Bulletin of the Technical Committee on

Data Engineering

December 2000 Vol. 23 No. 4

IEEE Computer Society

Letters

Special Issue on Data Cleaning

Conference and Journal Notices

The Bulletin of the Technical Committee on Data Engineering is published quarterly and is distributed to all TC members. Its scope includes the design, implementation, modelling, theory and application of database systems and their technology.

Letters, conference information, and news should be sent to the Editor-in-Chief. Papers for each issue are solicited by and should be sent to the Associate Editor responsible for the issue.

Opinions expressed in contributions are those of the authors and do not necessarily reflect the positions of the TC on Data Engineering, the IEEE Computer Society, or the authors' organizations.

Membership in the TC on Data Engineering is open to all current members of the IEEE Computer Society who are interested in database systems.

The Data Engineering Bulletin web page is http://www.research.microsoft.com/research/db/debull.

Distribution
IEEE Computer Society
1730 Massachusetts Avenue
Washington, D.C. 20036-1992
(202) 371-1013
nschoultz@computer.org

Letter from the Editor-in-Chief

An Acknowledgment

As editor-in-chief of the Data Engineering Bulletin, I have been helped by the hard work of others. Mark Tuttle of Digital's Cambridge Research Lab defined the Latex style files that provide the clean and readable format of each issue. System support staff, at both Digital and Microsoft, have helped enable delivery of the Bulletin over the web. Both Digital and Microsoft have generously provided the infrastructure to support Bulletin distribution, as well as providing me with the editorial time.

I'd like to add to that list of acknowledgments.

- The style files used to generate the Bulletin work flawlessly only with a past version of Latex. I carefully maintain this version, but it is not readily available to associate editors or authors. The style file problem involves including an IEEE copyright notice, and arose after the style files were designed. Due to the efforts of **Do Hong Hai**, an author in this issue, this problem has been solved. Do Hong Hai modified one of the style files to permit the latest Latex system to successfully format the Bulletin. The result both simplifies my life (only the latest system is required) and makes it possible for associate editors and authors to easily check the formatting of the issue and the individual papers. This is the first issue produced with the new style files.

- The TCDE membership list is very large. In September, we experienced difficulty in sending email messages to very large distribution lists. As a result, the distribution list has been shifted to a web-based distribution list management tool at
`http://list.research.microsoft.com/scripts/lyris.pl?enter=debull`.
I would like to thank **Jeff Chirico** and **Wyman Chong** for providing the system support. The new email distribution tool makes it easier to update the distribution list, and TCDE members can remove themselves from the list if they so desire. As before, if your email address changes, you should contact me, mentioning both old and new email addresses.

The Current Issue

Heterogenous databases, with their syntactic and semantic differences, are an information legacy that will probably be with us always. Despite our efforts to define standards for information interchange, e.g., the current efforts with XML, bringing information together so that it can be sensibly queried remains a difficult and expensive task, usually requiring human assistence. This is the "data cleaning" problem. Data warehouses usually face this problem every day, in their role as repository for information derived from multiple sources within and across enterprises.

Data cleaning has not been a very active area of research. Perhaps the problem was too messy to deal with, too hard to attack with any generality. However, while research has paid little attention to data cleaning, industry has been forced to deal with this on a regular basis. This issue of the Bulletin addresses this long-standing and important problem. Sunita Sarawagi has herself worked in this important area, and has succeeded in soliciting papers from a mix of academia and industry, with papers from Europe and Asia, as well as the US. Thus the issue provides a very broad perspective on this important subject. Sunita has done a fine job as editor of the issue, and I want to thank her for her hard work.

David Lomet
Microsoft Corporation

Letter from the Special Issue Editor

Data cleaning is an important topic which for some reason has taken the backstage with database researchers. Whenever multiple operational data sources have to be consolidated in a single queryable system data cleaning becomes necessary. Two scenarios are: constructing data warehouses from operational databases distributed across an enterprise and, building queryable search engines from web pages on the internet. The proliferation of data on the web heightens the relevance of data cleaning and makes the problem more challenging because more sources imply more variety and higher complexity.

The practical importance of data cleaning is well reflected in the commercial marketplace in the form of the large number of companies providing data cleaning tools and services. However, the topic never caught momentum in the research world — maybe data cleaning is viewed as a primarily labor-intensive task. We motivate in this special issue that there is scope for elegant research in providing convenient platforms for data cleaning, creatively reducing dependence on manual effort and designing practical algorithms that scale with increasing data sizes.

The first paper by Erhard Rahm and Hong Hai Do surveys the field of data cleaning after neatly classifying the topics based on the source (single or multiple) and the location (schema level or instance level) of the error. A key step in all data cleaning activities is identifying duplicates in spite of the myriad manifestations of the same instance at different sources. We have two papers focusing on two different aspects of the problem. The second paper by Alvaro Monge addresses the duplicate detection problem at the record level, as is relevant in the context of warehouse data cleaning. The paper presents an algorithm for reducing the number of comparisons in previous window-based merge-purge algorithms. An interesting snippet in the paper is the survey of prior work on record matching with references dating back to 1950. The third paper by Krishna Bharat and others is on detecting mirror websites, which is duplicate detection at the level of collections of HTML documents. This is useful when building high-performance search engines and web caches. The paper presents and evaluates several interesting matching algorithms for finding potentially mirrored sites — the Shingles algorithm is particularly interesting. Another crucial step during data cleaning is extracting structure from data stored as an unstructured text string. In addition to providing more powerful querying interfaces, structured records also enable robust duplicate elimination. The fourth paper by Vinayak Borker and others concentrates on segmenting address records stored as a text string into structured fields like "city-name" and "zip-code". Existing commercial tools are based on manual transformation rules, whereas the paper presents an automated approach to learn to extract such structure from training examples. The fifth paper by Craig Knoblock and others is about extracting structure from HTML documents — a field that has attracted a lot of interest in recent years. This paper is one of the few I have seen that addresses the practical issues of detecting when the underlying HTML source has changed and doing limited repairs of the wrapper in the event of a change. The final paper by Panos Vassiliadis and others presents a holistic view of the data cleaning problem in the form of a tool for modeling and executing several data cleaning activities in a session.

Compiling this special issue on data cleaning was challenging because not too many researchers work in the area. Yet I wanted to unleash the special role of this bulletin in consolidating the few scattered research efforts on this topic. Hope the special issue will bring out data cleaning as an important topic in need for further concentrated research.

S. Sarawagi
IIT Bombay

Data Cleaning: Problems and Current Approaches

Erhard Rahm* Hong Hai Do
University of Leipzig, Germany
http://dbs.uni-leipzig.de

Abstract

We classify data quality problems that are addressed by data cleaning and provide an overview of the main solution approaches. Data cleaning is especially required when integrating heterogeneous data sources and should be addressed together with schema-related data transformations. In data warehouses, data cleaning is a major part of the so-called ETL process. We also discuss current tool support for data cleaning.

1 Introduction

Data cleaning, also called *data cleansing* or *scrubbing*, deals with detecting and removing errors and inconsistencies from data in order to improve the quality of data. Data quality problems are present in single data collections, such as files and databases, e.g., due to misspellings during data entry, missing information or other invalid data. When multiple data sources need to be integrated, e.g., in data warehouses, federated database systems or global web-based information systems, the need for data cleaning increases significantly. This is because the sources often contain redundant data in different representations. In order to provide access to accurate and consistent data, consolidation of different data representations and elimination of duplicate information become necessary.

Data warehouses [6, 16] require and provide extensive support for data cleaning. They load and continuously refresh huge amounts of data from a variety of sources so the probability that some of the sources contain "dirty data" is high. Furthermore, data warehouses are used for decision making, so that the correctness of their data is vital to avoid wrong conclusions. For instance, duplicated or missing information will produce incorrect or misleading statistics ("garbage in, garbage out"). Due to the wide range of possible data inconsistencies and the sheer data volume, data cleaning is considered to be one of the biggest problems in data warehousing. During the so-called ETL process (extraction, transformation, loading), illustrated in Fig. 1, further data transformations deal with schema/data translation and integration, and with filtering and aggregating data to be stored in the warehouse. As indicated in Fig. 1, all data cleaning is typically performed in a separate data staging area before loading the transformed data into the warehouse. A large number of tools of varying functionality is available to support these tasks, but often a significant portion of the cleaning and transformation work has to be done manually or by low-level programs that are difficult to write and maintain.

Federated database systems and web-based information systems face data transformation steps similar to those of data warehouses. In particular, there is typically a *wrapper* per data source for extraction and a *mediator* for integration [32, 31]. So far, these systems provide only limited support for data cleaning, focusing

Bulletin of the IEEE Computer Society Technical Committee on Data Engineering

*This work was performed while on leave at Microsoft Research, Redmond, WA.

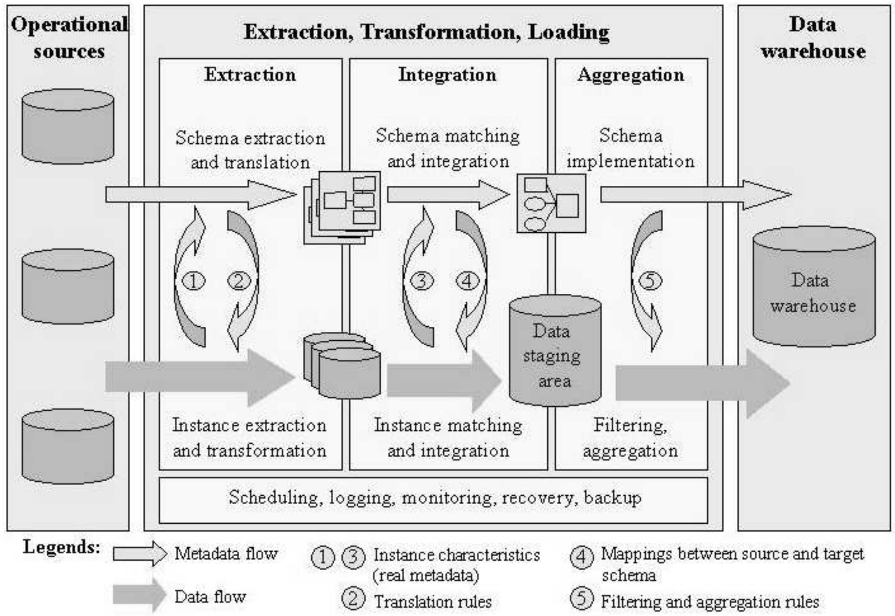

Figure 1: Steps of building a data warehouse: the ETL process

instead on data transformations for schema translation and schema integration. Data is not preintegrated as for data warehouses but needs to be extracted from multiple sources, transformed and combined during query runtime. The corresponding communication and processing delays can be significant, making it difficult to achieve acceptable response times. The effort needed for data cleaning during extraction and integration will further increase response times but is mandatory to achieve useful query results.

A data cleaning approach should satisfy several requirements. First of all, it should detect and remove all major errors and inconsistencies both in individual data sources and when integrating multiple sources. The approach should be supported by tools to limit manual inspection and programming effort and be extensible to easily cover additional sources. Furthermore, data cleaning should not be performed in isolation but together with schema-related data transformations based on comprehensive metadata. Mapping functions for data cleaning and other data transformations should be specified in a declarative way and be reusable for other data sources as well as for query processing. Especially for data warehouses, a workflow infrastructure should be supported to execute all data transformation steps for multiple sources and large data sets in a reliable and efficient way.

While a huge body of research deals with schema translation and schema integration, data cleaning has received only little attention in the research community. A number of authors focussed on the problem of duplicate identification and elimination, e.g., [11, 12, 15, 19, 22, 23]. Some research groups concentrate on general problems not limited but relevant to data cleaning, such as special data mining approaches [29, 30], and data transformations based on schema matching [1, 21]. More recently, several research efforts propose and investigate a more comprehensive and uniform treatment of data cleaning covering several transformation phases, specific operators and their implementation [11, 19, 25].

In this paper we provide an overview of the problems to be addressed by data cleaning and their solution. In the next section we present a classification of the problems. Section 3 discusses the main cleaning approaches used in available tools and the research literature. Section 4 gives an overview of commercial tools for data cleaning, including ETL tools. Section 5 is the conclusion.

