also be used with other approaches, as a 'first step' to eliminate most of clearly incompatible attributes. This allows the process of comparing attribute domains, that is more computationally expensive, to work on a smaller problem. However, the accuracy of results depends on the field specification information availability. One weakness of the technique presented in [13] is the rules to match attributes are 'pre-programmed' by DBAs, as in MUVIS [9].

2.3. Comparing attribute values and patterns in the data content level

The third approach of determining attribute correspondences is comparing attribute values. Larson et al. [11,21] and Sheth et al. [35] discussed how relationships and entity sets can be in-

tegrated primarily based on their domain relationships: EQUAL, CONTAINS, OVERLAP, CONTAINED-IN and DISJOINT. Determining such relationships can be time consuming and tedious [34]. Another problem is the ability to handle 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 [35], 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 that can be integrated. Sheth and Gala [32] 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.

Li and Clifton [14] proposed a method that utilizes data patterns and distributions rather than data values and domains. This approach has better fault tolerant ability and less time-consuming since only a small portion of data values are needed by employing data sampling techniques.

Other work that uses data patterns to determine attribute correspondences includes [36,38] that uses historical update information for instance identification in federated databases.

3. Attribute correspondence identification procedure

In this section, we give technical detail of SEMINT. We start with the choice of using neural network techniques, rather than fixed rules, for matching corresponding attributes.

3.1. Neural networks versus traditional programmed computing

Programmed computing is best used in 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 that solves the problems of identifying attribute correspondences. This is because that attribute relationships are usually fuzzy and the availability of database information may vary.

Neural networks have emerged as a powerful pattern recognition technique due to the development of more powerful hardware and efficient algorithms [29] and have been used in a wide range of application [40]. Neural networks can learn the similarities among data directly from its instances and empirically infer solutions from data without prior knowledge of regularities. Unlike traditional programs, neural networks are trained, not programmed. Neural networks act on data by detecting underlying organizations or clusters. For example, the input patterns can be grouped by detecting how they resemble one another. The networks learn the similarities among patterns directly from the instances of them. That means neural networks can infer classification without prior knowledge of regularities. We use them as a bridge over the gap between individual examples and general relationships to identify corresponding attributes in heterogeneous databases.

We feel that neural networks are more suitable than traditional programs for determining attribute correspondences in heterogeneous databases since:

1. the availability of metadata and the semantics of terms may vary;
2. the relationship between two attributes is usually fuzzy;
3. it is hard to define and assign probability to rules for comparing aspects of two attributes and

4. pre-defined rules and probabilities that work for one pair of databases may not work for other pairs of databases, but need to be adjusted dynamically. Wiederhold [43] also pointed out a similar problem in digital libraries – when dealing with multiple domains no consistency of terminology can be expected or even advocated.

3.2. Overview of SEMINT

SEMINT [15] is a system for identifying attribute correspondences using neural network techniques. We have developed a graphical interactive mode (allowing users to provide known information about the semantic integration problem at hand), or it can be ran as a suite of batch programs. It has been implemented using C and Motif, and runs on IBM RS6000s under AIX and Sun workstations under Sun OS. Databases to be integrated are accessed directly using automatic 'catalog parsers' to extract metadata from target databases. Fig. 3 shows the procedure in SEMINT. First, the metadata (schema information and data content statistics) is extracted from an individual database using DBMS specific parsers. This metadata is normalized and used as input data for a self-organizing map algorithm as a classifier to categorize attributes. We then use the classifier that learns how to discriminate among attributes within 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.

We will first give details of the metadata available from databases, how we use DBMS-specific parsers to extract the metadata, and how we normalize and present these 'metadata' as input vectors for neural networks, followed by design and usage of neural networks for a classifier and a category learning and recognition tool to determine attribute correspondences.

3.3. Metadata extraction using DBMS-specific parsers

3.3.1. Metadata used in semantic integration

Drew et al. [5] pointed out the meaning of information may be embodied within a database model, a conceptual schema, application programs, and the minds of users. We see three levels of metadata that can be automatically extracted from databases: attribute names (the dictionary level), schema information (the field specification level), and data contents and statistics (the data content level). This is shown in Fig. 4. Much work in the past has focused on the metadata available in the dictionary level. However, in many cases, the approach of using attribute names does not work such as the example of GMs efforts in database integration [24] (discussed in Section 1.2).

SEMINT focuses on utilizing the metadata at the field specification level and data content level. It can cooperate with other approaches based on naming, such as techniques used in the DELTA

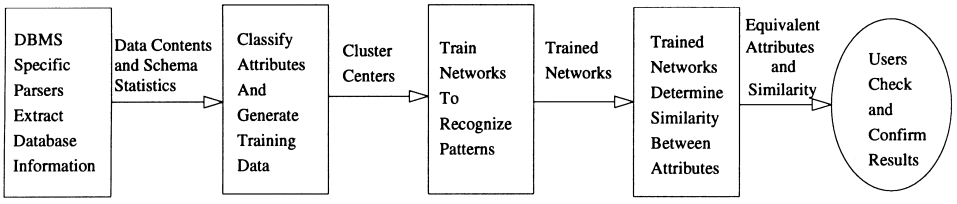

Fig. 3. Overview of attribute correspondence identification procedure using neural networks.

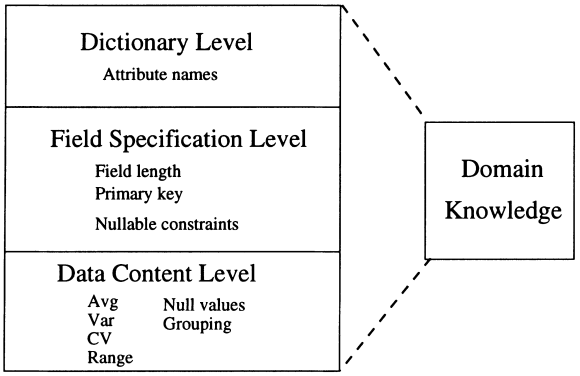

Fig. 4. Multi-level database semantics.

project at MITRE [1]. Domain specific knowledge provides SEMINT the inference ability to organize the metadata at different levels to identify possible heterogeneity. We will further discuss how domain knowledge can assist in determining attribute correspondences in Section 6.

The field specifications and data contents provide some clue to the semantics of an attribute. In particular, attributes that represent the same real-world information are likely to have similarities in field specifications and data content patterns (e.g., if Student_ID and employee_ID are both based on a US Social Security Number, they will both contain nine digits). However, we do not know just what these similarities are. Worse yet, the similarities may vary between domains as pointed out in [43]. The metadata used in SEMINT includes:

Schema information: The schema information used by SEMINT includes data types, length, scale, precision, and the existence of constraints, such as primary keys, foreign keys, candidate keys, value and range constraints, disallowing null values, and access restrictions. Sample SQL queries to extract schema information and constraints from Oracle7 databases are shown in Fig. 5. 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 are available to extract schema information from flat files.

```
SELECT a.TABLE_NAME, a.COLUMN_NAME, a.DATA_TYPE,
       a.DATA_LENGTH, a.NULLABLE, a.DATA_SCALE,
       a.DATA_PRECISION, a.DATA_DEFAULT,
       a.NUMBER_DISTINCT
   FROM USER_TAB_COLUMNS a, ALL_TAB_COLUMNS b
   WHERE a.TABLE_NAME=b.TABLE_NAME
         AND a.COLUMN_NAME=b.COLUMN_NAME
         AND b.OWNER=:owner_id;

SELECT DISTINCT CONSTRAINT_TYPE
   FROM USER_CONSTRAINTS a, USER_CONS_COLUMNS b
   WHERE a.CONSTRAINT_NAME=b.CONSTRAINT_NAME
         AND b.TABLE_NAME=:table_name;
         AND b.COLUMN_NAME=:column_name;
```

Fig. 5. SQL queries to extract schema and constraints.

Data content statistics: The data contents of different attributes tend to be different even though their schema designs may be the same. This shows up in their data patterns, value distributions, grouping or other characteristics. These can serve to characterize attributes. 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 characteristics. However, their data patterns, such as value distributions, and averages, are different. Thus, examining data contents can correct or enhance the accuracy of the outcomes from the dictionary level and the schema level. The statistics on data contents used in SEMINT include maximum, minimum, average (mean), variance, coefficient of variance, existence of null values, existence of decimals, scale, precision, grouping and number of segments. For numerical fields, we use their values to compute statistics. For character fields, whose values are not computable as ASCII code numbers, we compute statistics on number of bytes actually used to store data, rather than 'actual ASCII numbers'. SQL queries extracting data content statistics from Oracle7 databases for numeric and character fields are shown in Fig. 6. Only a small portion of sample data is needed (e.g., pick 1 row out of 10, not first 10statistics on the data contents. The size of a table can be found in the catalog, then we decide the percentage of rows we want to sample. Note that estimation techniques [22,26] can be used to speed up this process.

Other semantics: Security constraints such as read/write/grant authority specifications and 'behavior semantics' such as the use of cross-references, views, clusters, sequences, synonyms, and dependencies can also be extracted from the system data dictionary. We have incorporated some of these behavior semantics into our Oracle7 parser.

The complete list of metadata used in SEMINT is given in Table 1.

Although different DBMSs use different data dictionaries to contain schema information and integrity constraints, these DBMS specific parsers are similar. SEMINT automatically extracts schema information and constraints from the database catalogs and statistics on the data contents using queries over the data. The information extracted from different databases is then transformed into a single format and normalized. The advantages of having various DBMS specific parsers provided by SEMINT are:

- The data dictionaries of various DBMSs are different; however, they are fixed for each DBMS and pre-defined in DBA manuals. Therefore, the queries to access these systems, usually in SQL, can be pre-programmed using C with embedded SQL.

For Numeric and Date Data Types:

```
SELECT AVG(:column_name), MAX(:column_name),
       MIN(:column_name),STDDEV(:column_name),
       VARIANCE(:column_name)
   FROM :table_name;
```

For Character Data Types:

```
SELECT AVG(VSIZE(:column_name)),
       MAX(VSIZE(:column_name)),
       MIN(VSIZE(:column_name)),
       STDDEV(VSIZE(:column_name)),
       VARIANCE(VSIZE(:column_name))
   FROM :table_name;
```

Fig. 6. SQL queries to extract data content statistics.