2 Data cleaning problems

This section classifies the major data quality problems to be solved by data cleaning and data transformation. As we will see, these problems are closely related and should thus be treated in a uniform way. Data transformations [26] are needed to support any changes in the structure, representation or content of data. These transformations become necessary in many situations, e.g., to deal with schema evolution, migrating a legacy system to a new

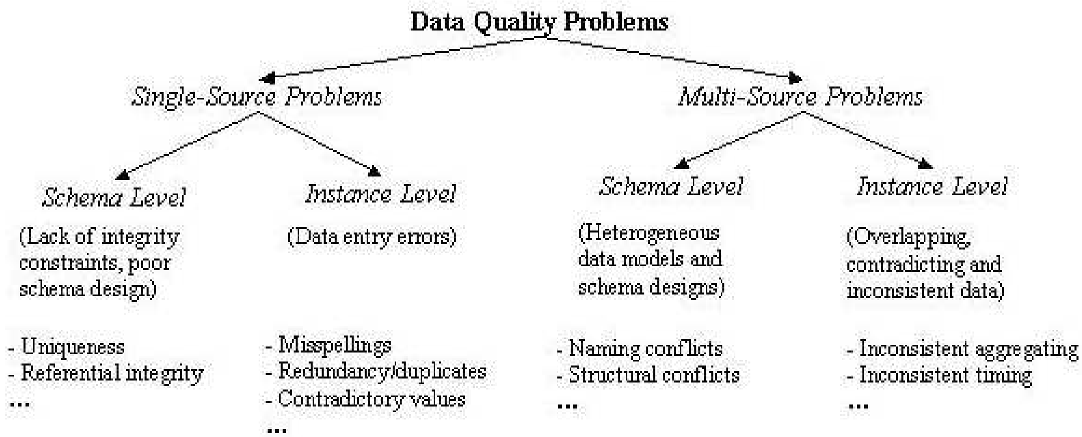

Figure 2: Classification of data quality problems in data sources

information system, or when multiple data sources are to be integrated.

As shown in Fig. 2 we roughly distinguish between single-source and multi-source problems and between schema- and instance-related problems. Schema-level problems of course are also reflected in the instances; they can be addressed at the schema level by an improved schema design (schema evolution), schema translation and schema integration. Instance-level problems, on the other hand, refer to errors and inconsistencies in the actual data contents which are not visible at the schema level. They are the primary focus of data cleaning. Fig. 2 also indicates some typical problems for the various cases. While not shown in Fig. 2, the single-source problems occur (with increased likelihood) in the multi-source case, too, besides specific multi-source problems.

2.1 Single-source problems

The data quality of a source largely depends on the degree to which it is governed by schema and integrity constraints controlling permissible data values. For sources without schema, such as files, there are few restrictions on what data can be entered and stored, giving rise to a high probability of errors and inconsistencies. Database systems, on the other hand, enforce restrictions of a specific data model (e.g., the relational approach requires simple attribute values, referential integrity, etc.) as well as application-specific integrity constraints. Schema-related data quality problems thus occur because of the lack of appropriate model-specific or application-specific integrity constraints, e.g., due to data model limitations or poor schema design, or because only a few integrity constraints were defined to limit the overhead for integrity control. Instance-specific problems relate to errors and inconsistencies that cannot be prevented at the schema level (e.g., misspellings).

Scope/Problem		**Dirty Data**	**Reasons/Remarks**
Attribute	Illegal values	bdate=30.13.70	values outside of domain range
Record	Violated attribute dependencies	age=22, bdate=12.02.70	age = current year - birth year should hold
Record type	Uniqueness violation	emp_1=(name="John Smith", SSN="123456"); emp_2=(name="Peter Miller", SSN="123456")	uniqueness for SSN (social security number) violated
Source	Referential integrity violation	emp=(name="John Smith", deptno=127)	referenced department (127) not defined

Table 1: Examples for single-source problems at schema level (violated integrity constraints)

For both schema- and instance-level problems we can differentiate different problem scopes: attribute (field), record, record type and source; examples for the various cases are shown in Tables 1 and 2. Note that uniqueness constraints specified at the schema level do not prevent duplicated instances, e.g., if information on the same real world entity is entered twice with different attribute values (see example in Table 2).

Given that cleaning data sources is an expensive process, preventing dirty data to be entered is obviously an important step to reduce the cleaning problem. This requires an appropriate design of the database schema and integrity constraints as well as of data entry applications. Also, the discovery of data cleaning rules during warehouse design can suggest improvements to the constraints enforced by existing schemas.

Scope/Problem		Dirty Data	Reasons/Remarks
Attribute	Missing values	phone=9999-999999	unavailable values during data entry (dummy values or null)
	Misspellings	city="Liipzig"	usually typos, phonetic errors
	Cryptic values, Abbreviations	experience="B"; occupation="DB Prog."	
	Embedded values	name="J. Smith 12.02.70 New York"	multiple values entered in one attribute (e.g. in a free-form field)
	Misfielded values	city="Germany"	
Record	Violated attribute dependencies	city="Redmond", zip=77777	city and zip code should correspond
Record type	Word transpositions	$name_1$= "J. Smith", $name_2$="Miller P."	usually in a free-form field
	Duplicated records	emp_1=(name="John Smith",...); emp_2=(name="J. Smith",...)	same employee represented twice due to some data entry errors
	Contradicting records	emp_1=(name="John Smith", bdate=12.02.70); emp_2=(name="John Smith", bdate=12.12.70)	the same real world entity is described by different values
Source	Wrong references	emp=(name="John Smith", deptno=17)	referenced department (17) is defined but wrong

Table 2: Examples for single-source problems at instance level

2.2 Multi-source problems

The problems present in single sources are aggravated when multiple sources need to be integrated. Each source may contain dirty data and the data in the sources may be represented differently, overlap or contradict. This is because the sources are typically developed, deployed and maintained independently to serve specific needs. This results in a large degree of heterogeneity w.r.t. data management systems, data models, schema designs and the actual data.

At the schema level, data model and schema design differences are to be addressed by the steps of schema translation and schema integration, respectively. The main problems w.r.t. schema design are naming and structural conflicts [2, 24, 17]. Naming conflicts arise when the same name is used for different objects (homonyms) or different names are used for the same object (synonyms). Structural conflicts occur in many variations and refer to different representations of the same object in different sources, e.g., attribute vs. table representation, different component structure, different data types, different integrity constraints, etc.

In addition to schema-level conflicts, many conflicts appear only at the instance level (data conflicts). All problems from the single-source case can occur with different representations in different sources (e.g., duplicated records, contradicting records,...). Furthermore, even when there are the same attribute names and data types, there may be different value representations (e.g., for marital status) or different interpretation of the values (e.g., measurement units Dollar vs. Euro) across sources. Moreover, information in the sources may be provided at different aggregation levels (e.g., sales per product vs. sales per product group) or refer to different points in time (e.g. current sales as of yesterday for source 1 vs. as of last week for source 2).

A main problem for cleaning data from multiple sources is to identify overlapping data, in particular matching records referring to the same real-world entity (e.g., customer). This problem is also referred to as the object identity problem [11], duplicate elimination or the merge/purge problem [15]. Frequently, the information is only partially redundant and the sources may complement each other by providing additional information about an entity. Thus duplicate information should be purged out and complementing information should be consolidated and merged in order to achieve a consistent view of real world entities.

The two sources in the example of Fig. 3 are both in relational format but exhibit schema and data conflicts. At the schema level, there are name conflicts (synonyms *Customer/Client, Cid/Cno, Sex/Gender*) and structural conflicts (different representations for names and addresses). At the instance level, we note that there are different gender representations ("0"/"1" vs. "F"/"M") and presumably a duplicate record (Kristen Smith). The latter observation also reveals that while *Cid/Cno* are both source-specific identifiers, their contents are not comparable between the sources; different numbers (11/493) may refer to the same person while different persons can have the same number (24). Solving these problems requires both schema integration and data cleaning; the third table shows a possible solution. Note that the schema conflicts should be resolved first to allow data

Customer (source 1)

CID	*Name*	*Street*	*City*	*Sex*
11	Kristen Smith	2 Hurley Pl	South Fork, MN 48503	0
24	Christian Smith	Hurley St 2	S Fork MN	1

Client (source 2)

Cno	*LastName*	*FirstName*	*Gender*	*Address*	*Phone/Fax*
24	Smith	Christoph	M	23 Harley St, Chicago IL, 60633-2394	333-222-6542 / 333-222-6599
493	Smith	Kris L.	F	2 Hurley Place, South Fork MN, 48503-5998	444-555-6666

Customers (integrated target with cleaned data)

No	*LName*	*FName*	*Gender*	*Street*	*City*	*State*	*ZIP*	*Phone*	*Fax*	*CID*	*Cno*
1	Smith	Kristen L.	F	2 Hurley Place	South Fork	MN	48503-5998	444-555-6666		11	493
2	Smith	Christian	M	2 Hurley Place	South Fork	MN	48503-5998			24	
3	Smith	Christoph	M	23 Harley Street	Chicago	IL	60633-2394	333-222-6542	333-222-6599		24

Figure 3: Examples of multi-source problems at schema and instance level

cleaning, in particular detection of duplicates based on a uniform representation of names and addresses, and matching of the *Gender/Sex* values.

3 Data cleaning approaches

In general, data cleaning involves several phases

- *Data analysis*: In order to detect which kinds of errors and inconsistencies are to be removed, a detailed data analysis is required. In addition to a manual inspection of the data or data samples, analysis programs should be used to gain metadata about the data properties and detect data quality problems.
- *Definition of transformation workflow and mapping rules*: Depending on the number of data sources, their degree of heterogeneity and the "dirtyness" of the data, a large number of data transformation and cleaning steps may have to be executed. Sometime, a schema translation is used to map sources to a common data model; for data warehouses, typically a relational representation is used. Early data cleaning steps can correct single-source instance problems and prepare the data for integration. Later steps deal with schema/data integration and cleaning multi-source instance problems, e.g., duplicates. For data warehousing, the control and data flow for these transformation and cleaning steps should be specified within a workflow that defines the ETL process (Fig. 1).
  The schema-related data transformations as well as the cleaning steps should be specified by a declarative query and mapping language as far as possible, to enable automatic generation of the transformation code. In addition, it should be possible to invoke user-written cleaning code and special-purpose tools during a data transformation workflow. The transformation steps may request user feedback on data instances for which they have no built-in cleaning logic.
- *Verification*: The correctness and effectiveness of a transformation workflow and the transformation definitions should be tested and evaluated, e.g., on a sample or copy of the source data, to improve the definitions if necessary. Multiple iterations of the analysis, design and verification steps may be needed, e.g., since some errors only become apparent after applying some transformations.
- *Transformation*: Execution of the transformation steps either by running the ETL workflow for loading and refreshing a data warehouse or during answering queries on multiple sources.
- *Backflow of cleaned data*: After (single-source) errors are removed, the cleaned data should also replace the dirty data in the original sources in order to give legacy applications the improved data too and to avoid redoing the cleaning work for future data extractions. For data warehousing, the cleaned data is available from the data staging area (Fig. 1).

The transformation process obviously requires a large amount of metadata, such as schemas, instance-level data characteristics, transformation mappings, workflow definitions, etc. For consistency, flexibility and ease of reuse, this metadata should be maintained in a DBMS-based repository [4]. To support data quality, detailed information about the transformation process is to be recorded, both in the repository and in the transformed instances, in particular information about the completeness and freshness of source data and lineage information about the origin of transformed objects and the changes applied to them. For instance, in Fig. 3, the derived table *Customers* contains the attributes *CID* and *Cno*, allowing one to trace back the source records.

In the following we describe in more detail possible approaches for data analysis (conflict detection), transformation definition and conflict resolution. For approaches to schema translation and schema integration, we refer to the literature as these problems have extensively been studied and described [2, 24, 26]. Name conflicts are typically resolved by renaming; structural conflicts require a partial restructuring and merging of the input schemas.