Table 1
Metadata (items 1–15) and data content based discriminators (items 16–20) used in SEMINT

No.	Discriminator	Descriptions
1	Data length	
2	Character type	
3	Number Type	
4	Date Type	Valid dates
5	Row ID	Data type: Row pointer
6	Raw data	Raw binary of variable length
7	Check	Constraint exists on column values
8	Primary key	
9	Unique value	Value is unique but is not part of the key
10	Foreign key constraint	Column refers to key in another table
11	Check on View	
12	Nullable	Null values allowed
13	Data Precision	
14	Data Scale	
15	Default	Has Default value
16	Minimum	Minimum non-blanks for character attributes
17	Maximum	Maximum non-blanks for character attributes
18	Average	Average non-blanks for character attributes
19	Coefficient of variance	CV of non-blanks for character attributes
20	Standard deviation	SD of non-blanks for character attributes

- As DBMS specific parsers are pre-programmed, the metadata extraction process is fully automated. No user intervention is needed.
- SEMINT users are not required to be aware of the differences of various DBMSs. The heterogeneity of data dictionaries, SQL language command interfaces, functionalities, and facilities of different DBMSs are resolved by SEMINT's DBMS specific parsers.

In SEMINT, users only need to specify the DBMS types, such as Oracle, Ingres, IBM AS/400, or DB2, and supply DBMS-specific connection information for the desired database. For

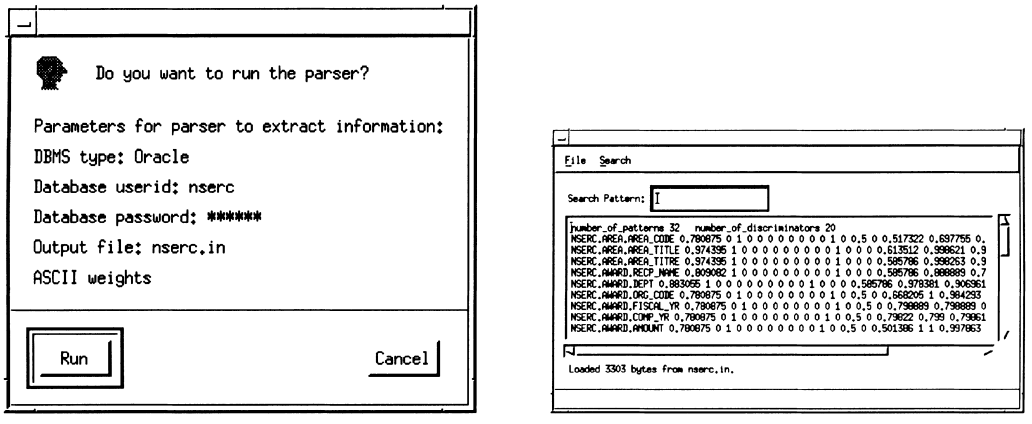

Fig. 7. (a) Parser window in SEMINT and (b) Parser output result.

example, in Oracle7 the user id, user password, and table owner id are needed. SEMINT uses the user's login information as default unless the user specifies otherwise. The Oracle7 parser window in SEMINT and its output are shown in Figs. 7(a). DBMS specific parsers are implemented using C with embedded SQL (e.g., Pro*C in Oracle7).

3.3.2. Data type conversion for comparisons of metadata from heterogeneous databases

The data types used in Oracle7 are Char, Long, Varchar, Number, Date, Raw, Long raw, MLSlabel (binary format of an operating system label) and Rowid. Different DBMSs may use different sets of data types or even have their own data types that cannot be found in other DBMSs. An attribute **age** may be specified as Integer data type in one database and as a Number data type in other database since there is no Integer data type available, such as in Oracle7.

We handle this problem by defining five distinct types (character, number, date, rowid, and raw). For each DBMS, we map that DBMSs data types to these five types (e.g., Oracle's char, long, and varchar all map to 'character'.) Note that the 'length' discriminator will allow us to distinguish between different uses of these data types. We present data type metadata of all DBMSs as Number, Date, Char, Raw, RowID, Precision, Scale and Length as shown in Fig. 8. As this form can capture all the data types used in various DBMSs, we are able to compare attributes of various DBMSs that may use different sets of data types. How to determine the similarity between these data types is 'learned' directly from the metadata presented to neural networks as training data.

3.3.3. Normalization of metadata

As the input values for our neural networks need to be in the range of 0–1 (whether a neuron is triggered or not), the metadata of attributes, which can be any values (e.g., an attribute of character data type with length of 20), need to be normalized into values of range [0,1]. The problems such as "how can we present a data type as a value of range [0,1]?" and "how can we convert length into a value of [0,1]?" need to be solved. Three types of normalization methods are used to convert the metadata into a vector of values between 0 and 1.

Binary values: We map true/false information to binary values (e.g., 1 for key field and 0 for not a key field). If a null value is allowed in this field, a null value is presented as 0.5, that is, it could be either 0 or 1.

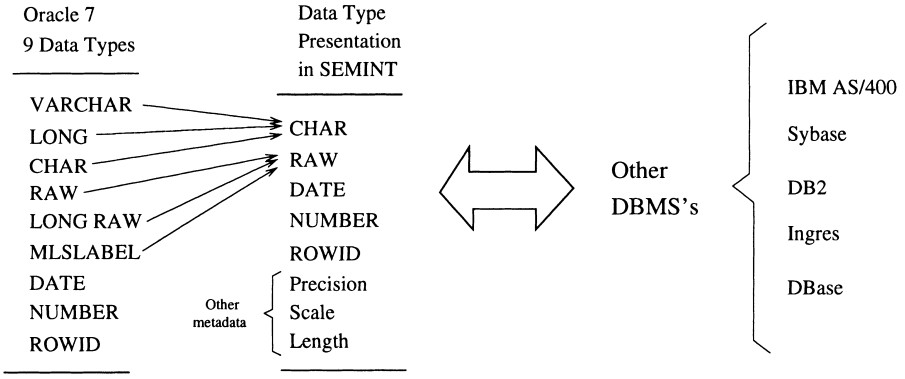

Fig. 8. Data type presentation in SEMINT.

Category values: Category information such as data types require 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 neural network determine if this is true. Instead we convert this category input to a vector of binary values (e.g., 1,0,0 for character data type, 0,1,0 for numeric data type, and 0,0,1 for date data type).

Range values: We map others to a range [0,1] using a *SIGMOID*-like function as shown in Fig. 9. We use the function $f(x) = 1/(1 + k^{-x})$ for normalizing numeric ranges, with $k = 1.01$. Varying k changes the steepness of the function curve. For example, choice of using $k = 1.1$ provides reasonable discrimination for the range $[-50, 50]$. For some metadata, such as field length, that have only positive values, we use the function $f(\text{length}) = 2 * (1/(1 + k^{-\text{length}}) - 0.5)$ with $k = 1.01$. This normalization gives us discrimination for the range [0,100].

The reasons for not using a linear normalization function, such as value/maximum value, are as follows:

- *Avoid false match*: an attribute with field length of 10 in a database where the maximum field length is 100 and an attribute with field length of 100 in a database where the maximum field length is 1000 become the same after the linear normalization ($10/100 = 100/1000$); however, they are different: this is a false match.
- *Avoid false drop*: an attribute with field length of 10 in a database where the maximum field length is 100 and an attribute with field length of 10 in a database where the maximum field length is 1000 become the different after the linear normalization ($10/100 \neq 10/1000$); however, as they should be the same this is a false drop.

The information extracted from different databases is then transformed into a single format and normalized. The parser output is a set of vectors as shown in Fig. 7b; each vector presents the signatures of an attribute, in term of its schema design and data content patterns. Fig. 10 shows an example of the metadata representation of an attribute *book_title*. Note this captures that the data type is character and it is not a key. If a database contains 10 attributes and we extract 20 discriminators to describe the signatures of these attributes, the parser output has 10 vectors and each vector has 20 values in the range of [0..1].

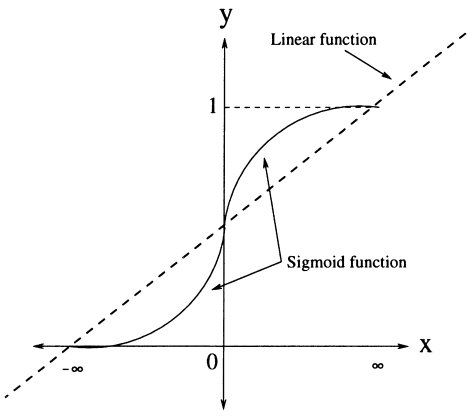

Fig. 9. *SIGMOID* function versus linear function.

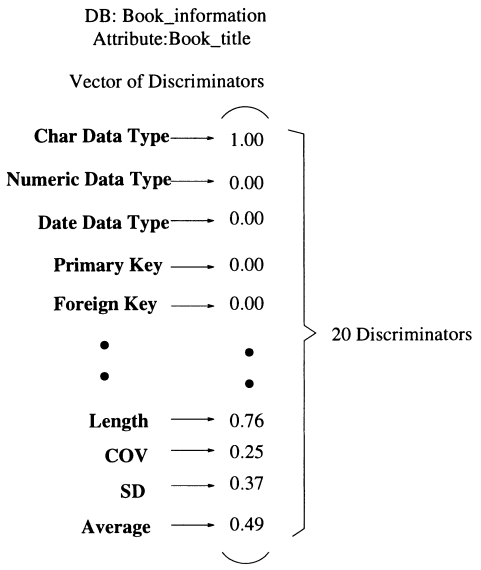

Fig. 10. Discriminator vector of an attribute.

3.4. Classifier

Before the metadata described in Section 3.3.3 is used for neural network training, we use a classifier to cluster attributes into categories in a single database. The are several reasons for using the classifier as the first step of this semantic integration system:

1. If we have information in one database that refers to the same real-world information, we do not want to separate them into two different categories.
2. Ease of training. Classification lowers the number of nodes in the back propagation network output layer. It reduces the problem size and therefore the training time.
3. 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; this is less computationally expensive.
4. Networks cannot be trained with a training set where there are two identical answers to a question and one is correct while another is not. The classifier detects cases where this is true, and groups them to one cluster. Otherwise, the network cannot be trained to point out they are different.