3.1 Data analysis

Metadata reflected in schemas is typically insufficient to assess the data quality of a source, especially if only a few integrity constraints are enforced. It is thus important to analyse the actual instances to obtain real (reengineered) metadata on data characteristics or unusual value patterns. This metadata helps finding data quality problems. Moreover, it can effectively contribute to identify attribute correspondences between source schemas (schema matching), based on which automatic data transformations can be derived [20, 9].

There are two related approaches for data analysis, data profiling and data mining. *Data profiling* focusses on the instance analysis of individual attributes. It derives information such as the data type, length, value range, discrete values and their frequency, variance, uniqueness, occurrence of null values, typical string pattern (e.g., for phone numbers), etc., providing an exact view of various quality aspects of the attribute. Table 3 shows examples of how this metadata can help detecting data quality problems.

Problems	**Metadata**	**Examples/Heuristics**
Illegal values	cardinality	e.g., cardinality (gender) $>$ 2 indicates problem
	max, min	max, min should not be outside of permissible range
	variance, deviation	variance, deviation of statistical values should not be higher than threshold
Misspellings	attribute values	sorting on values often brings misspelled values next to correct values
Missing values	null values	percentage/number of null values
	attribute values + default values	presence of default value may indicate real value is missing
Varying value representations	attribute values	comparing attribute value set of a column of one table against that of a column of another table
Duplicates	cardinality + uniqueness	attribute cardinality = # rows should hold
	attribute values	sorting values by number of occurrences; more than 1 occurrence indicates duplicates

Table 3: Examples for the use of reengineered metadata to address data quality problems

Data mining helps discover specific data patterns in large data sets, e.g., relationships holding between several attributes. This is the focus of so-called descriptive data mining models including clustering, summarization, association discovery and sequence discovery [10]. As shown in [28], integrity constraints among attributes such as functional dependencies or application-specific "business rules" can be derived, which can be used to complete missing values, correct illegal values and identify duplicate records across data sources. For example, an association rule with high confidence can hint to data quality problems in instances violating this rule. So a confidence of 99% for rule "$total = quantity * unit\ price$" indicates that 1% of the records do not comply and may require closer examination.

3.2 Defining data transformations

The data transformation process typically consists of multiple steps where each step may perform schema- and instance-related transformations (mappings). To allow a data transformation and cleaning system to generate transformation code and thus to reduce the amount of self-programming it is necessary to specify the required transformations in an appropriate language, e.g., supported by a graphical user interface. Various ETL tools (see Section 4) offer this functionality by supporting proprietary rule languages. A more general and flexible approach is the use of the standard query language SQL to perform the data transformations and utilize the possibility of application-specific language extensions, in particular user-defined functions (UDFs) supported in SQL:99 [13, 14]. UDFs can be implemented in SQL or a general-purpose programming language with embedded SQL statements. They allow implementing a wide range of data transformations and support easy reuse for different transformation and query processing tasks. Furthermore, their execution by the DBMS can reduce data access cost and thus improve performance. Finally, UDFs are part of the SQL:99 standard and should (eventually) be portable across many platforms and DBMSs.

CREATE VIEW	Customer2 (LName, FName, Gender, Street, City, State, ZIP, CID)
AS SELECT	**LastNameExtract** (Name), **FirstNameExtract** (Name), Sex, Street, **CityExtract** (City), **StateExtract** (City), **ZIPExtract** (City), CID
FROM	Customer

Figure 4: Example of data transformation mapping

Fig. 4 shows a transformation step specified in SQL:99. The example refers to Fig. 3 and covers part of the necessary data transformations to be applied to the first source. The transformation defines a view on which further mappings can be performed. The transformation performs a schema restructuring with additional attributes in the view obtained by splitting the name and address attributes of the source. The required data extractions are achieved by UDFs (shown in boldface). The UDF implementations can contain cleaning logic, e.g., to remove misspellings in city names or provide missing zip codes.

UDFs may still imply a substantial implementation effort and do not support all necessary schema transformations. In particular, simple and frequently needed functions such as attribute splitting or merging are not generically supported but need often to be re-implemented in application-specific variations (see specific extract functions in Fig. 4). More complex schema restructurings (e.g., folding and unfolding of attributes) are not supported at all. To generically support schema-related transformations, language extensions such as the SchemaSQL proposal are required [18]. Data cleaning at the instance level can also benefit from special language extensions such as a Match operator supporting ”approximate joins” (see below). System support for such powerful operators can greatly simplify the programming effort for data transformations and improve performance. Some current research efforts on data cleaning are investigating the usefulness and implementation of such query language extensions [11, 25].

3.3 Conflict resolution

A set of transformation steps has to be specified and executed to resolve the various schema- and instance-level data quality problems that are reflected in the data sources at hand. Several types of transformations are to be performed on the individual data sources in order to deal with single-source problems and to prepare for integration with other sources. In addition to a possible schema translation, these preparatory steps typically include:

- *Extracting values from free-form attributes (attribute split)*: Free-form attributes often capture multiple individual values that should be extracted to achieve a more precise representation and support further cleaning steps such as instance matching and duplicate elimination. Typical examples are name and address fields (Table 2, Fig. 3, Fig. 4). Required transformations in this step are reordering of values within a field to deal with word transpositions, and value extraction for attribute splitting.
- *Validation and correction*: This step examines each source instance for data entry errors and tries to correct them automatically as far as possible. Spell checking based on dictionary lookup is useful for identifying and correcting misspellings. Furthermore, dictionaries on geographic names and zip codes help to correct address data. Attribute dependencies (birthdate - age, total price - unit price / quantity, city - phone area code,. . .) can be utilized to detect problems and substitute missing values or correct wrong values.
- *Standardization*: To facilitate instance matching and integration, attribute values should be converted to a consistent and uniform format. For example, date and time entries should be brought into a specific format; names and other string data should be converted to either upper or lower case, etc. Text data may be condensed and unified by performing stemming, removing prefixes, suffixes, and stop words. Further-

more, abbreviations and encoding schemes should consistently be resolved by consulting special synonym dictionaries or applying predefined conversion rules.

Dealing with multi-source problems requires restructuring of schemas to achieve a schema integration, including steps such as splitting, merging, folding and unfolding of attributes and tables. At the instance level, conflicting representations need to be resolved and overlapping data must to be dealt with. The *duplicate elimination* task is typically performed after most other transformation and cleaning steps, especially after having cleaned single-source errors and conflicting representations. It is performed either on two cleaned sources at a time or on a single already integrated data set. Duplicate elimination requires to first identify (i.e. match) similar records concerning the same real world entity. In a second step, similar records are merged into one record containing all relevant attributes without redundancy. Furthermore, redundant records are purged. In the following we discuss the key problem of instance matching. More details on the subject are provided elsewhere in this issue [22].

In the simplest case, there is an identifying attribute or attribute combination per record that can be used for matching records, e.g., if different sources share the same primary key or if there are other common unique attributes. Instance matching between different sources is then achieved by a standard equi-join on the identifying attribute(s). In the case of a single data set, matches can be determined by sorting on the identifying attribute and checking if neighboring records match. In both cases, efficient implementations can be achieved even for large data sets. Unfortunately, without common key attributes or in the presence of dirty data such straightforward approaches are often too restrictive. To determine most or all matches a "fuzzy matching" (approximate join) becomes necessary that finds similar records based on a matching rule, e.g., specified declaratively or implemented by a user-defined function [14, 11]. For example, such a rule could state that person records are likely to correspond if name and portions of the address match. The degree of similarity between two records, often measured by a numerical value between 0 and 1, usually depends on application characteristics. For instance, different attributes in a matching rule may contribute different weight to the overall degree of similarity. For string components (e.g., customer name, company name,) exact matching and fuzzy approaches based on wildcards, character frequency, edit distance, keyboard distance and phonetic similarity (soundex) are useful [11, 15, 19]. More complex string matching approaches also considering abbreviations are presented in [23]. A general approach for matching both string and text data is the use of common information retrieval metrics. WHIRL represents a promising representative of this category using the cosine distance in the vector-space model for determining the degree of similarity between text elements [7].

Determining matching instances with such an approach is typically a very expensive operation for large data sets. Calculating the similarity value for any two records implies evaluation of the matching rule on the cartesian product of the inputs. Furthermore sorting on the similarity value is needed to determine matching records covering duplicate information. All records for which the similarity value exceeds a threshold can be considered as matches, or as match candidates to be confirmed or rejected by the user. In [15] a multi-pass approach is proposed for instance matching to reduce the overhead. It is based on matching records independently on different attributes and combining the different match results. Assuming a single input file, each match pass sorts the records on a specific attribute and only tests nearby records within a certain window on whether they satisfy a predetermined matching rule. This reduces significantly the number of match rule evaluations compared to the cartesian product approach. The total set of matches is obtained by the union of the matching pairs of each pass and their transitive closure.

4 Tool support

A large variety of tools is available on the market to support data transformation and data cleaning tasks, in particular for data warehousing.[†] Some tools concentrate on a specific domain, such as cleaning name and address data, or a specific cleaning phase, such as data analysis or duplicate elimination. Due to their restricted domain, specialized tools typically perform very well but must be complemented by other tools to address the broad spectrum of transformation and cleaning problems. Other tools, e.g., ETL tools, provide comprehensive transformation and workflow capabilities to cover a large part of the data transformation and cleaning process.

[†]For comprehensive vendor and tool listings, see commercial websites, e.g., Data Warehouse Information Center (www.dwinfocenter.org), Data Management Review (www.dmreview.com), Data Warehousing Institute (www.dw-institute.com)

A general problem of ETL tools is their limited interoperability due to proprietary application programming interfaces (API) and proprietary metadata formats making it difficult to combine the functionality of several tools [8].

We first discuss tools for data analysis and data rengineering which process instance data to identify data errors and inconsistencies, and to derive corresponding cleaning transformations. We then present specialized cleaning tools and ETL tools, respectively.

4.1 Data analysis and reengineering tools

According to our classification in 3.1, *data analysis tools* can be divided into data profiling and data mining tools. MIGRATIONARCHITECT (EvokeSoftware) is one of the few commercial *data profiling tools*. For each attribute, it determines the following real metadata: data type, length, cardinality, discrete values and their percentage, minimum and maximum values, missing values, and uniqueness. MIGRATIONARCHITECT also assists in developing the target schema for data migration. *Data mining tools*, such as WIZRULE (WizSoft) and DATAMININGSUITE (InformationDiscovery), infer relationships among attributes and their values and compute a confidence rate indicating the number of qualifying rows. In particular, WIZRULE can reveal three kinds of rules: mathematical formula, if-then rules, and spelling-based rules indicating misspelled names, e.g., "$value$ Edinburgh $appears$ 52 $times\ in\ field$ Customer; 2 $case(s)\ contain\ similar\ value(s)$". WIZRULE also automatically points to the deviations from the set of the discovered rules as suspected errors.

Data reengineering tools, e.g., INTEGRITY (Vality), utilize discovered patterns and rules to specify and perform cleaning transformations, i.e., they reengineer legacy data. In INTEGRITY, data instances undergo several analysis steps, such as parsing, data typing, pattern and frequency analysis. The result of these steps is a tabular representation of field contents, their patterns and frequencies, based on which the pattern for standardizing data can be selected. For specifying cleaning transformations, INTEGRITY provides a language including a set of operators for column transformations (e.g., move, split, delete) and row transformation (e.g., merge, split). INTEGRITY identifies and consolidates records using a statistical matching technique. Automated weighting factors are used to compute scores for ranking matches based on which the user can select the real duplicates.

4.2 Specialized cleaning tools

Specialized cleaning tools typically deal with a particular domain, mostly name and address data, or concentrate on duplicate elimination. The transformations are to be provided either in advance in the form of a rule library or interactively by the user. Alternatively, data transformations can automatically be derived from schema matching tools such as described in [21].

- *Special domain cleaning*: Names and addresses are recorded in many sources and typically have high cardinality. For example, finding customer matches is very important for customer relationship management. A number of commercial tools, e.g., IDCENTRIC (FirstLogic), PUREINTEGRATE (Oracle), QUICKADDRESS (QASSystems), REUNION (PitneyBowes), and TRILLIUM (TrilliumSoftware), focus on cleaning this kind of data. They provide techniques such as extracting and transforming name and address information into individual standard elements, validating street names, cities, and zip codes, in combination with a matching facility based on the cleaned data. They incorporate a huge library of pre-specified rules dealing with the problems commonly found in processing this data. For example, TRILLIUM's extraction (parser) and matcher module contains over 200,000 business rules. The tools also provide facilities to customize or extend the rule library with user-defined rules for specific needs.
- *Duplicate elimination*: Sample tools for duplicate identification and elimination include DATACLEANSER (EDD), MERGE/PURGELIBRARY (Sagent/QMSoftware), MATCHIT (HelpITSystems), and MASTERMERGE (PitneyBowes). Usually, they require the data sources already be cleaned for matching. Several approaches for matching attribute values are supported; tools such as DATACLEANSER and MERGE/PURGELIBRARY also allow user-specified matching rules to be integrated.

4.3 ETL tools

A large number of commercial tools support the ETL process for data warehouses in a comprehensive way, e.g., COPYMANAGER (InformationBuilders), DATASTAGE (Informix/Ardent), EXTRACT (ETI), POWERMART (Informatica), DECISIONBASE (CA/Platinum), DATATRANSFORMATIONSERVICE (Microsoft), METASUITE

(Minerva/Carleton), SAGENTSOLUTIONPLATFORM (Sagent) and WAREHOUSEADMINISTRATOR (SAS). They use a repository built on a DBMS to manage all metadata about the data sources, target schemas, mappings, script programs, etc., in a uniform way. Schemas and data are extracted from operational data sources via both native file and DBMS gateways as well as standard interfaces such as ODBC and EDA. Data transformations are defined with an easy-to-use graphical interface. To specify individual mapping steps, a proprietary rule language and a comprehensive library of predefined conversion functions are typically provided. The tools also support reusing existing transformation solutions, such as external C/C++ routines, by providing an interface to integrate them into the internal transformation library. Transformation processing is carried out either by an engine that interprets the specified transformations at runtime, or by compiled code. All engine-based tools (e.g., COPYMANAGER, DECISIONBASE, POWERMART, DATASTAGE, WAREHOUSEADMINISTRATOR), possess a scheduler and support workflows with complex execution dependencies among mapping jobs. A workflow may also invoke external tools, e.g., for specialized cleaning tasks such as name/address cleaning or duplicate elimination.

ETL tools typically have little built-in data cleaning capabilities but allow the user to specify cleaning functionality via a proprietary API. There is usually no data analysis support to automatically detect data errors and inconsistencies. However, users can implement such logic with the metadata maintained and by determining content characteristics with the help of aggregation functions (sum, count, min, max, median, variance, deviation,). The provided transformation library covers many data transformation and cleaning needs, such as data type conversions (e.g., date reformatting), string functions (e.g., split, merge, replace, sub-string search), arithmetic, scientific and statistical functions, etc. Extraction of values from free-form attributes is not completely automatic but the user has to specify the delimiters separating sub-values.

The rule languages typically cover *if-then* and *case* constructs that help handling exceptions in data values, such as misspellings, abbreviations, missing or cryptic values, and values outside of range. These problems can also be addressed by using a table lookup construct and join functionality. Support for instance matching is typically restricted to the use of the join construct and some simple string matching functions, e.g., exact or wildcard matching and soundex. However, user-defined field matching functions as well as functions for correlating field similarities can be programmed and added to the internal transformation library.

5 Conclusions

We provided a classification of data quality problems in data sources differentiating between single- and multi-source and between schema- and instance-level problems. We further outlined the major steps for data transformation and data cleaning and emphasized the need to cover schema- and instance-related data transformations in an integrated way. Furthermore, we provided an overview of commercial data cleaning tools. While the state-of-the-art in these tools is quite advanced, they do typically cover only part of the problem and still require substantial manual effort or self-programming. Furthermore, their interoperability is limited (proprietary APIs and metadata representations).

So far only a little research has appeared on data cleaning, although the large number of tools indicates both the importance and difficulty of the cleaning problem. We see several topics deserving further research. First of all, more work is needed on the design and implementation of the best language approach for supporting both schema and data transformations. For instance, operators such as Match, Merge or Mapping Composition have either been studied at the instance (data) or schema (metadata) level but may be built on similar implementation techniques. Data cleaning is not only needed for data warehousing but also for query processing on heterogeneous data sources, e.g., in web-based information systems. This environment poses much more restrictive performance constraints for data cleaning that need to be considered in the design of suitable approaches. Furthermore, data cleaning for semi-structured data, e.g., based on XML, is likely to be of great importance given the reduced structural constraints and the rapidly increasing amount of XML data.

Acknowledgments
We would like to thank Phil Bernstein, Helena Galhardas and Sunita Sarawagi for helpful comments.

References

[1] Abiteboul, S.; Clue, S.; Milo, T.; Mogilevsky, P.; Simeon, J.: *Tools for Data Translation and Integration*. In [26]:3-8, 1999.

[2] Batini, C.; Lenzerini, M.; Navathe, S.B.: *A Comparative Analysis of Methodologies for Database Schema Integration*. In Computing Surveys 18(4):323-364, 1986.

[3] Bernstein, P.A.; Bergstraesser, T.: *Metadata Support for Data Transformation Using Microsoft Repository*. In [26]:9-14, 1999

[4] Bernstein, P.A.; Dayal, U.: *An Overview of Repository Technology*. Proc. 20th VLDB, 1994.

[5] Bouzeghoub, M.; Fabret, F.; Galhardas, H.; Pereira, J; Simon, E.; Matulovic, M.: *Data Warehouse Refreshment*. In [16]:47-67.

[6] Chaudhuri, S., Dayal, U.: *An Overview of Data Warehousing and OLAP Technology*. ACM SIGMOD Record 26(1), 1997.

[7] Cohen, W.: *Integration of Heterogeneous Databases without Common Domains Using Queries Based Textual Similarity*. Proc. ACM SIGMOD Conf. on Data Management, 1998.

[8] Do, H.H.; Rahm, E.: *On Metadata Interoperability in Data Warehouses*. Techn. Report 1-2000, Department of Computer Science, University of Leipzig. http://dol.uni-leipzig.de/pub/2000-13.

[9] Doan, A.H.; Domingos, P.; Levy, A.Y.: *Learning Source Description for Data Integration*. Proc. 3rd Intl. Workshop The Web and Databases (WebDB), 2000.

[10] Fayyad, U.: *Mining Database: Towards Algorithms for Knowledge Discovery*. IEEE Techn. Bulletin Data Engineering 21(1), 1998.

[11] Galhardas, H.; Florescu, D.; Shasha, D.; Simon, E.: *Declaratively cleaning your data using AJAX*. In Journees Bases de Donnees, Oct. 2000. http://caravel.inria.fr/ galharda/BDA.ps.

[12] Galhardas, H.; Florescu, D.; Shasha, D.; Simon, E.: *AJAX: An Extensible Data Cleaning Tool*. Proc. ACM SIGMOD Conf., p. 590, 2000.

[13] Haas, L.M.; Miller, R.J.; Niswonger, B.; Tork Roth, M.; Schwarz, P.M.; Wimmers, E.L.: *Transforming Heterogeneous Data with Database Middleware: Beyond Integration*. In [26]:31-36, 1999.

[14] Hellerstein, J.M.; Stonebraker, M.; Caccia, R.: *Independent, Open Enterprise Data Integration*. In [26]:43-49, 1999.

[15] Hernandez, M.A.; Stolfo, S.J.: *Real-World Data is Dirty: Data Cleansing and the Merge/Purge Problem*. Data Mining and Knowledge Discovery 2(1):9-37, 1998.

[16] Jarke, M., Lenzerini, M., Vassiliou, Y., Vassiliadis, P.: *Fundamentals of Data Warehouses*. Springer, 2000.

[17] Kashyap, V.; Sheth, A.P.: *Semantic and Schematic Similarities between Database Objects: A Context-Based Approach*. VLDB Journal 5(4):276-304, 1996.

[18] Lakshmanan, L.; Sadri, F.; Subramanian, I.N.: *SchemaSQL - A Language for Interoperability in Relational Multi-Database Systems*. Proc. 26th VLDB, 1996.

[19] Lee, M.L.; Lu, H.; Ling, T.W.; Ko, Y.T.: *Cleansing Data for Mining and Warehousing*. Proc. 10th DEXA, 1999.

[20] Li, W.S.; Clifton, S.: *SEMINT: A Tool for Identifying Attribute Correspondences in Heterogeneous Databases Using Neural Networks*. In Data and Knowledge Engineering 33(1):49-84, 2000.

[21] Milo, T.; Zohar, S.: *Using Schema Matching to Simplify Heterogeneous Data Translation*. Proc. 24th VLDB, 1998.

[22] Monge, A. E.: *Matching Algorithm within a Duplicate Detection System*. IEEE Techn. Bulletin Data Engineering 23(4), 2000 (this issue).

[23] Monge, A. E.; Elkan, P.C.: *The Field Matching Problem: Algorithms and Applications*. Proc. 2nd Intl. Conf. Knowledge Discovery and Data Mining (KDD), 1996.

[24] Parent, C.; Spaccapietra, S.: *Issues and Approaches of Database Integration*. Comm. ACM 41(5):166-178, 1998.

[25] Raman, V.; Hellerstein, J.M.: *Potter's Wheel: An Interactive Framework for Data Cleaning*. Working Paper, 1999. http://www.cs.berkeley.edu/ rshankar/papers/pwheel.pdf.

[26] Rundensteiner, E. (ed.): Special Issue on Data Transformation. IEEE Techn. Bull. Data Engineering 22(1), 1999.

[27] Quass, D.: *A Framework for Research in Data Cleaning*. Unpublished Manuscript. Brigham Young Univ., 1999

[28] Sapia, C.; Höfling, G.; Müller, M.; Hausdorf, C.; Stoyan, H.; Grimmer, U.: *On Supporting the Data Warehouse Design by Data Mining Techniques*. Proc. GI-Workshop Data Mining and Data Warehousing, 1999.

[29] Savasere, A.; Omiecinski, E.; Navathe, S.: *An Efficient Algorithm for Mining Association Rules in Large Databases*. Proc. 21st VLDB, 1995.

[30] Srikant, R.; Agrawal, R.: *Mining Generalized Association Rules*. Proc. 21st VLDB conf., 1995.

[31] Tork Roth, M.; Schwarz, P.M.: *Don't Scrap It, Wrap It! A Wrapper Architecture for Legacy Data Sources*. Proc. 23rd VLDB, 1997.

[32] Wiederhold, G.: *Mediators in the Architecture of Future Information Systems*. In IEEE Computer 25(3): 38-49, 1992.

Matching Algorithms within a Duplicate Detection System

Alvaro E. Monge
California State University Long Beach
Computer Engineering and Computer Science Department,
Long Beach, CA, 90840-8302

Abstract

Detecting database records that are approximate duplicates, but not exact duplicates, is an important task. Databases may contain duplicate records concerning the same real-world entity because of data entry errors, unstandardized abbreviations, or differences in the detailed schemas of records from multiple databases – such as what happens in data warehousing where records from multiple data sources are integrated into a single source of information – among other reasons. In this paper we review a system to detect approximate duplicate records in a database and provide properties that a pair-wise record matching algorithm must have in order to have a successful duplicate detection system.

1 Introduction

Many of the current technological improvements has lead to an explosion in the growth of data available in digital form. The most significant of these being the popularity of the Internet and specifically the world wide web. There are other more traditional sources of data however that has also contributed to this exponential growth. The commercial success of relational databases in the early 1980's has lead to the efficient storage and retrieval of data. Hardware improvements have also contributed significantly now that external storage has faster access times and continue to increase in density.

Such technological changes allow for easy and widespread distribution and publishing of data. The availability of these data sources increases not only the amount of data, but also the variety of and quality in which such data appears. These factors create a number of problems. The work presented here concentrates on one such problem: the detection of multiple representations of a single entity. Following are examples where solutions to such a problem are needed:

- As more and more data becomes available, it is desirable to integrate the data whenever possible. For example, one data source may contain bibliographic data for published scientific papers. Another data source may contain a white pages service for web pages of people. By integrating the data from these two sources, one can go directly to the authors' web pages to obtain information about the more recent work by the authors. Such an integration is much like the *join* operation in relational databases [1, 2]. While the join of two relations is based on an exact match of corresponding attributes, here we relax that condition and allow for an approximate match.