SEMINT uses the Self-Organizing Map algorithm [10], an *unsupervised* learning algorithm, as a classifier to categorize the attributes within a single database. The original SOM algorithm allows users to specify the number of clusters to be created. However, we do not know in advance how many clusters to expect. Therefore, we have adapted this algorithm so that the users or the DBA (database administrator) can determine how fine the categories are by setting the radius of clusters rather than the number of categories.

Assume that we use N discriminators to characterize an attribute. Each attribute's characteristic vector represents a point in the N-dimensional space. The self-organizing map algorithm clusters (classifies) these points in the N-dimensional space according to the Euclidean distance

between them and the maximum allowed radius of a cluster, given by the user. Given our normalization procedure, the maximum distance between any two points in the N-dimensional space is $\sqrt{N}$. In our experiments with SEMINT we have found that a maximum radius value of $\sqrt{N}/10$ is a good default value that can be adjusted by the user if so desired. The output of the classifier is the weights of these cluster centers (locations in this N-dimensional space).

We now use an example to illustrate this. Assume there are six attributes in a database. Table 2 shows there are two tables in the database and there are four and two attributes in those two tables, respectively. Fig. 11 shows how these six attributes are located in a three-dimensional space corresponding to the discriminators: key field, length and data type. Note that this represents a simplification of the actual representation in SEMINT so that we can draw points in a three-dimensional space. We may specify a maximum radius value of $\sqrt{3}/10$ for clustering these six attributes. Hence, the data type specification is lumped into one characteristic value (instead of two): 1 represents a numeric data type and 0 represents a character data type. The classifier clusters these attributes into four clusters as shown in Fig. 11. The output of the classifier are the vectors of cluster center weights that are computed by taking the average over the attribute

Table 2
Example of attribute schema

Attribute name	*Key field?*	*Length*	*Data type*	*Representation*
Personnel table:				
SSN	Yes	9	Numeric	(1, 0.47, 0)
Name	No	11	Character	(0, 0.6, 1)
Address	No	25	Character	(0, 0.7, 1)
Tel#	No	10	Numeric	(0, 0.51, 0)
Employee table:				
Emp_ID	Yes	9	Numeric	(1, 0.47, 0)
Emp_name	No	12	Character	(0, 0.62, 1)

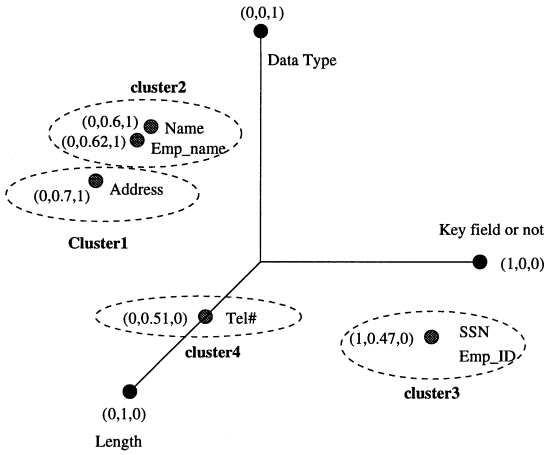

Fig. 11. Illustration of a 3-D self-organizing map.

vectors in the corresponding clusters. In this example, the cluster centers for *clusters* 1, 2, 3 and 4 are (0,0.7,1), (0,0.61,1), (1,0.47,0), (0,0.51,0), respectively.

Fig. 12 shows a self-organizing map neural network architecture that contains two layers and whose nodes are fully connected. The number of input nodes, N, is the number of discriminators used to characterize an attribute and M is the number of clusters generated. We observe again that M is determined by the neural network rather than being specified by the user as in the original SOM algorithm [10]. The detail of the adapted self-organizing map neural network algorithm is given in Appendix A.

The output of the classifier is the vectors of cluster center weights and each vector still has the same number of discriminators, N. The classifier window in SEMINT and its output are shown in Fig. 13.

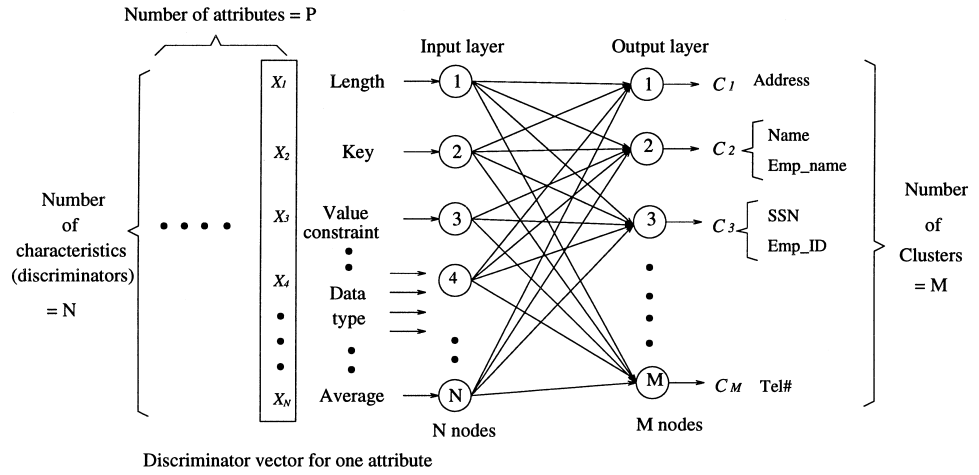

Fig. 12. Classifier architecture in SEMINT.

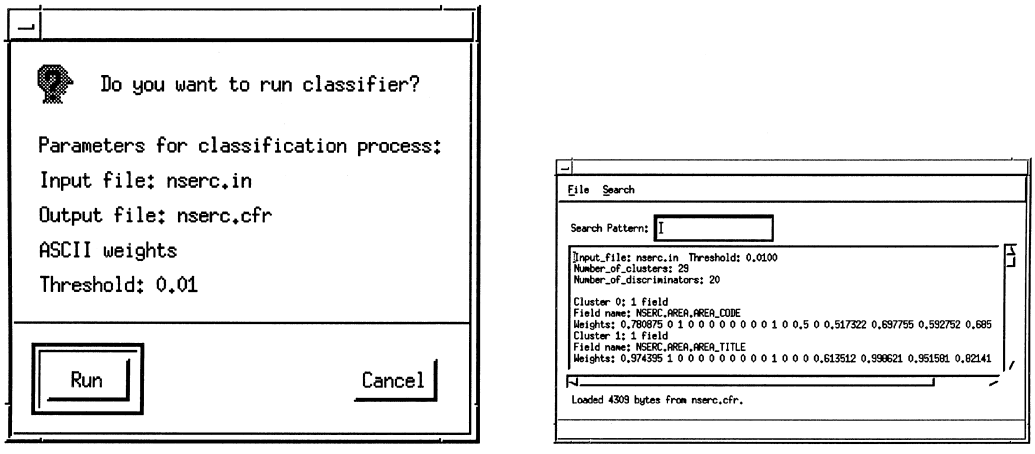

Fig. 13. (a) Classifier window in SEMINT and (b) Classifier output result.

3.5. Category learning and recognition neural networks

The output of the classifier (M vectors) is then used as training data for a back-propagation network. The back-propagation learning algorithm [27,28] is a *supervised* learning algorithm, where 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 database attributes' signatures. The training data here is the cluster centers output from the classifier, that represent the signatures and patterns of attributes in a particular database. When we provide the trained neural network signatures of attributes (from another database) it tells us if there are matches – corresponding attributes in the other database. Although the need for training data normally implies human effort, our training data is generated automatically by the classifier. The process is fully automated and no *human*-generated training data is needed.

The output of the classification process for our running example from Fig. 11 (where $N = 3$ and $M = 4$) are the clusters center weights, as discussed before. These weight vectors are tagged with target results (cluster numbers) and they constitute the training data for the back-propagation neural network. The training data are as follows:

0	0.70		1	→	Cluster center 1 (Address)
1	0	0	0	→	Target result
0	0.61		1	→	Cluster center 2 (Name, Emp_name)
0	1	0	0	→	Target result
1	0.47		0	→	Cluster center 3 (SSN, Emp_ID)
0	0	1	0	→	Target result
0	0.51		0	→	Cluster center 4 (Tel#)
0	0	0	1	→	Target result

Fig. 14 shows a three-layer neural network for recognizing M categories of patterns. There are N nodes in the input layer on the left side, each of which represents a discriminator. The hidden

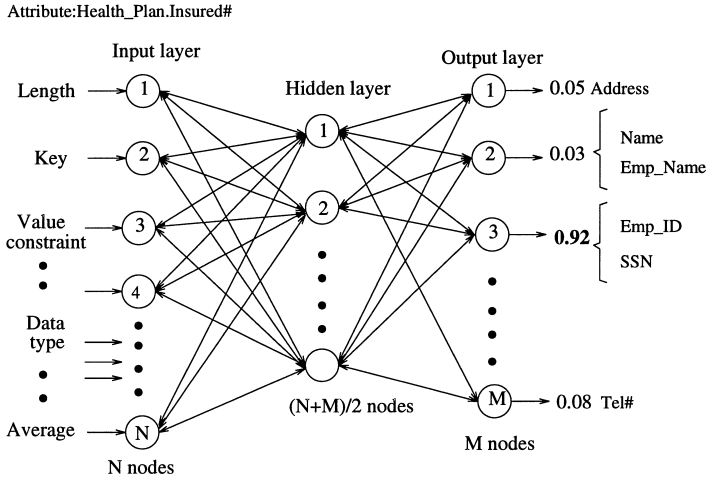

Fig. 14. Back-propagation neural network architecture in SEMINT.

layer consists of $(N+M)/2$ nodes in the middle that are connected to all input and output nodes. The output layer (on the right side) is composed of M nodes (M categories). Thus the number of connections (edges) in the network is $(N+M)^2/2$.

Initially, the connections in the network are assigned arbitrary values in the range $[-0.5, 0.5]$. During training the network changes the weights of the connections between the nodes so that each node in the output layer generates its target result (corresponding category number). The forward propagation (generating the 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 error threshold given by the user.

For example, for the training data just shown, when presented with the input '0 0.70 1' (weights of cluster center of category 1), the network should output the target result '1 0 0 0', the target result that indicates category 1. Assume that for the current iteration of the back-propagation algorithm the output generated for this input is '0.95 0.05 0 0.1'. If the error threshold is 0.005, i.e., the maximum difference between any two corresponding components of the target result and the generated result cannot exceed 0.005, the iteration continues until the required convergence in results. After training, the network can be used in order to encode the metadata of another database by matching each input attribute pattern to the closest output node and giving the similarity between them.

As an example, take the result of the classifier in Fig. 12 that clustered 'Emp_ID' and 'SSN' into one cluster. The weights of these cluster centers are then tagged to train the network in Fig. 14. After the back-propagation network is trained, we present it with a new pattern of N discriminators standing for attribute *health_Plan.Insured#*. This network determines the similarity between the given input pattern and each of the M categories. In Fig. 14, the network shows that the input pattern 'Insured#' is closest to the category 3 (*SSN* and *Emp_ID*) with similarity 0.92. It also shows *health_Plan.Insured#* is not likely related to other attributes since the similarity is low.

We have implemented two versions of back-propagation algorithms, standard [27,28] and quick-propagation [6]. For details of the training algorithms, see Appendix B.

3.6. Using neural networks

To determine attribute correspondences between two databases, users take the network trained for one database, and use information extracted from the other database as input to this network

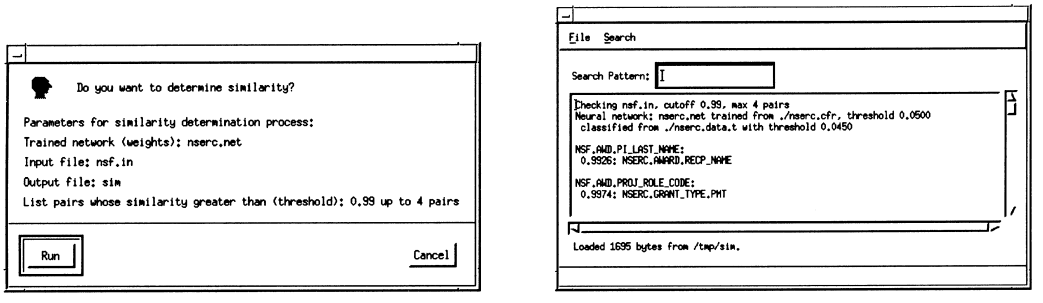

Fig. 15. (a) Similarity determination window in SEMINT and (b) Output results.

as shown in Fig. 15a. The network then gives the similarity between each pair of attributes in the two databases. Users can also specify filtering functions so that SEMINT only presents those pairs with very high similarity (e.g., only recommend the top 10 pairs of similar attributes or those similarity of at least 0.80). System users check and confirm the output results of the trained network. The output results, shown in Fig. 15b, include lists of corresponding attributes and their similarity.

3.7. Automated attribute correspondence identification procedure (summary)

Fig. 16 shows a 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).

Step 1: Use DBMS specific parsers 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. 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 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 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. During the training, the network changes its weights so that each node in the output layer represents a cluster. The threshold and the learning rate can be set by the system users. After training, the network encodes data by matching each input pattern to the closest output node and giving the probability values of similarity between input pattern and each category. Note that the Steps 1–3 must be performed only once per database.

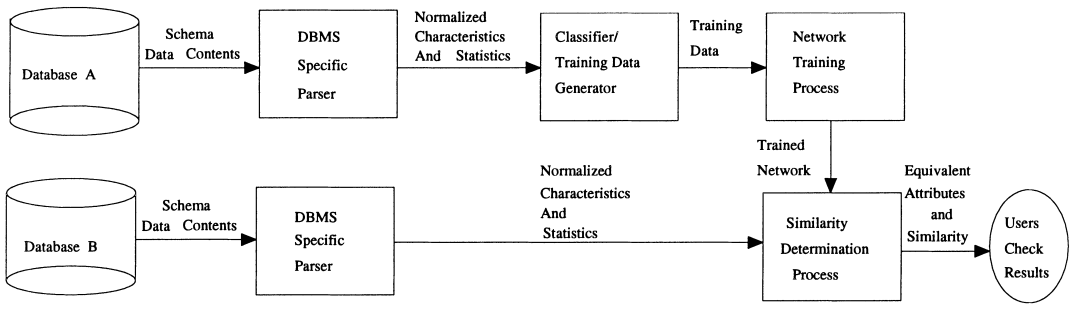

Fig. 16. Attribute correspondence identification procedure.

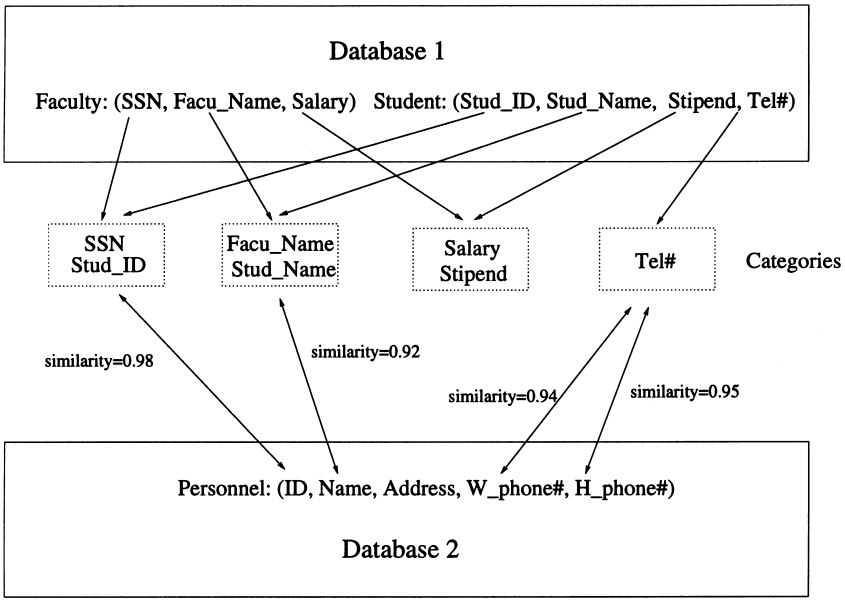

Fig. 17. Example of attribute correspondence determination process.

(Database1.Faculty.SSN, Database1.Student.Stud_ID, Database2.Personnel.ID, similarity = 0.98)
(Database1.Faculty.Facu_Name, Database1.Student.Stud_Name, Database2.Personnel.Name, similarity = 0.92)
(Database1.Student.Tel#, Database2.Personnel.W_phone#, similarity = 0.94)
(Database1.Student.Tel#, Database2.Personnel.H_phone#, similarity = 0.95)

Fig. 18. SEMINT output for example in Fig. 17.

Step 4: The input for the network trained in Step 3 is the attribute information of another database (database B). [4] 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.

In Fig. 17, the attributes in database 1 are clustered into four categories using the classifier and the cluster center weights are then used to train neural network. After the neural network is trained (it knows the signatures of these four categories), it matches the attributes in database 2 with these categories classified from database 1. The output result of this process is shown in Fig. 18.

[4] Clustering attributes of database B is optional. This may add work to a single integration, but the clustering information can be reused to save both computation time and human effort in future integrations.

4. Experiments on real databases

Given the 'fuzzy' nature of both this problem and the solution presented, it is important to verify this work experimentally. In this section, we present experimental results using SEMINT to solve real database integration problem in identifying attribute correspondences. Two measures of success/failure are examined. They are defined as follows:

$$\text{Recall}: \frac{\text{Corresponding attributes correctly recommended}}{\text{Total actual corresponding attributes in the databases}},$$

$$\text{Precision}: \frac{\text{Corresponding attributes correctly recommended}}{\text{Total corresponding attributes recommended}}.$$

For example, assume there are 50 pairs of corresponding attributes in two databases, and SEMINT recommends 60 pairs of corresponding attributes. After examining the results, the user discovers that 45 of the 60 pairs of recommended attribute correspondences are correct. The *Recall* is then 90% (45/50) and the *Precision* is 75% (45/60). *Recall* tells us the effectiveness of SEMINT; that is, if we can count on SEMINT to find corresponding attributes and save us from searching on our own (at a cost of an average 4 h to find one pair of corresponding attributes in US West's experience). *Precision* gives a measure of efficiency of SEMINT; that is, how reliable the recommendations are.

It is not easy to find appropriate databases for experiments. The criteria of databases suitable for our experiments is that databases need to be realistic enough to manifest the qualities that make this problem challenging and well-understood (the corresponding attributes are known), so as to allow us to compute the preceding measures.

Small or 'toy' data sets do not meet the requirement that they be realistic. Finding large databases that are available for research use is difficult as well, finding pairs of databases that are well-understood and thus can serve as a baseline for comparison is exceedingly difficult. Therefore some of our experiments using tables containing related data from a single database. (This gives well-understood and realistic data, but is dubious with respect to the *problem* being realistic.)

Our experimental results include using SEMINT on real databases of various DBMSs including IBM AS/400, Sybase and Oracle. These databases range from small to huge database systems in large organizations; and databases from different countries using different languages. The sizes of these databases range from a few megabytes to terabytes. We categorized the experimental environments as follows:

- Similar databases from a single organization. This is done both through simulation with tables from a single database, and by comparing 'old' and 'new' versions of a database (this tends to yield solid recall and precision numbers; and shows the applicability of SEMINT in legacy databases). The databases for our experiments are from Reuters Information Systems and others.
- A sample of a very large database in a single large organization, by separating the database into two parts, and searching for similarities between those parts (this tests the ability of SEMINT to scale to large problems). The databases for our experiments are from Boeing Computer Services.
- Databases containing similar information from different organizations, where a manual detection of attribute equivalences has been performed (a realistic test, however we have been unable

to find enough databases to perform a substantial number of these tests). The databases for our experiments are from NSF in USA and NSERC in Canada.

We first present some preliminary results using limited metadata manually extracted from small databases. In many collaborations, we were not permitted to run our parser programs on the computers of participating organizations. Rather, we were only provided hard copies of database schema. We then present results from larger databases, based on the complete SEMINT implementation and automated collection of metadata.

4.1. Preliminary experimental results

Our initial experiments with SEMINT were on similar databases from the same organization containing a good deal of common information. [5] We extracted metadata information from these databases manually; not all of the information described in Table 1 was available. One pair was transaction data for British pound call options and put options from 2 March 1983 to 27 June 27 1985. The only information available for these databases was a sample of the data. However, we derived the schema based on this sample. The second pair was IBM AS/400 databases provided by Marketing Information systems in Evanston, Illinois; we used the field reference files to obtain schema information (the data contents were not available.) The third pair was Sybase databases provided by Reuters Information Systems in Oak Brook, Illinois, for which we also had only schema information.