Bulletin of the IEEE Computer Society Technical Committee on Data Engineering

- Another typical example is the prevalent practice in the mass mail market of buying and selling mailing lists. Such practice leads to inaccurate or inconsisten data. One inconsistency is the multiple representations of the same individual household in the combined mailing list. In the mass mailing market, this leads to expensive and wasteful multiple mailings to the same household

Relational database systems disallow the entry of records containing duplicate primary key values. Unfortunately, due to errors such as from data entry, whenever the value of the primary key attributes is affected by these errors the relational database system can no longer guarantee the non-existence of duplicate records.

The next section reviews the system for detecting approximate duplicate database records [3, 4, 5]. One important module in the system is the algorithm to detect pair-wise approximate duplicates. The system uses the algorithm to establish "is a duplicate of" relationships between pairs of records leading to its adaptability among different data sources and different dynamics of the database records. Section 2 looks at the different matching algorithms that could be plugged in to the duplicate detection system. The article concludes in Section 4 with final remarks about this work.

2 Algorithms to match records

The word *record* is used to mean a syntactic designator of some real-world object, such as a tuple in a relational database. The record matching problem arises whenever records that are not identical, in a bit-by-bit sense – or in a primary key value sense – may still refer to the same object. For example, one database may store the first name and last name of a person (e.g. "Jane Doe"), while another database may store only the initials and the last name of the person (e.g. "J. B. Doe").

The record matching problem has been recognized as important for at least 50 years. Since the 1950s over 100 papers have studied matching for medical records under the name "record linkage." These papers are concerned with identifying medical records for the same individual in different databases, for the purpose of performing epidemiological studies [16]. Record matching has also been recognized as important in business for decades. For example tax agencies must do record matching to correlate different pieces of information about the same taxpayer when social security numbers are missing or incorrect. The earliest paper on duplicate detection in a business database is by [17]. The "record linkage" problem in business has been the focus of workshops sponsored by the US Census Bureau [12, 18]. Record matching is also useful for detecting fraud and money laundering [19].

Almost all published previous work on record matching is for specific application domains, and hence gives domain-specific algorithms. For example, papers discuss record matching for customer addresses, census records, or variant entries in a lexicon. Other work on record matching is not domain-specific, but assumes that domain-specific knowledge will be supplied by a human for each application domain [20, 7].

Record matching algorithms vary by the amount of domain-specific knowledge that they use. The pairwise record matching algorithms used in most previous work have been application-specific. Many algorithms use production rules based on domain-specific knowledge. The process of creating such rules can be time consuming and the rules must be continually updated whenever new data is added to the mix that does not follow the patterns by which the rules were originally created. Another disadvantage of these domain-specific rules is that they answer whether or not the records are or are not duplicates, there is no in between.

The goal of this work is to create a detection system that is of general use. Record matching algorithms should use as little domain-specific knowledge as possible and should also provide a measure of the strength of the match. This information is crucial to the efficiency with which the detection system can process the data.

2.1 The record matching problem

In this work, we say that two records are *equivalent* if they are equal semantically, that is if they both designate the same real-world entity. Semantically, this problem respects the reflexivity, symmetry, and transitivity properties. The record matching algorithms which solve this problem depend on the syntax of the records. The syntactic calculations performed by the algorithms are approximations of what we really want – semantic equivalence. In such calculations, errors are bound to occur and thus the semantic equivalence will not be properly calculated. However, the claim is that there are few errors and that the approximation is good.

Equivalence may sometimes be a question of degree, so a function solving the record matching problem returns a value between 0.0 and 1.0, where 1.0 means certain equivalence and 0.0 means certain non-equivalence. Degree of match scores are not necessarily probabilities or fuzzy degrees of truth. An application will typically just compare scores to a threshold that depends on the domain and the particular record matching algorithm in use.

2.2 Algorithms based on approximate string matching

One important area of research that is relevant to approximate record matching is approximate string matching. String matching has been one of the most studied problems in computer science [21, 22, 23, 24, 25, 26]. The main approach is based on edit distance [27]. Edit distance is the minimum number of operations on individual characters (e.g. substitutions, insertions, and deletions) needed to transform one string of symbols to another [28, 23]. In [23], the authors consider two different problems, one under the definition of equivalence and a second using similarity. Their definition of equivalence allows only small differences in the two strings. For examples, they allow alternate spellings of the same word, and ignore the case of letters. The similarity problem allows for more errors, such as those due to typing: transposed letters, missing letters, etc. The equivalence of strings is the same as the mathematical notion of equivalence, it always respects the reflexivity, symmetry, and transitivity property. The similarity problem on the other hand, is the more difficult problem, where any typing and spelling errors are allowed. The similarity problem then is not necessarily transitive; while it still respects the reflexivity and symmetry properties. Edit distance approaches are typically implemented using dynamic programming and run in $O(mn)$ time where m and n are the lengths of the two records. Thus the importance on avoiding unnecessary calls to the record matching function by the duplicate detection system.

Any of the approximate string matching algorithms can be used in place of the record matching algorithm in the duplicate detection system. In previous work we have used a generalized edit-distance algorithm. This domain-independent algorithm is a variant of the well-known Smith-Waterman algorithm [29], which was originally developed for finding evolutionary relationships between biological protein or DNA sequences.

The Smith-Waterman algorithm is domain-independent under the assumptions that records have similar schemas and that records are made up of alphanumeric characters. The first assumption is needed because the Smith-Waterman algorithm does not address the problem of duplicate records containing fields which are transposed, see [5] for solutions to this problem. The second assumption is needed because any edit-distance algorithm assumes that records are strings over some fixed alphabet of symbols. Naturally this assumption is true for a wide range of databases, including those with numerical fields such as social security numbers that are represented in decimal notation.

3 System to detect duplicate database records

This section summarizes the system used in detecting approximately duplicate database records [3, 4, 5]. In general, we are interested in situations where several records may refer to the same real-world entity, while not being syntactically equivalent. A set of records that refer to the same entity can be interpreted in two ways. One way is to view one of the records as correct and the other records as duplicates containing erroneous information. The task then is to cleanse the database of the duplicate records [6, 7]. Another interpretation is to consider each matching record as a partial source of information. The aim is then to merge the duplicate records, yielding one record with more complete information [8]. The system described here gives a solution to the detection of approximately duplicate records only. In particular, it does not provide a solution to the problem of consolidating the detected duplicate records into a single representation for the real-world entity.

3.1 Standard duplicate detection

The well known and standard method of detecting *exact duplicates* in a table is to sort the table and then to check if consecutive records are identical. Exact duplicates are guaranteed to be next to each other in the sorted order regardless of which part of a record the sort is performed on. The approach can be extended to detect approximate duplicates. The idea is to do sorting to achieve preliminary clustering, and then to do pairwise comparisons of nearby records [9, 10, 11]. In this case, there are no guarantees as to where duplicates are located relative to each other in the sorted order. At best, the approximate duplicate records may not be situated next to each other but will be found nearby. In the worse case they will be found in opposite extremes of the sorted order. Such results are possible due to the choice of field to sort on and also to the errors present in the records. In order to capture all possible duplicate records, every possible pair of records must be compared, leading to a quadratic number of comparisons – where the comparison performed is typically an expensive operation as we will see in section 2. This gives rise to an inefficient and possibly infeasible solution since the number of records in the database may be in the order of hundreds of millions.

To avoid so many comparisons, we can instead compare only records that are within a certain distance from each other. For example, in [7], the authors compare nearby records by sliding a window of fixed size over the sorted database. As a window of size W slides over the database one record at a time, the new record is compared against the other $W - 1$ records in the window. Now, the number of record comparisons decreases from $O(T^2)$ to $O(TW)$ where T is the total number of records in the database.

There is a tradeoff here between the number of comparisons performed and the accuracy of the detection algorithm. The more records the window contains (large value of W) the better the system will do in detecting duplicate records. However this also increases the number of comparisons performed and thus leads to an increase in running time. An effective approach is to scan the records more than once but in a different order and apply the fixed windowing strategy to compare records and combine the results from the different passes[9, 12]. Typically, combining the results of several passes over the database with small window sizes yields better accuracy for the same cost than one pass over the database with a large window size.

One way to combine the results of multiple passes is by explicitly computing the transitive closure of all discovered pairwise "is a duplicate of" relationships [7]. If record R_1 is a duplicate of record R_2, and record R_2 is a duplicate of record R_3, then by transitivity R_1 is a duplicate of record R_3. Transitivity is true by definition if duplicate records concern the same real-world identity, but in practice there will always be errors in computing pairwise "is a duplicate of" relationships, and transitivity will propagate these errors. However, in typical databases, sets of duplicate records tend to be distributed sparsely over the space of possible records, and the propagation of errors is rare. The experimental results confirm this claim [7, 13, 4, 5].

3.2 An adaptive and efficient duplicate detection system

Under the assumption of transitivity, the problem of detecting duplicates in a database can be described in terms of keeping track of the connected components of an undirected graph. Let the vertices of a graph G represent the records in a database of size T. Initially, the graph will contain T unconnected vertices, one for each record in the database. There is an undirected edge between two vertices if and only if the records corresponding to the pair of vertices are found to match according to the pairwise record matching algorithm. At any time, the connected components of the graph G correspond to the transitive closure of the "is a duplicate of" relationships discovered so far. To incrementally maintain the connected components of an undirected graph we use the union-find data structure[14, 15].

The standard algorithm has another weakness in that the window used for scanning the database records is of fixed size. If a cluster in the database has more duplicate records than the size of the window, then it is possible that some of these duplicates will not be detected because not enough comparisons are being made. Furthermore if a cluster has very few duplicates or none at all, then it is possible that comparisons are being done which may not be needed. An approach is needed that responds adaptively to the size and homogeneity of the clusters discovered as the database is scanned, in effect expanding/shrinking the window when necessary. To achieve this, the fixed size window is replaced by a priority queue of duplicate records.

The system scans the sorted database with a priority queue of record subsets belonging to the last few clusters detected. The priority queue contains a fixed number of sets of records. Each set contains one or more records from a detected cluster. For efficiency reasons, entire clusters should not always be saved since they may contain many records. On the other hand, a single record may be insufficient to represent all the variability present in a cluster. Records of a cluster will be saved in the priority queue only if they add to the variability of the cluster being represented. The set representing the cluster with the most recently detected cluster member has highest priority in the queue, and so on.

Suppose that record R_j is the record currently being considered. The algorithm first tests whether R_j is already known to be a member of one of the clusters represented in the priority queue. This test is done by comparing the cluster representative of R_j to the representative of each cluster present in the priority queue. If one of these comparisons is successful, then R_j is already known to be a member of the cluster represented by the set in the priority queue. We move this set to the head of the priority queue and continue with the next record, R_{j+1}. Whatever their result, these comparisons are computationally inexpensive because they are done just with *Find* operations.

Next, in the case where R_j is not a known member of an existing priority queue cluster, the algorithm uses the matching algorithm to compare R_j with records in the priority queue. The algorithm iterates through each set in the priority queue, starting with the highest priority set. For each set, the algorithm scans through the members R_i of the set. R_j is compared to R_i using the matching algorithm. If a match is found, then R_j's cluster is combined with R_i's cluster, using a $Union(R_i, R_j)$ operation. In addition, R_j may also be included in the priority queue set that represents R_i's cluster – and now also represents the new combined cluster. Intuitively, if R_j is very similar to R_i, it is not necessary to include it in the subset representing the cluster, but if R_j is only somewhat similar then including R_j in the subset will help in detecting future members of the cluster.

On the other hand, if the comparison between R_i and R_j yields a very low score then the system continues directly with the next set in the priority queue. The intuition here is that if R_i and R_j have no similarity at all, then comparisons of R_j with other members of the cluster containing R_i will likely also fail. If the comparison still fails but the score is close to the matching threshold, then it is worthwhile to compare R_j with the remaining members of the cluster. These heuristics are used to counter the errors which are propagated when computing pairwise "is a duplicate of" relationships.