The result of these experiments is shown in Fig. 19. These indicated that our method had promise: The metadata values clearly corresponded to the semantics of the attributes, and the neural network training process was able to determine what this correspondence was for each pair of databases. We can see that there is a substantial gap between high and low similarity as determined by SEMINT, and with one exception pairs with high similarity reflected corresponding attributes, and low similarity reflected non-corresponding attributes.

The one false match in these experiments was with the British Pound Option Transaction Databases (upper portion of Fig. 19). SEMINT was unable to distinguish between spot ask and spot bid price. These attributes were identical in our generated schema information, and as market forces cause spot ask and spot bid prices to move in 'lock step', the data content statistics are quite similar as well. However, the *recall* in these experiments is excellent (100%), and the *precision* is very high (90%, 100% and 100%). For more details of these experiments, please see [14]. [6]

These preliminary results convinced us that this technique had promise, so we developed a full-scale implementation allowing us to automatically extract metadata from databases. This allowed us to perform larger experiments. We now present two large-scale tests: One in collaboration with Boeing Computer Services (to test how SEMINT works on very large databases), and another on funding award databases provided by the US National Science Foundation (NSF) and the Canadian National Science and Engineering Research Council (NSERC) (a test of inter-organiza-

[5] These databases were also used for our experiments in [13].

[6] We also performed some tests of cross-DBMS integration, however given the data we had available these were not very interesting. For example, we tried integrating one of the IBM AS/400 databases with one of the Sybase database. These contain no common information. The average similarity was 0.006 and the maximal similarity was 0.021. The result shows that attributes of these two databases are totally different (as expected).

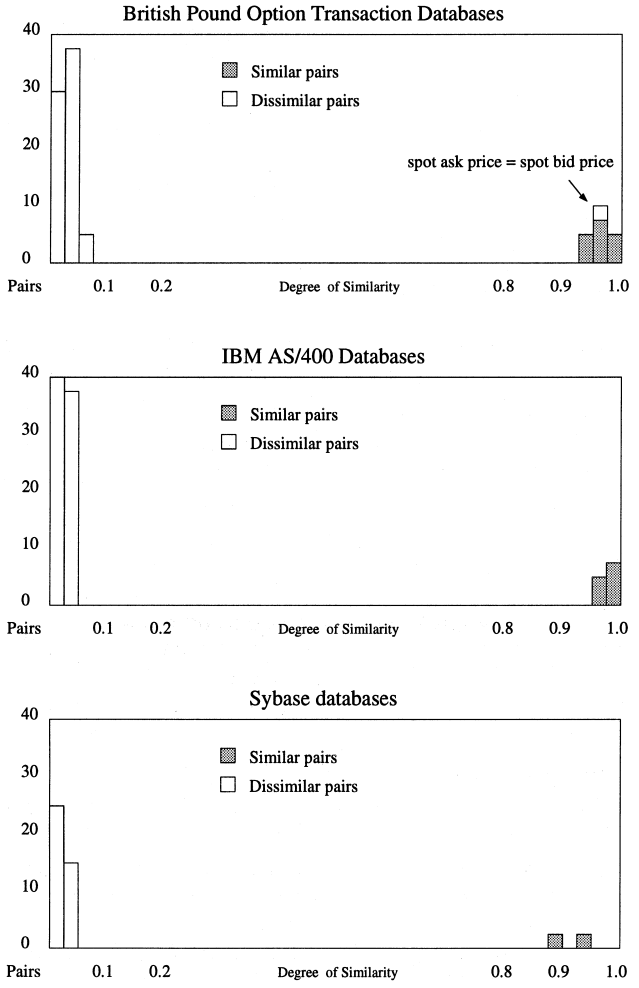

Fig. 19. Preliminary experimental results on various DBMSs.

tion database integration). We will first discuss the collaboration with Boeing, then the results on the funding award databases.

4.2. The Boeing company tooling database

Boeing Computer Services is interested in general problems of knowledge discovery in databases, and sees SEMINT as a possible component of this. A first step is to verify the effectiveness of SEMINT. Their environment is ideal for this, as they have a large number of well designed and well understood production databases. This enables us to test if SEMINT can 'discover' *already known* relationships in the data.

4.2.1. Experiment description

We tested SEMINT on the tooling database at Boeing Computer Services in Bellevue, Washington. The tooling database at Boeing runs under the Oracle7 DBMS. The database

contains 12 tables and 412 attributes; some attributes have millions of rows. The biggest table has 147 attributes and the smallest table has 5 attributes. Its complexity and size (in terms of the numbers of rows and attributes) provide a good test bed for SEMINT.

We first used our Oracle7 parser (Section 3.3) to extract metadata from the database (schema specification information and statistics on data contents). All of the 20 discriminators shown in Table 1, are used in this experiment.

We split this information into two parts, to represent two databases containing related information. The first part (denoted AM) had 4 tables with 260 attributes; the second (OZ) had eight tables with 152 attributes. As the database as a whole was well understood, we were able to categorize the relationship between two attributes into three categories: Equivalent (e.g., SSN and ID), similar (e.g., student name and employee name), and different (including completely unrelated and slightly related attributes). We then checked to see how well SEMINT could find corresponding attributes (either similar or equivalent): Those that are 'candidates' for integration, applying integrity constraints, or further study on their behaviors (data mining, etc.). To do this, we trained a neural network to recognize the attributes of AM, and then determined the similarity between this and OZ.

4.2.2. Experimental results

We ran the experiment *without* assistance from DBAs, database owners, or anyone else with knowledge of the database, as with some knowledge of the database, we could interact with SEMINT and change various parameters to improve the results.

The precision in this test was approximately 80%. And the recall was close to 90%. Please note that we cannot give specific *recall* figures since we were only provided with the tooling database for the experiment. We can only judge based on the tooling database. However, we cannot determine how many attributes in the database other than the tooling database that are actually associated with. As given the size and complexity of the tooling database and the whole information system supported in BCS, it is extremely difficult to say exactly how many 'corresponding' attributes exist since this is based on applications and the level of integration efforts.

Determining *Precision* is much easier, as only the recommendations need to be checked (and not all possible pairs). We can use foreign key relationships to find completely equivalent attributes, but this would give an unfairly high recall number as there are many corresponding attributes that we want SEMINT to find, but that are not reflected in foreign key relationships. We were able to find most of these completely equivalent attributes.

One interesting question is how much benefit was gained from using a neural network to determine the similarity (as opposed to a preprogrammed 'similarity function'). To test this, we computed the Euclidean distance between attributes; carefully tuning the distance where we considered attributes to be similar gave us a precision close to 90%. This would appear better than the networks at 80%, however classification based on Euclidean distance only gave 59 'groups' of corresponding attributes, the network gave us 134. This is shown in Fig. 20. Increasing the Euclidean distance where we consider attributes to be similar caused a huge drop in precision. The neural network was much less sensitive to such changes; the results using 0.8 and 0.9 as a threshold for considering attributes similar were nearly identical (only six additional groups, some of which did contain similar information).

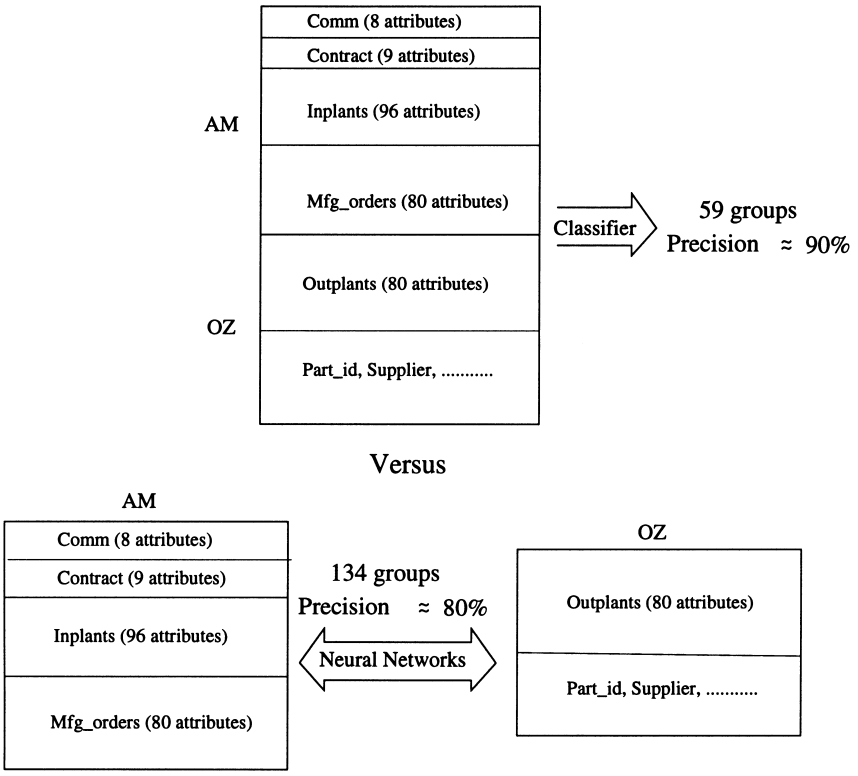

Fig. 20. Experimental results on boeing tooling database.

Without assistance of a tool like SEMINT, to find equivalent and similar attributes (without knowledge of the databases) is tedious. For example, in order to integrate AM and OZ, we need to go over 39,520 pairs of attributes (260 × 152). This is perhaps more easily visualized as 659 pages of printout (at 60 lines per page). With SEMINT, users can get many of the relationships with comparatively little effort (two pages of recommendations).

Another interesting question is the computational requirements of SEMINT. Table 3 shows the training time in our experiments. The time required to parse metadata from Boeing databases is less than 10 min wall clock time with applying sampling techniques. However, the time needed to parse metadata from a database varies. It is a function of the number of attributes, the number of

Table 3
System performance and human effort needed for BCS database experiments

Process	CPU time	Human effort
Extract information from databases	Varies	Little (once per DBMS)
None (handled by parser)		
Classify attributes into categories	< 1 second	Little (give threshold, once per database)
Train networks for AM	1983 seconds	Little (give threshold, once per database)
Train networks for OZ	369 seconds	Little (give threshold, once per database)
Determine similarity	< 0.1 second	Some (check results)

rows in a table (assuming no data sampling technique is applied), the machine and DBMS used, and the location of the database (e.g., remote or local database), and the load on the database.

The rest of the experiment was run on an IBM RS/6000 model 320H, running AIX 3.2.5. The CPU time needed for the classification or recognition processes was under 1 s. The training time in our experiments was reasonable when we set the threshold to 0.1. It took less than 10,900 epochs (cycles of forward-propagation, backward-propagation, and weight-update) to train the networks; this is very efficient for a network of this complexity. The training time can be reduced using larger thresholds, and better training algorithms are also an area for further research.

4.3. US NSF and Canada NSERC funding award databases

The proceeding experiments show that this method can be useful. However, we are primarily interested in integrating much more diverse collections of information than we have discussed so far. Gaining access to databases in diverse organizations is difficult (although with improved ability to make use of such access from *I*3 research, we expect more databases will become available). We were able to obtain one pair of databases that reflect a likely real-world case where intelligent integration of information would be of interest.

The NSF and NSERC award databases are both stored in relational databases. They contain information on research grants, including the project investigators, title, award amount, institution, program manager, division granting the award, etc. The NSF database contains 47 separate attributes (in one table), the NSERC database contains 32 (in six tables). The NSERC data represents 21 distinct types of information (there are five keys duplicated in award and each of the other tables, and six titles given in both English and French). They contain a wide variety of information, with a few attributes reflecting the same information.

An additional challenge posed by these databases is the lack of common entities. There are no common grants, institutions, program managers, or investigators between the two databases. Such 'common information' could be expected to ease the integration problem.

Although unable to access these databases directly, we were able to obtain the data and most of the schema information and transfer these to an Oracle7 database. We copied the Schema information as directly as possible, although some information (such as constraints on the data) was unavailable. We did not *add* any information; we feel the results would only be better if we actually had access to the databases.

To give a baseline for the research, we performed a manual comparison of the databases. This required considerable 'domain expertise', such as knowledge of individual grants, program managers, and the structure of the granting organizations. Table 4 shows the common attributes between the databases. Although certain relationships were obvious, it took several days to ascertain all the information in the table (particularly the meaning of some of the code values).

Table 5 summarizes the results of SEMINT trained on the NSERC database and used to find corresponding attributes in the NSF database. The NSERC database was classified at 0.01 (this separates French and English titles). The threshold for similarity was 0.99. At this similarity threshold, we achieve a recall of 38% (6/16). The precision is 20% (6/30).

Note that we do well with project title, recipient name, and award amount. We fail on institution name – NSERC abbreviates these differently from NSF, and the NSERC database has many other character fields that appear closer to the institution name than *org_name*. The dates

Table 4
Actual attribute correspondences between NSF and NSERC databases

NSF	*NSERC*	*Description*
awd_titl_txt	project	Project Title
pi_last_name, copi(1-4)	recp_name	Recipient
inst_shrt_name	org_name	School/organization name
est_tot_awd_dol,to_date_dol	amount	Dollar amount
awd_eff_date	comp_yr	Starting year
inst_id	org_code	School/organization code
org_code	ctee_code,area_code	Program giving grant
pgm_ele_code	area_code,disc_code	Program discipline
awd_istr_abbr	grant_code	Type of grant

Table 5
SEMINT results comparing NSF and NSERC databases (correct matches in **boldface**)

NSF	*NSERC*	*Similarity*
awd_titl_txt	**project**	**1.0000**
pi_last_name	**recp_name**	**0.9999**
copi_last_name_1	**recp_name**	**0.9949**
copi_last_name_2	**recp_name**	**0.9970**
est_tot_awd_dol	**amount**	**0.9983**
to_date_dol	**amount**	**0.9944**
equp_dol	amount	0.9955
awd_istr_abbr	province	0.9942
org_code	grant_code	1.0000
pgm_ele_code	grant_code	0.9998
inst_id	grant_code	0.9976
and 8 other matches to grant_code		
sen_acad_mnth_cnt	ctee_code	0.9990
and 6 other matches to ctee_code		
proj_role_abbr	pmt	0.9990
and 3 other matches to pmt		

are difficult, as the NSF database stores them as a DATETIME, and the NSERC database stores them as an integer year. This would be difficult to handle without some domain knowledge. One option would be to normalize so that statistics on a date and an integer year correspond (this helps), but this makes all dates appear very similar, and cuts our ability to distinguish between, for example, expiration and start dates.

The codes are difficult – they reflect similar information, but any meaningful mapping would require substantial domain knowledge (beyond ANY information available in the database). For example, the NSF database stores all codes as alphanumeric strings, and the NSERC database stores them as integers (with the exception of type of grant).

Table 6 shows the time required for the various stages of SEMINT (the database was on an IBM RS/6000 model 220, the remaining tasks were on an RS/6000 model 320H). Note that the 'slow' steps (extracting information and training the network) only need to be done once per

Table 6
System performance (wall-clock time in seconds): Award databases

Process	*NSERC*	*NSF*
Extract information from databases	83	184
Classify attributes into categories	< 0.1 s	
Train networks (classified at .05)	156	247
Train networks (classified .01)	724	5290
Determine similarity	< 0.1 s	

database. The similarity determination is fast enough to be done interactively (for each new database we want to integrate with) and this allows dynamic integration.

5. Experiences with SEMINT

In this section, we discuss our experiences using SEMINT to the problem of database integration. Then we highlight some other applications where we have applied SEMINT.

5.1. Implementation of neural networks

One common major concern of using neural networks is training time. The foundation of the neural networks paradigm was laid in the 1950s. Recently, with development of more powerful hardware and neural network algorithms, neural networks have emerged as a powerful pattern recognition technique. Selecting proper algorithms and implementing and tuning them for the tasks are essential to their effectiveness and efficiency.

We selected the back-propagation algorithm for its fast convergence speed and more noise-resistant, over other machine learning techniques such as symbolic learning and genetic algorithms. For advantages of back-propagation algorithm and comparison with other algorithms, see [2].

We have implemented two back-propagation algorithms: the standard back-propagation algorithm [27,28] and the quick back-propagation algorithm [6], that employs a more 'aggressive' and 'adaptable' learning rate method to speed up the training process. The quick back-propagation algorithm does perform substantially better than the standard algorithm in most cases; further improvement in training algorithms is possible. Our learning process also implements batch learning to speed up the learning process as the empirical study described in [7].

We also employ a classifier to cluster attributes before presenting them as a training set for the pattern recognition algorithms. This step further reduces the training time substantially. Classifying at a higher threshold gives a shorter training time, as it is easier to train a net where the distance between the input data items is larger. When we did not use the classifier at all (except to eliminate items with identical descriptions), training took much longer.

5.2. Experiences with neural networks

We find that the training time is reasonable as long as we provide less than 50 categories of attributes as training data. We also found that neural networks generally work well in solving this 'fuzzy' problem. Our experiments on the tooling database at Boeing Computer Service shows that

classifier (based on Euclidean distances) is very effective in identifying *identical* or *almost identical* attributes as it treats each metadata equally. On the other hand, pattern recognition neural networks are more effective in detecting *similar* attributes in additional to *identical* or *almost identical* attributes.

As we are working on heterogeneous database integration, ability to identify *identical* and *almost identical* attributes is not sufficient because we are dealing with fuzzy attribute relationships. Many corresponding attributes are really 'similar' in schema design or data patterns, such as char(14) versus char(15). Recall is as important as *precision*, and some time it is even more important since manual search is almost impossible. Training process gives the neural networks the ability to identify *identical*, *almost identical* as well as *similar* attributes by exploring their signatures in schema and data contents at some cost to *precision*. This is understandable because we are trying to identify corresponding attributes in different databases, DBMSs, organization, even different countries and languages (NSF and NSERC databases) and usually designed by different DBAs at different times.

5.3. Experiences with database integration

We have testing SEMINT in different environments from databases within a single small organization to databases contributed from two different countries. We argue that the focus and difficulties are different in each environment:

Enterprise-wide database integration: SEMINT has been very effective in providing attribute correspondence identification functionality. We see from our experiments on transaction data of British pound option, and databases provided by M.I.S. Systems and Reuters Information Systems that schema designs and data patterns are quite similar. The metadata we used is effective signature of attributes.

Very large database integration: The databases in larger organizations are more complicated than those in smaller organizations. The difficulties include databases designed for different purposes at different times. Databases may migrate from some older database systems. Data may be accumulated over a very long period so that we need to deal with inconsistency and missing data. Therefore, the focus is to collect metadata and transform it into a single format. Data sampling is also very important as we need a huge amount of data and attributes. Moreover, these data and attributes may be inconsistent. Our approach is more fault tolerant and less time consuming compared to other approaches.

How to present the results (recommended corresponding attributes) is also an issue since a poor presentation of results cannot visualize effectively and efficiently such as presenting users hundreds of pages of printout. From our discussion with Boeing Computer Service, providing an interactive working environment for attributed correspondence identification is helpful. Although SEMINT is automated, the results (a ranking of likely correspondences) can be adjusted to allow the user to specify the number and minimum similarity of potential correspondences to examine.

Inter-enterprise integration (NSF and NSERC Databases): This is the most difficult information integration situation we face. Here the attribute correspondence identification problem is compounded by other types of heterogeneity [42]. These include representation, scale, and level of abstraction as well as language problem (English versus French) and lack of complete metadata. In our experiment, we use only metadata that can be extracted from DBMSs. However, further

work, such as integrating with domain knowledge bases, is needed to determine specifically what post-processing works in general.

5.4. Scalability of SEMINT

The only human input necessary to run SEMINT is to specify the DBMS types and database connection information and to examine and confirm the output results. The other processes can be fully automated. Users are shown corresponding attributes with similarity greater than a desired threshold. The users can further specify the maximum number of similar attributes to be retrieved (e.g., top 10 pairs with similarity >0.8). Note that the training of a neural network needs to be done only once; the actual use of the network to determine attribute correspondences is very efficient and can be done almost instantly. In addition, users can provide additional domain knowledge to eliminate some unnecessary comparisons.

We can further reduce the neural network training time by building several smaller networks rather than a large network. For example, assume that we want to train neural networks to recognize 1000 distinct attributes with 20 discriminators. Instead of one three layer back-propagation network we could build 20 small neural networks, each of them capable of recognizing 50 distinct attributes. The advantage of this technique is that the total number of connections in the neural network can be substantially reduced from $20 * (20 + 1000)/2 * 1000$ (10.2 million) to $20 * (20 + 50)/2 * 50 * 20$ (0.7 million).