Finally, if R_j is compared to members of each set in the priority queue without detecting that it is a duplicate of any of these, then R_j must be a member of a cluster not currently represented in the priority queue. In this case R_j is saved as a singleton set in the priority queue, with the highest priority. If this action causes the size of the priority queue to exceed its limit then the lowest priority set is removed from the priority queue.

Earlier research showed that this adaptive duplicate detection system performs much fewer comparisons than previous work [3, 4, 5]. Fewer comparisons usually translates to decreased accuracy. However, similar accuracy was observed because the comparisons which are not performed correspond to records which are already members of a cluster, most likely due to the transitive closure of the "is a duplicate of" relationships. All experiments show that the improved algorithm is as accurate as the standard method while significantly performing many fewer record comparisons – as much as a 75% savings over the methods by [7, 13].

4 Conclusion

The integration of information sources is an important area of research. There is much to be gained from integrating multiple information sources. However, there are many obstacles that must be overcome to obtain valuable results from this integration.

This article has explored the problem of approximate duplicate detection. To integrate data from multiple sources, one must first identify the information which is common in these sources. Different record matching algorithms were presented that determine the equivalence of records from these sources. Section 2 presents the properties of such record matching algorithms to be applied to alphanumeric records that contain fields such as names, addresses, titles, dates, identification numbers, and so on.

The duplicate detection system described in this work and in [4, 5] improve previous related work in two significant ways. The first contribution is to show how to compute the transitive closure of "is a duplicate of" relationships incrementally, using the union-find data structure. The second contribution is a heuristic method for minimizing the number of expensive pairwise record comparisons that must be performed while comparing individual records with potential duplicates. These two contributions can be combined with any pairwise record matching algorithm. It is desirable for the record matching algorithms to be as domain-independent as possible and also for the algorithms to indicate the strength of the match (or non-match) between the records.

References

[1] A. E. Monge and C. P. Elkan, "WebFind: Automatic retrieval of scientific papers over the world wide web," in *Working notes of the Fall Symposium on AI Applications in Knowledge Navigation and Retrieval*, p. 151, AAAI Press, Nov. 1995.

[2] A. E. Monge and C. P. Elkan, "The WebFind tool for finding scientific papers over the worldwide web," in *Proceedings of the 3rd International Congress on Computer Science Research*, (Tijuana, Baja California, México), pp. 41–46, Nov. 1996.

[3] A. E. Monge and C. P. Elkan, "An efficient domain-independent algorithm for detecting approximately duplicate database records," in *Proceedings of SIGMOD Workshop on Research Issues on Data Mining and Knowledge Discovery*, (Tucson, Arizona), May 1997.

[4] A. Monge, *Adaptive detection of approximately duplicate database records and the database integration approach to information discovery*. Ph.D. thesis, Department of Computer Science and Engineering, University of California, San Diego, 1997. Available from University Microfilms International.

[5] A. E. Monge, "An adaptive and efficient algorithm for detecting approximately duplicate database records," June 2000. Submitted for journal publication.

[6] A. Silberschatz, M. Stonebraker, and J. D. Ullman, "Database research: achievements and opportunities into the 21st century." A report of an NSF workshop on the future of database research, May 1995.

[7] M. Hernández and S. Stolfo, "The merge/purge problem for large databases," in *Proceedings of the ACM SIGMOD International Conference on Management of Data*, pp. 127–138, May 1995.

[8] J. A. Hylton, "Identifying and merging related bibliographic records," M.S. thesis, MIT, 1996. Published as MIT Laboratory for Computer Science Technical Report 678.

[9] H. B. Newcombe, J. M. Kennedy, S. J. Axford, and A. P. James, "Automatic linkage of vital records," *Science*, vol. 130, pp. 954–959, Oct. 1959. Reprinted in [12].

[10] I. P. Fellegi and A. B. Sunter, "A theory for record linkage," *Journal of the American Statistical Association*, vol. 64, pp. 1183–1210, 1969.

[11] C. A. Giles, A. A. Brooks, T. Doszkocs, and D. Hummel, "An experiment in computer-assisted duplicate checking," in *Proceedings of the ASIS Annual Meeting*, p. 108, 1976.

[12] B. Kilss and W. Alvey, eds., *Record linkage techniques, 1985: Proceedings of the Workshop on Exact Matching Methodologies*, (Arlington, Virginia), Internal Revenue Service, Statistics of Income Division, 1985. U.S. Internal Revenue Service, Publication 1299 (2-86).

[13] M. Hernández, *A Generalization of Band Joins and the Merge/Purge Problem*. Ph.D. thesis, Columbia University, 1996.

[14] J. E. Hopcroft and J. D. Ullman, "Set merging algorithms," *SIAM Journal on Computing*, vol. 2, pp. 294–303, Dec. 1973.

[15] T. H. Cormen, C. E. Leiserson, and R. L. Rivest, *Introduction to Algorithms*. MIT Press, 1990.

[16] H. B. Newcombe, *Handbook of record linkage: methods for health and statistical studies, administration, and business*. Oxford University Press, 1988.

[17] M. I. Yampolskii and A. E. Gorbonosov, "Detection of duplicate secondary documents," *Nauchno-Tekhnicheskaya Informatsiya*, vol. 1, no. 8, pp. 3–6, 1973.

[18] U. C. Bureau, ed., *U.S. Census Bureau's 1997 Record Linkage Workshop*, (Arlington, Virginia), Statistical Research Division, U.S. Census Bureau, 1997.

[19] T. E. Senator, H. G. Goldberg, J. Wooton, and M. A. C. *et al.*, "The financial crimes enforcement network AI system (FAIS): identifying potential money laundering from reports of large cash transactions," *AI Magazine*, vol. 16, no. 4, pp. 21–39, 1995.

[20] Y. R. Wang, S. E. Madnick, and D. C. Horton, "Inter-database instance identification in composite information systems," in *Proceedings of the Twenty-Second Annual Hawaii International Conference on System Sciences*, pp. 677–84, Jan. 1989.

[21] R. S. Boyer and J. S. Moore, "A fast string-searching algorithm," *Communications of the ACM*, vol. 20, no. 10, pp. 762–772, 1977.

[22] D. E. Knuth, J. H. M. Jr., and V. R. Pratt, "Fast pattern matching in strings," *SIAM Journal on Computing*, vol. 6, no. 2, pp. 323–350, 1977.

[23] P. A. V. Hall and G. R. Dowling, "Approximate string matching," *ACM Computing Surveys*, vol. 12, no. 4, pp. 381–402, 1980.

[24] Z. Galil and R. Giancarlo, "Data structures and algorithms for approximate string matching," *Journal of Complexity*, vol. 4, pp. 33–72, 1988.

[25] W. I. Chang and J. Lampe, "Theoretical and empirical comparisons of approximate string matching algorithms," in *CPM: 3rd Symposium on Combinatorial Pattern Matching*, pp. 175–84, 1992.

[26] M.-W. Du and S. C. Chang, "Approach to designing very fast approximate string matching algorithms," *IEEE Transactions on Knowledge and Data Engineering*, vol. 6, pp. 620–633, Aug. 1994.

[27] V. Levenshtein, "Binary codes capable of correcting deletions, insertions, and reversals," *Soviet Physics – Doklady 10*, vol. 10, pp. 707–710, 1966.

[28] J. Peterson, "Computer programs for detecting and correcting spelling errors," *Communications of the ACM*, vol. 23, no. 12, pp. 676–687, 1980.

[29] T. F. Smith and M. S. Waterman, "Identification of common molecular subsequences," *Journal of Molecular Biology*, vol. 147, pp. 195–197, 1981.

A Comparison of Techniques to Find Mirrored Hosts on the WWW

Krishna Bharat
Google Inc.
2400 Bayshore Ave
Mountain View, CA 94043
bharat@google.com

Andrei Broder
AltaVista Company
1825 S. Grant St.
San Mateo, CA 94402
andrei.broder@av.com

Jeffrey Dean
Google Inc.
2400 Bayshore Ave
Mountain View, CA 94043
jeff@google.com

Monika R. Henzinger
Google Inc.
2400 Bayshore Ave
Mountain View, CA 94043
monika@google.com

Abstract

We compare several algorithms for identifying mirrored hosts on the World Wide Web. The algorithms operate on the basis of URL strings and linkage data: the type of information about web pages easily available from web proxies and crawlers.

Identification of mirrored hosts can improve web-based information retrieval in several ways: First, by identifying mirrored hosts, search engines can avoid storing and returning duplicate documents. Second, several new information retrieval techniques for the Web make inferences based on the explicit links among hypertext documents – mirroring perturbs their graph model and degrades performance. Third, mirroring information can be used to redirect users to alternate mirror sites to compensate for various failures, and can thus improve the performance of web browsers and proxies.

We evaluated 4 classes of "top-down" algorithms for detecting mirrored host pairs (that is, algorithms that are based on page attributes such as URL, IP address, and hyperlinks between pages, and not on the page content) on a collection of 140 million URLs (on 230,000 hosts) and their associated connectivity information. Our best approach is one which combines 5 algorithms and achieved a precision of 0.57 for a recall of 0.86 considering 100,000 ranked host pairs.

1 Introduction

The WWW differs from information retrieval in classical collections in many respects. Two of the most obvious are the very large size of the Web (the largest search engines have over a billion URLs in their index [2] as of mid 2000) and redundancy in the form of sources with the same or roughly the same information. The fraction of the total WWW collection consisting of duplicates and near-duplicates has been estimated at 30 to 45 percent. (See [6] and [15].)

Bulletin of the IEEE Computer Society Technical Committee on Data Engineering

Duplication has both positive and negative aspects. On one hand the redundancy makes retrieval easier: if a search engine has missed one copy, maybe it has the other; or if one page has become unavailable, maybe a replica can be retrieved. On the other hand, from the point of view of search engines storing duplicate content is a waste of resources and from the user's point of view, getting duplicate answers in response to a query is a nuisance.

The principal reason for duplication on the Web is the systematic replication of content across distinct hosts, a phenomenon known as "mirroring" (These notions are defined more precisely below.) It is estimated that at least 10% of the hosts on the WWW are mirrored [3, 11]. The aim of this paper is to present and evaluate algorithms for detecting mirroring on the WWW in an information retrieval framework.

Our definition of mirroring is as follows:

> Two hosts A and B are mirrors iff for every document on A there is a *highly similar* document on B with the same path, and vice versa.

Highly similar is a subjective measure. We made this notion precise by adopting the resemblance distance described in [6, 5] that experimentally computes a number between 0 and 1 as a measure of similarity. Other edit distance measures can be substituted.

Reliable algorithms for finding host mirrors can improve web-based information access in several ways:

- Knowledge of replication can save search engines crawling and index space overhead.
- Recent web IR techniques analyze the linkage between pages in the web graph to rank document (e.g., PageRank [12], CLEVER [8, 7, 10], and Topic Distillation [4]) The independence assumption between subsections of the web graph is violated by mirroring and can perturb the outcome of such link based computations.
- Knowledge of alternate mirrors can help compensate for "broken links" on the WWW.
- Caches on the web can exploit knowledge of redundancy to increase their effective coverage.

The algorithms described in this paper use only the URLs of pages on the web and in some cases the hyperinkage between them (also called "connectivity"). In particular we do not require web page content, unlike the technique of Cho, Shivakumar and Garcia-Molina [9].

Our input corresponds to a graph in which the nodes are web pages and edges are hyperlinks. We require neither the set of nodes, nor edges to be complete. In particular a given host can be represented partially, with paths that differ from those sampled from its mirrors. Such incompleteness is typical of the graphs computable from web crawls or proxy usage logs.

In this paper we discuss and evaluate four classes of algorithms for finding mirrored hosts on the Web. Each algorithm produced a ranked list of distinct host pairs, each pair representing a possible mirroring relationship. The validity of each mirroring relationship was then judged in an automated way. This was done by fetching from the web a subset of pages from each host and testing to see if the other host had a page with an identical path with highly similar content. The host pair was considered valid iff all of 40 such tests succeeded. We then used the classic IR measures of precision and recall to graph the results. The best approach we found was a combination of 5 algorithms, which on our test data achieved a precision of 0.57 for a recall of 0.86 considering 100,000 result host pairs.

Section 2 presents a brief description of our algorithms. The evaluation methodology and performance of algorithms is summarized in section 3. Section 4 discusses related work. For more details see our full paper [1].