5.5. Other applications of SEMINT

SEMINT aims at assisting DBAs in finding corresponding attributes among heterogeneous databases in large organizations. It has been tested using various databases to solve real world problems. SEMINT has been applied to identify attribute correspondences in large organizations, such as Boeing Computer Services (BCS) as described in Section 4 in their efforts in database integration.

Although SEMINT primarily serves as a tool for identifying attribute correspondences in heterogeneous databases, we have also applied it to software reuse problems.

There are three steps in constructing software from reusable components: Conceptualization (modeling process), retrieval and selection (retrieval process), and reuse (reuse engineering process). Conceptualization determines what components are needed according to the user requirements. Retrieval and selection allocate candidate components and evaluate them for reuse. The reuse engineering process involves validating useful components and making necessary changes to incorporate them into the system.

Reuse is a hunter/gatherer intellectual activity. We see that the retrieval phase in software reuse and matching phase in heterogeneous database integration are two sides of the same problem: Determine the similarity between two existing schemas. The two schemas can be either two component databases, a conceptual schema and a candidate component schema, or a component schema and a global schema. In the software reuse process, designers of software systems have a conceptual schema in mind, then search for reusable component schemas (CS) that match the specification of the conceptual schema. This is a 'top-down' process as shown on the right in Fig. 21.

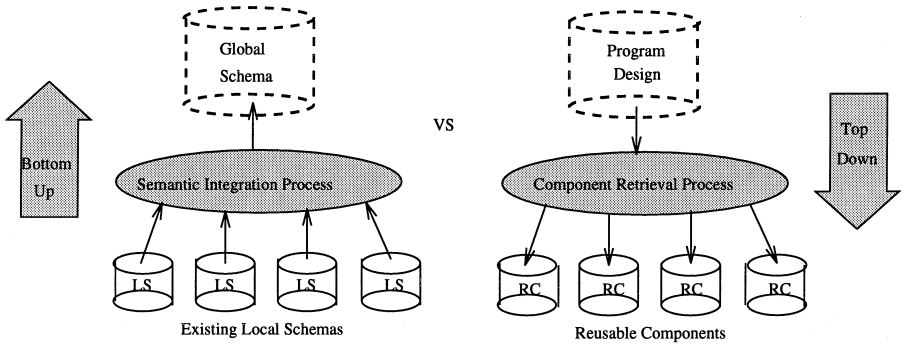

Fig. 21. Semantic integration process vs. component retrieval process.

SEMINT has also been extended and applied to the problem of component clustering retrieval in software reuse. We have developed a technique for constructing information systems, such as a data warehouse, based on component schema reuse, as described in [16]. Based on the concept of metadata extraction and clustering techniques, we also apply SEMINT to identify C program subroutines with similar structures, fan-in, and fan-out. In this application, a CASE tool is used to extract structural features and other signatures of C program sub-routines. The output of the CASE tool is then used as input for SEMINT. For more detail, see [3].

6. Conclusions and future research

We have presented a method that provides substantial assistance in identifying corresponding attributes in heterogeneous databases. SEMINT uses only information that can be automatically extracted from a database. One novel feature is that *how* this metadata characterizes attribute semantics is learned, not pre-programmed. This allows SEMINT to automatically develop different 'solutions' for different databases. The means of performing semantic integration is customized to the domain of the databases being integrated. This is done without human (or other outside) intervention; the 'domain knowledge' is *learned* directly from the database. This enables us to find attribute correspondences between databases without providing any advance domain knowledge; rather than substantial a priori effort to codify domain knowledge, users need only use their domain knowledge to verify the recommendations of SEMINT.

We discuss our experiences of database integration with SEMINT in different environments. SEMINT does a good job in identifying attribute correspondences as we can see how much metadata information that SEMINT is provided (in fact, SEMINT automatically extracts this metadata rather than being provided). We argue that in order to automate semantic integration process in different domains, explicitly expressing semantics as metadata and developing domain knowledge to incorporate all kinds of available information are essential to the success. This is also pointed out in [5,25].

Integrating heterogeneous databases is not an easy problem, particularly when dealing with multiple domains where no consistent terminology can be expected. The meaning of a term or attribute in a database can be ambiguous. Synonyms and homonyms are common. Even using the

same terms to describe concepts still does not assure consistency of factual information in databases [41]. Once the attribute correspondences are identified, more complex hierarchically structured correspondences can be constructed maybe through ontological engineering, such as described in [4,12].

There is still room for improvement in SEMINT, particularly in the neural network training process. We are working on combining SEMINT with the naming-based approach and applying domain specific ontology to improve SEMINTs performance. For example, when we examine a data item 'departure', its field name only tells us this data item is related to departure. We then find some city names in this field, say, 'Chicago' or 'ORD', we may conclude departure records departure city or airport, not departure date or time. On the other hand, if we find the contents in this field are dates, we may come up a different conclusion – 'departure' records departure date, not departure city. Also, if we find some first names such as 'John' exist in two fields, we may conclude these two fields are 'first name' even if their field names are not assigned with meaningful names and their schema are different, say, Char(10) and Varchar(20).

Another area for research is how we can use the ideas in SEMINT to assist in the problem of identifying heterogeneity. [42] pointed out different types of heterogeneity such as representation, scale, and level of abstraction. [30] shows how semantic values can either be stored explicitly or be defined by environments using conversion functions (e.g., *4(LengthUnit = 'feet')* and *48(LengthUnit = 'inches')*). This allows heterogeneity to be 'resolved' because of the explicit *conversion*. However, this does not tell us if both items refer to the same real-world class of information, e.g., one may be a width and the other a height, nor how to develop these conversions.

We may be able to develop an automatic means to 'suggest' that such a conversion is needed, however. For example, an extended SEMINT could determine that two attributes, say *annual_sale* and *monthly_sale*, are similar in schema design, *except* for the range of values. By examining their Coefficient of Variance (based on percentage rather than scale), SEMINT can suggest that *annual_sale* and *monthly_sale* are corresponding attributes with possible scale heterogeneity. We are incorporating scaling functions (possibly from a domain specific library of scaling functions) that would improve SEMINT's performance in presentation of heterogeneity.

Acknowledgements

We would like to express our appreciation to Larry Henschen and Peter Scheuermann at Northwestern University for their invaluable comments and suggestions on this work. We would like to thank Chuck Johnson at Boeing Computer Services for his assistance in running the Oracle7 parser and some experiments in this paper. We also thank Jim Presti, James Fulton, Pat Riddle, and Scott Smith at Boeing Computer Services for their helpful comments on the prototype of SEMINT. We would also like to thank Al Charity and Michael Morse at the National Science Foundation for making the NSF data available, and Yongjian Fu at Simon Fraser University for providing the NSERC data.

Appendix A. Self-organizing classifier algorithm

The detail of the self-organizing classifier algorithm used in this paper is:

Step 1: Present a new pattern $\underline{X} = X_i, i = 1, 2, \ldots, N$, where N is the number of characteristics compared. The values of X_i after normalization are between 0 and 1. Without normalization, the effect of one node with analog input (e.g., value range) may be much more significant than the effect of other nodes with binary input (e.g., key field or not). We call these values weights of the input pattern.

Step 2: Calculate the Euclidean distances from a pattern to existing cluster centers. If no cluster exists, go to Step 5.

Step 3: Choose the cluster nearest the pattern if the input pattern is within the threshold radius. If it is not within the threshold radius, go to Step 5.

Step 4: Include a new member in the cluster by recalculating the weights of this cluster center. Go to Step 6 if there is no input pattern left. Otherwise go back to Step 1

Step 5: Create a new cluster node. The weights of the new cluster center are the weights of the input pattern. Repeat with Step 1 until no input patterns remain.

Step 6: Each cluster is a category. The weights of cluster i are $\underline{C} = C_i, i = 1, 2, \ldots, M$, where M is the number of clusters (categories).

The key features of the self-organizing classifier algorithm are:

- Learning within a resonant state either refines the code of a previously established recognition code, based on any new information, or establishes a new code. For example, a new input is added. The system will search the established categories. If a match is found, the category representation is refined (by recalculating the new cluster center), if necessary, to incorporate the new pattern. If no match is found, a new category is formed; and
- The criterion for an adequate match between an input pattern and a particular category template is adjustable. The value of threshold determines how fine the categories will be. If the value of the threshold is set small, then confirmation requirements are high, and a body of patterns is likely to be split up into a large number of separate clusters. On the other hand, if the value of the threshold is set large, the same set of patterns might be organized into coarser recognition categories giving a small set of clusters. This allows us to adjust the 'sensitivity' of the system according to the availability of the database information.

Appendix B. Pattern learning and recognition algorithm

The algorithm for back-propagation training is:

Step 1: Start with an input data vector y_0 and initial values of the weight matrices W_1 and W_2. Small arbitrary random weights are chosen to initialize W_1 and W_2 (in $[-.5, .5]$). W_1 are the weights of the connections between the input layer and the hidden layer and W_2 are the weights of the connections between the hidden layer and the output layer.

Step 2: Perform forward propagation by calculating the following vectors:

$$net_1 = W_1 Y_0,$$

$$Y_1 = f_1(net_1),$$

$$net_2 = W_2 Y_1,$$

$$Y_2 = f_2(net_2),$$

where net_1 and net_2 are the input for the hidden layer and the output layer respectively and y_1 and y_2 are the output of the hidden layer and the output layer respectively. The net function $f_1()$ and $f_2()$ are defined as sigmoid functions.

Step 3: Perform back-propagation, by calculating:

$$\hat{\delta}_2 = d - Y_2,$$
$$\delta_2 = \hat{\delta}_2 * f_2'(net_2),$$
$$\hat{\delta}_1 = W_2^T \delta_2,$$
$$\delta_1 = \hat{\delta}^1 * f_1'(net_1),$$

where d is the desired output vector, $\hat{\delta}_2$ and $\hat{\delta}_1$ are the errors in the output and the hidden layers, δ_2 is the error propagating from the output layer back to the hidden layer and δ_1 is the error propagating from the hidden layer back to the input layer.