2 Ranking Algorithms

In our experiments the input was a subset of an approximately 179 million URL crawl by AltaVista. We chose a subset of 140.6 million URLs that represented hosts with at least 100 pages in our input, corresponding to 233,035 hosts. The URL strings and hyperlinkage required 11.2 Gigabytes uncompressed.

Only mirroring relations between host pairs drawn from this set were determined.

We used four classes of algorithms to compute a ranked list of host pairs representing potential mirrors. These are summarized below (see full paper for details and rationale [1]:

2.1 IP Address Based (Algorithms *IP3* and *IP4*)

These algorithms list hosts that have identical or highly similar IP addresses. Identical or highly similar IP addresses are indicative of 2 hosts on the same server or subnet - which can imply mirroring. However, when many hosts resolve to the same IP address it is indicative of virtual hosting by an internet service provider, which hosts many distinct web sites.

These considerations led us to two algorithms:

- Algorithm *IP4:* We cluster hosts with the same IP address. With a bias against virtual hosting, we process clusters in the increasing order of siz ande enumerate up to 200 host pairs from each cluster
- Algorithm *IP3:* Same as the above case but we instead cluster based solely on the first three octets of the IP address and list at most 5 pairs per cluster.

2.2 URL String Based (Algorithms *hosts, paths, prefix* and *shingles*)

In this class of algorithms we detect similarity between either the hostnames or the path structure (which usually refects the directory structure) of the hosts being compared. In each algorithm a set of features (or "terms") is selected from each host. Term vector matching [14] is used to compute the likelihood that a pair of hosts are mirrors based on weights of terms in common. Based on the type of term used we get four algorithms, each with a weighting scheme. Only terms occurring in fewer than 100 documents were considered.

These are summarized below:

- Hostname Matching (Algorithm *hosts*): Substrings of the hostname by breaking only at periods ('.') are terms. The weight of term t is $\frac{\log(len(t))}{1+\log(df(t))}$ where $df(t)$ is the number of documents with the term, and $len(t)$ is the number of '.' separated segments.
- Full Path Matching (Algorithm *paths*): Entire paths of URLs are used as terms. The weight of term t is $1 + \log(\frac{maxdf}{df(t)})$, where $maxdf$ is the maximum value of $df(t)$ over all t.
- Prefix Matching (Algorithm *prefix*): Prefixes of the path that either end in a '/' or terminate the path are terms. The weighing is as in *path*, except, to remedy the bias against small hosts we multiply by $\frac{1}{0.1+0.15(\log(n_1)+log(n_2))}$, where n_1 and n_2 represent the number of URL strings in the input from each of the two hosts.
- Positional Word Bigram Matching (Algorithm *shingles*): The path is broken into a list of tokens by treating '/' and '.' as breaks. Pairs of tokens in sequence with ordinal position are used as terms. Non-alphabetical character sequences are stemmed to '*'. E.g., the path `conferences/sigweb/web2001/xyz` will have `<2:sigweb 3:web*>` as a term. The same weighting as in *prefix* is used.

2.3 URL String and Connectivity Based (Algorithm *conn*)

The identification of mirrors can be aided by the use of connectivity information. We extend the URL string based *paths* algorithm (which selects host pairs with many common paths), with a connectivity based filtering stage.

Given a host pair $< host_1, host_2 >$ from the output of *paths* we test if 0.9 of the union of their outlinks are common to both. An approximate measure is used because some outlinks may be missing in the input. Paths that qualify are said to be "equivalent". After testing 20 common paths, if 0.75 of the paths are found to be equivalent then the host pair is allowed to remain in the ranking. Otherwise, it is eliminated.

2.4 Host Connectivity-Based (Algorithms *hconn1* and *hconn2*)

In this class of algorithms similarity between hosts is computed based on the similarity between the sets of other hosts they connect to.

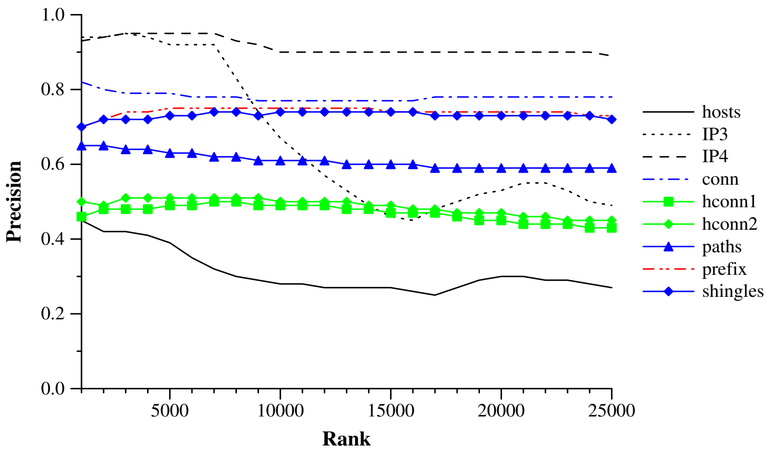

Figure 1: Precision at $rank$ for the algorithms

Again, term vector matching is used to compute host similarity. The terms in this case are hosts. Specifically, host t is a term in host x's vector if there exists at least one hyperlink from a page on x to a page on t. We denote the total number of such links by $f(t, x)$. For efficiency we keep only $10 \log P(x)$ terms with the highest value of $f(t, x)$, where $P(x)$ is the number of URLs sampled from the host x. The weight of each term however is a function of $in(t)$, which is computed by summing $f(t, y)$ for all hosts y that contain t in their term vector.

Algorithm *hconn1* weights term t proportional to $1/in(t)$, varying in the range $[\frac{1}{5} \cdots 1]$. Algorithm *hconn2* further multiplies by a factor $1 + \log(\frac{maxin}{in(t)})$ where $maxin$ is the maximum value of the in function over all terms.

3 Evaluation

3.1 Methodology

As mentioned before each algorithm outputs a ranked set of host pairs. We evaluate whether a given host pair is indeed a mirror using an automated technique, and we call this a *correct* host pair. This allows us to use the standard information retrieval notion of precision and recall to compare algorithms.

- We define *precision at rank* k to be the fraction of correct host pairs given within the first k host pairs.
- We define *recall at rank* k to be the number of correct host pairs found within the first k host pairs divided by R, where R is the total number of distinct correct host pairs.

We computed these numbers as follows: Since some of our algorithms returned several million host pairs, human evaluation for mirroring was impossible. Hence to decide if a host pair was "correct" we sampled 20 random paths from each host and tested for the presence of a highly similar document on the other host. If all 40 tests succeeded the host pair was considered correct.

We evaluated the up to 30,000 host pairs from each ranking in this manner. Figure 1 graphs precision vs rank up to rank 25,000. The best algorithms were *IP4*, *prefix*, *shingles* and *conn*. We found that *IP3* rapidly deteriorated in precision, and *hosts* was too naive. Connectivity analysis in *conn* improved *paths* significantly, but did not help as much with host connectivity (*hconn1* and *hconn2*). *Hconn1* and *hconn2* tended to pairs hosts on the same topic rather than mirrors.

The total pool of mirrors consisted of 41,388 host pairs at rank 25,000. The best single algorithm *IP4* was able to find only 53.9% of the known mirrors. Further, looking at the overlap between the pairs found indicated that algorithms in different classes were finding complementary sets of mirrors. Hence, we introduced a new algorithm *combined* which merged the top 100,000 results from the output of five different algorithms *IP4*, *prefix*, *hconn2*, *paths* and *hostnames*. The host pairs in the resulting union were sorted in the decreasing order of

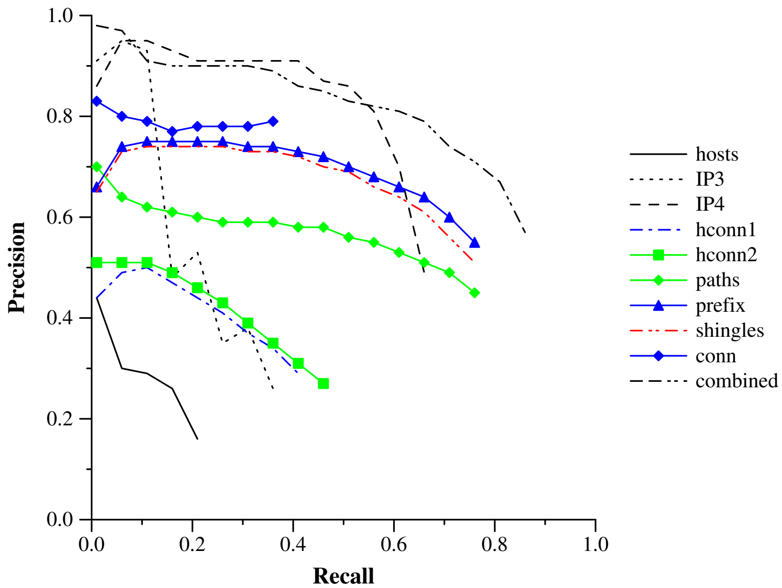

Figure 2: Precision vs Recall

the number of algorithms that found them. Ties between host pairs were resolved in favor of the host pair with the lowest rank in any of the five algorithms.

Figure 2 shows a precision vs recall plot. For this graph we considered the top 100,000 results from each ranking. Host pairs not seen previously were considered correct if and only if they appeared in the transitive closure of the set of known correct host pairs - which we took to be the universe of correct host pairs.

We found that *IP4*'s precision drops after a recall of 0.5. With increasing cluster size IP address matching becomes progressively less reliable. *Prefix* is the best of the individual algorithms, achieving a precision of 0.49 for a recall of 0.8 (*shingles* performs similarly). *Conn* seems potentially better but since we were unable to run the connectivity filtering on the output of *paths* beyond rank 30,000, it was incompletely evaluated. The combined algorithm *combined* had the best performance overall. In all there were 66,337 distinct correct host pairs in our set, and *combined* found 57,000 of them at rank 100,000 (precision=0.57, recall=0.86).

4 Related Work

The problem of computing mirrored host pairs on the WWW was introduced in [3]. They use a more generalized definition of mirroring, which includes *partial mirroring* as well. They tested a single algorithm corresponding to a weighted combination of our *hosts* and *shingles* algorithms, but did not investigate the use of IP addresses or connectivity in finding mirrors. Since we were interested in a comparative study of algorithms we extended their validation scheme to include the pooling of validated results from various algorithms, and also the use of transitive inferences to expedite validation.

At first sight the problem of finding mirrors seems like it could be cast as a clustering problem in which each host is represented by the contents of the documents it contains. There has been considerable prior work in document clustering (see e.g. [16, 13]). As mentioned earlier, owing to the scale of the Web comparing the full contents of hosts is neither feasible, nor (as we believe) necessarily more useful than comparing URL strings and connectivity. See also [17] for a discussion on the applicability of clustering algorithms to the WWW.

Connectivity information has been applied to the problem of improving precision of WWW search results [12, 8, 7, 10, 4]. The graph model used by these methods is perturbed by duplicate pages. We believe our *hconn** and *conn* algorithms represent the first attempt to use connectivity information to finding mirrors. Furthermore, these two algorithms can be easily integrated within the methods cited above to alleviate the above perturbation problem.

The "bottom up" approach taken by Cho et al in [9] requires content analysis and is more compute intensive. Also, it requires all URLs to have been crawled within a small window of time. Our techniques can operate solely with URL structure, which tends to be more stable. The two approaches have yet to be compared.

5 Conclusion

Finding mirrors on the WWW is an information retrieval problem of interest to the search engine and web caching communities. In this paper we evaluated 4 classes of algorithms for ranking potential mirrored host pairs, namely: (i) IP address based approaches, (ii) URL string comparison techniques, (iii) host connectivity based approaches, and (iv) an approach that compares connectivity of documents with shared paths. These were evaluated on a collection of 140 million URLs (on 230,000 hosts) and associated connectivity information. *IP4* and *prefix* were our best single algorithms. We concluded that single approaches are limited in terms of recall. Our best approach is one which combines 5 algorithms and achieves a precision of 0.57 for a recall of 0.86 considering the top 100,000 results.