Step 4: Obtain the error gradient matrices:

$$\frac{\partial E}{\partial W_1} = -\delta_1 Y_0^T,$$
$$\frac{\partial E}{\partial W_2} = -\delta_2 Y_1^T.$$

Step 5: Adjust the weight matrices W_1 and W_2 by calculating:

$$\Delta W_i(t+1) = -G_i \frac{\partial E}{\partial W_i} + \alpha_i \Delta W_i(t) i = 1, 2,$$

where G_i are learning rates, and α_i are constants that determine the effect of past weights changes on the current direction of movement in weight space.

There are two variations used to adjust the weights during the training phase: *batch* replacement and *real time* replacement. *Batch* replacement is used in the paper.

Step 6: Go back to Step 2 until the output error is reduced below a pre-set threshold level. Sum-of-squares error is used as the measure of accuracy in this paper.

References

[1] S.S. Benkley, J.F. Fandozzi, E.M. Housman, G.M. Woodhouse. Data element tool-based analysis (DELTA). Technical Report MTR 95B0000147, The MITRE Corporation, Bedford, MA, December 1995.
[2] H. Chen, Machine learning for information retrieval:Neural networks symbolic learning and genetic algorithms, J. Am. Society for Info. Sci. 46 (3) (1995) 194–216.
[3] C. Clifton, W.-S. Li, Classifying software components using design characteristics, in: IEEE 10th Knowledge-Based Software Engineering Conference (KBSE-95), Boston, MA, USA, November 1995.
[4] C. Collet, M.N. Huhns, W.-M. Shen, Resource integration using a large knowledge base in carnot, Computer, December 1991, pp. 55–62.
[5] P. Drew, R. King, D. McLeod, M. Rusinkiewicz, A. Silberschatz, Report of the workshop on semantic heterogeneity and interoperation in multidatabase systems, SIGMOD Record, September 1993, pp. 47–56.
[6] S. Fahlman, Fast-learning variations on back-propagation: An empirical study, in: Proceedings of the 1988 Connectionist Models Summer School, 1988, pp. 38–51.

[7] D.H. Fisher, K.B. McKusick, An empirical comparison of id3 and backpropagation, in: Proceedings of the 11th Joint Conference of Artificial Intelligence, 1989, pp. 788–793.
[8] J. Grant, W. Litwin, N. Roussopoulos, T. Sellis, Query languages for relational multidatabases, VLDB J. 2 (1993) 153–171.
[9] S. Hayne, S. Ram, Multi-user view integration system (MUVIS): An expert system for view integration, in: Proceedings of the Sixth International Conference on Data Engineering, IEEE Press, New York, February 1990, pp. 402–409.
[10] T. Kohonen, Adaptive associative and self-organizing functions in neural computing, Appl. Optics 26 (1987) 4910–4918.
[11] J.A. Larson, S.B. Navathe, R. Elmasri, A theory of attribute equivalence in database with application to schema integration, IEEE Trans. Softw. Engrg. 15 (4) (1989) 449–463.
[12] D.B. Lenat, Cyc: A large-scale investment in knowledge infrastructure, Commun. ACM, New York, November 1995, pp. 33–38.
[13] W.-S. Li, C. Clifton, Using field specification to determine attribute equivalence in heterogeneous databases, in: Proceedings of RIDE-IMS '93, Third International Workshop on Research Issues in Data Engineering: Interoperability in Multidatabase Systems, Vienna, Austria, 19–20 April 1993, IEEE Press, New York, pp. 174–177.
[14] W.-S. Li, C. Clifton, Semantic integration in heterogeneous databases using neural networks, in: Proceedings of the 20th International Conference on Very Large Data Bases, Santiago, Chile, 12–15 September 1994, VLDB, pp. 1–12.
[15] W.-S. Li, C. Clifton, Semint: A system prototype for semantic integration in heterogeneous databases (demonstration description). in: Proceedings of the 1995 ACM SIGMOD Conference, San Jose, California, 23–25 May 1995.
[16] W.-S. Li, R.D. Holowczak, Constructing information systems based on schema reuse, in: Proceedings of the Fifth International Conference on Information and Knowledge Management (CIKM'96), November 1996, pp. 197–204.
[17] W. Litwin, A. Abdellatif, A. Zeroual, B. Nicolas, P. Vigier, MSQL: A multidatabase language, Info. Sci. 49 (1989) 59–101.
[18] D. McLeod, D. Heimbigner, A federated architecture for database systems, in: Proceedings of the National Computer Conference, Anaheim, CA, AFIPS Press, Reston, May 1980, pp. 283–289.
[19] G.A. Miller, WordNet: A Lexical Databases for English, Commun., ACM, New York, November 1995, pp. 39–41.
[20] A. Motro, P. Buneman, Constructing superviews, in: Proceeding of the SIGMOD International Conference on Management of Data, Ann Arbor, Michigan, April 1981, ACM, New York, pp. 56–64.
[21] S. Navathe, P. Buneman, Integrating user views in database design, Computer 19 (1) (1986) 50–62.
[22] G. Özsoyoglu, K. Du, A. Tjahjana, W.-C. Hou, D. Rowland, On estimating COUNT, SUM, and AVERAGE relational algebra queries, in: Proceedings of the Second International Conference on Database and Expert Systems Applications, IEEE Press, New York, 1991.
[23] C. Parent, S. Spaccapietra, Database integration: An overview of issues and approaches, Commun., 41(5es), ACM, New York, 1998.
[24] W.J. Premerlani, M.R. Blaha, An approach for reverse engineering of relational databases Commun. ACM 37 (5) (1994) 42–49.
[25] A. Rosenthal, L.J. Seligman, Data integration in the large: The challenge of reuse, in: Proceedings of the 20th International Conference on Very Large Data Bases, Santiago, Chile, 12–15 September 1994, VLDB, pp. 669–675.
[26] N. Rowe, An expert system for statistical estimates on databases, in: Proceedings of AAAI, 1983.
[27] D.E. Rumehart, G.E. Hinton, J.L. McClelland, Parallel Distributed Processing, Chapter A General Framework for Parallel Distributed Processing, MIT Press, Cambridge, MA, 1986, pp. 45–76.
[28] D.E. Rumehart, G.E. Hinton, R.J. Williams, Parallel Distributed Processing, Chapter Learning Internal Representations by Error Propagation, MIT Press, Cambridge, MA, 1986, pp. 318–362.
[29] D.E. Rumehart, B. Widrow, M.A. Lehr, The basic ideas in neural networks, Commun. ACM 37 (1994) 87–92.
[30] E. Sciore, M.D. Siegel, A. Rosenthal, Using semantic values to facilitate interoperability among heterogeneous information systems, ACM Trans. Database Syst. 19 (2) (1994) 254–290.
[31] A. Sheth, Managing and integrating unstructured data problem of representation, features, and abstraction, in: Proceedings of the Fourth International Conference on Data Engineering, Los Angeles, CA, IEEE Press, New York, February 1988, pp. 598–599.
[32] A. Sheth, S.K. Gala, Attribute relationships: An impediment in automating schema integration, in: Proceedings of the NSF Workshop on Heterogeneous Database Systems, Evanston, IL, December 1989.
[33] A. Sheth, V. Kashyap, So far (schematically) yet so near (semantically), in: Proceedings of the IFIP TC2/WG2.6 Conference on Semantics of Interoperable Database Systems, Victoria, Australia, November 1992.
[34] A. Sheth, J. Larson, Federated database systems for managing distributed heterogeneous and autonomous databases, ACM Comput. Surv. 22 (3) (1990) 183–236.
[35] A. Sheth, J. Larson, A. Cornelio, S.B. Navathe, A tool for integrating conceptual schemas and user views, in: Proceedings of the Fourth International Conference on Data Engineering, Los Angeles, CA, IEEE Press, New York, February 1988.
[36] A. Si, C.C. Ying, D. McLeod, On using historical update information for instance identification in federated databases, In First IFCIS International Conference on Cooperative Information Systems CoopIS'96 (1996) 68–77.
[37] J.M. Smith, P.A. Bernstein, U. Dayal, N. Goodman, T. Landers, T. Lin, E. Wang, Multibase – integrating heterogeneous distributed database systems, in: Proceeding of the National Computer Conference, AFIPS, 1981, pp. 487–499.
[38] U. Srinivasan, A.H.H. Ngu, T. Gedeon, Concept clustering for cooperation in heterogeneous information systems, in: Proceedings of the IFIP Workshop on Data Semantics, Stone Mountain, GA, 1995, Chapman & Hall, London, pp. 481–499.

[39] V. Ventrone, S. Heiler, Some advice for dealing with semantic heterogeneity in federated database systems, in: Proceedings of the Database Colloquium, San Diego, August 1994, Armed Forces Communications and Electronics Assc. (AFCEA).
[40] B. Widrow, D.E. Rumehart, M.A. Lehr, Neural networks: Applications in industry business and science, Commun. ACM 37 (1994) 93–105.
[41] G. Wiederhold, The roles of artificial intelligence in information systems, J. Intelligent Info. Sys. 1 (1) (1992) 35–55.
[42] G. Wiederhold, Intelligent integration of information, SIGMOD Record (1993) 434–437.
[43] G. Wiederhold, Digital libraries value and productivity, Commun. ACM (1995) 85–96.
[44] C. Yu, W. Sun, S. Dao, D. Keirsey, Determine relationships among attributes for interoperability of multi-database systems, in: Proceedings of RIDE-IMS, IEEE, New York, March 1991.
[45] J.L. Zhao, A. Segev, A. Chatterjee, A universal relation approach to federated database management, in: Proceedings of the 11th International Conference on Data Engineering, Taipei, Taiwan, March 1995, IEEE Press, New York, pp. 261–270.

Wen-Syan Li is currently a research staff member at Computers & Communications Research Laboratories (CCRL), NEC USA Inc. He received his Ph.D. in Computer Science from Northwestern University in December 1995. He also holds an MBA degree. His main research interests include multimedia/hypermedia/document databases, WWW, E-Commerce, and Internet/intranet search engines.

Chris Clifton is a Principal Scientist in the Information Technology Center at the MITRE Corporation. He has a Ph.D. from Princeton University, and Bachelor's and Master's degrees from the Massachusetts Institute of Technology. Prior to joining MITRE in 1995, he was an Assistant Professor of Computer Science at Northwestern University. His research interests include data mining, database support for text, and heterogeneous databases.