References

[1] K. Bharat, A. Broder, J. Dean, M. Henzinger. A Comparison of Tehcniques to Find Mirrored Hosts on the WWW. *Journal of the American Society for Information Science (JASIS)* (To Appear, Nov 2000.)

[2] Google Inc. Google Launches World's Largest Search Engine Press Release, June 26, 2000. URL: http://www.google.com/pressrel/pressrelease26.html

[3] K. Bharat and A. Z. Broder. Mirror, mirror on the web: A study of host pairs with replicated content. In *Proceedings of the Eighth International World Wide Web Conference*, May 1999.

[4] K. Bharat and M. Henzinger. Improved algorithms for topic distillation in hyperlinked environments. In *Proceedings of the 21st International ACM SIGIR Conference on Research and Development in Information Retrieval (SIGIR'98)*, pages 111–104, 1998.

[5] A. Z. Broder. Filtering near-duplicate documents. In *Proceedings of FUN 98*, 1998.

[6] A. Z. Broder, S. C. Glassman, M. S. Manasse, and G. Zweig. Syntactic clustering of the Web. In *Sixth International World Wide Web Conference*, pages 391–404, Santa Clara, California, April 1997.

[7] S. Chakrabarti, B. Dom, D. Gibson, S. Kumar, P. Raghavan, S. Rajagopalan, and A. Tomkins. Experiments in topic distillation. In *SIGIR'98 Workshop on Hypertext Information Retrieval for the Web*, 1998.

[8] S. Chakrabarti, B. Dom, R. P., S. Rajagopalan, D. Gibson, and J. Kleinberg. Automatic resource compilation by analyzing hyperlink structure and associated text. In *Proceedings of the Seventh International World Wide Web Conference*, pages 65–74.

[9] J. Cho, N. Shivakumar, and H. Garcia-Molina. Finding Replicated Web Collections. In *ACM SIGMOD 2000*, pages 355–366.

[10] J. Kleinberg. Authoritative sources in a hyperlinked environment. In *Proceedings of the 9th Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 668–677, January 1998.

[11] E. T. O'Neill, P. D. McClain, and B. F. Lavoie. A methodology for sampling the world wide web. Technical report, OCLC Annual Review of Research, 1997.

[12] S. Brin and L. Page. The Anatomy of a Large-Scale Hypertextual Web Search Engine In *Proceedings of the Seventh International World Wide Web Conference*, pages 107-117, 1998.

[13] E. Rasmussen. Clustering algorithms. In W. Frakes and R. Baeza-Yates, editors, *Information Retrieval*, pages 419–42, 1992.

[14] G. Salton and C. Buckley. Term-weighting approaches in automatic text retrieval. *Information Processing and Management*, 24(5):513–523, 1988.

[15] N. Shivakumar and H. García-Molina. Finding near-replicas of documents on the web. In *Proceedings of Workshop on Web Databases (WebDB'98)*, March 1998.

[16] P. Willet. Recent trends in hierarchical document clustering: a critical review. *Information Processing and Management*, 24(5):577–597, 1988.

[17] O. Zamir and O. Etzioni. Web document clustering: A feasibility demonstration. In *Proceedings of the 21st International ACM SIGIR Conference on Research and Development in Information Retrieval (SIGIR'98)*, pages 46–53, 1998.

Automatically Extracting Structure from Free Text Addresses

Vinayak R. Borkar Kaustubh Deshmukh Sunita Sarawagi
Indian Institute of Technology, Bombay
`sunita@it.iitb.ernet.in`

Abstract

In this paper we present a novel way to automatically elementize postal addresses seen as a plain text string into atomic structured elements like "City" and "Street name". This is an essential step in all warehouse data cleaning activities. In spite of the practical importance of the problem and the technical challenges it offers, research effort on the topic has been limited. Existing commercial approaches are based on hand-tuned, rule-based approaches that are brittle and require extensive manual effort when moved to a different postal system. We present a Hidden Markov Model based approach that can work with just about any address domain when seeded with a small training data set. Experiments on real-life datasets yield accuracy of 89% on a heterogeneous nationwide database of Indian postal addresses and 99.6% on US addresses that tend to be more templatized.

1 Introduction

Address elementization [7] is the process of extracting structured elements like "Street name", "Company" and "City name" from an address record occurring as a free text string. For example consider the following address. `18100 New Hamshire Ave. Silver Spring, MD 20861`. This address can be elementized as: `House Number : 18100, Street : New Hamshire Ave., City : Silver Spring, State : MD, Zip : 20861.`

Address Elementatization is one of the key steps in the warehouse data cleaning process. Large customer-oriented organizations like banks, telephone companies and universities store millions of addresses. In the original form, these addresses have little explicit structure. Often for the same person, there are different address records stored in different databases. During warehouse construction, it is necessary to put all these addresses in a standard canonical format where all the different fields are identified and duplicates removed. An address record broken into its structured fields not only enables better querying, it also provides a more robust way of doing deduplication and householding — a process that identifies all addresses belonging to the same household. Previous approaches of deduplication without elementization relied on hand tuned rules to define similarity treating the address as a text string [6, 9].

Existing commercial approaches [5] rely on hand-coded rule-based methods coupled with a database of cities, states and zip codes. This solution is not practical and general because postal addresses in different parts of the world have drastically different structures. In some countries zip codes are five digit numbers whereas in others they are allowed to have strings. The problem is more challenging in older countries like India because most street names do not follow a uniform building numbering scheme, the reliance on ad hoc descriptive

Bulletin of the IEEE Computer Society Technical Committee on Data Engineering

landmarks is common, city names keep changing, state abbreviations are not standardized, spelling mistakes are rampant and zip codes optional. Further each region has evolved its own style of writing addresses that differs significantly from those of the other region. Consider for instance the following two valid addresses from two different Indian cities:

`7D-Brijdham 16-B Bangur Nagar Goregaon (West) Bombay 400 090`

`13 Shopping Centre Kota (Raj) 324 007`

The first address consists of seven elements: house number: `'7D'`, building name: `'Brijdham'`, building number: `'16-B'`, colony name: `'Bangur Nagar'`, area: `'Goregaon (West)'`, city: `'Bombay'` and zip code: `'400 090'`. The second address consists of the following five elements: house number: `'13'`, Colony name: `Shopping centre`, city: `'Kota'`, State: `(Raj)` and zip code: `'324 007'`. In the first address, 'East' was enclosed in parentheses and depicted direction while in the second the string 'Raj' within parentheses is the name of a geographical State. This element is missing in the first address. Whereas, in the second address building name, colony name and area elements are missing.

We propose an automated method for elementizing addresses based on Hidden Markov Models [12] (HMM). Hidden Markov Modeling is a powerful statistical machine learning technique that can handle new data robustly, is computationally efficient and easy for humans to interpret and tweak. The last property makes this technique particularly suitable for the address tagging problem.

An HMM combines information about multiple different aspects of the record in segmenting it. One source is the characteristic words in each element, for example the word "street" appears in road-names. A second source is the limited partial ordering between its elements. Often the first element is a house number, then a possible building name and so on and the last few elements are zip code and state-name. A third source is the typical number of words in each element. For example, state names usually have one or two words whereas road names are longer. Finally, the HMM simultaneously extracts each element from the address to optimize some global objective function. This is in contrast to existing rule learners used in traditional information tasks [4, 10, 8, 1, 11] that treat each element in isolation.

2 Our Approach

The input to the address elementization problem is a fixed set of E tags of the form "House number", "Road name" and "City" and a collection of T example addresses that have been segmented into one or more of these tags. We do not assume any fixed ordering between the elements nor are all elements required to be present in all addresses. The output of the training phase is a model that when presented with a address record segments it into one or more of its constituent elements.

2.1 Hidden Markov Models

A Hidden Markov Model(HMM) is a probabilistic finite state automaton [12, 13] where each state probabilistically outputs a symbol from a dictionary. Each state transition in the automaton is associated with a probability value. This probability is independent of the output generated by the current state or the next state, and also of the time at which this state is reached. Mathematically a HMM is completely described by two parameters, m and n and two probability distributions A and B. The description of these parameters is as follows.

- n, the number of states in the model.
- m, the number of distinct observation symbols, the discrete dictionary size.
- The state transition probability distribution $A = a_{ij}$ where a_{ij} is the probability of transiting from state i to state j.
- The observation symbol probability distribution, $B = b_j(k)$, where $b_j(k)$is the probability of emitting the k^{th} dictionary symbol in state j

We have shown a typical Hidden Markov Model used for Address Elementization in Figure 1. In this figure, the value of n is 10. The edge labels depict the state transition probabilities (A Matrix).

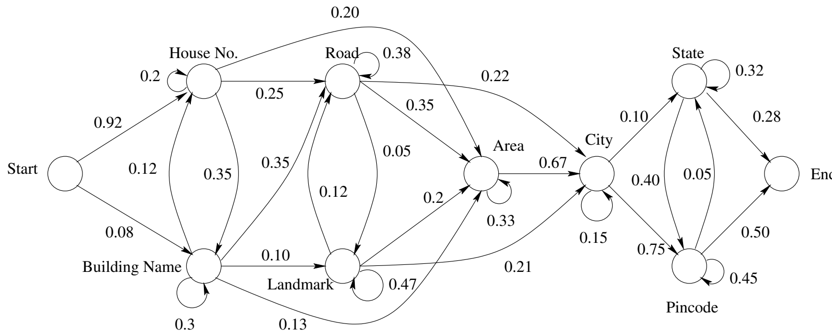

Figure 1: Structure of a typical naive HMM

2.2 HMMs for Address Elementization

This basic HMM model needs to be augmented for segmenting free text into the constituent elements. Let E be the total number of elements. Each state of the HMM is marked with exactly one of these E elements, although more than one state could be marked with the same element. The training data consists of a sequence of element-symbol pairs — for each pair $\langle e, s\rangle$ the symbol s can only be emitted from a state marked with element e.

The training process consists of two main tasks. The first task is choosing the structure of the HMM, that is, the number of states in the HMM and edges amongst states. In general, it is difficult to get the optimal number of states in the HMM. We will describe our approach in Section 2.3. The second task is learning the transition and output probabilities of the dictionary symbols. The dictionary consists of all words, digits and delimiters appearing the training data. In [2] we present further refinements on the dictionary where we introduce the concept of a taxonomy on the symbols (words, numbers, delimiters) appearing in the training sequence and show how to generalize the dictionary of each HMM state to a level that provides highest accuracy.

Training the HMM The goal during training is finding the A and B matrices such that the probability of the HMM generating these training sequences is maximized. Each training sequence makes transitions from the start state to the end state through a series of intermediate states. When the transitions made by each sequence is known, transition probabilities can be calculated using a *Maximum Likelihood* approach. The probability of transition from state i to state j is computed as,

$$a_{ij} = \frac{\text{Number of transitions from state } i \text{ to state } j}{\text{Total number of transitions out of state } i} \tag{1}$$

The emission probabilities are computed similarly. The probability of emitting symbol k in state j is computed as,

$$b_{jk} = \frac{\text{Number of times the } k\text{-th symbol was emitted at state } j}{\text{Total number of symbols emitted at state } j} \tag{2}$$

Using the HMM for testing Given an output symbol sequence $O = o_1, o_2, \ldots, o_k$, we want to associate each symbol with an element. Since each state in the HMM is associated with exactly one element, we associate each symbol with the state that emitted the symbol. Hence we need to find a path of length k from the start state to the end state, such that the i^{th} symbol o_i is emitted by the i^{th} state in the path. In general, an output sequence can be generated through multiple paths each having some probability. We assume the *Viterbi approximation* and say that the path having the highest probability is the path which generated the given output sequence. Given n states and a sequence of length k, there can be $O(k^n)$ possible paths that can generate the given sequence. This exponential complexity in finding the most probable path is cut down to $O(kn^2)$ by the famous dynamic programming-based *Viterbi Algorithm* [13].

2.3 Learning the structure of the HMM

We impose a nested hierarchical model on the structure of the HMM. An outer HMM captures the sequencing relationship amongst elements treating each element as a single state. Each element's state is expanded to an inner HMM that captures its internal structure.

Accordingly, training of the HMM is done in two stages. In the first stage we learn the outer HMM. In this stage, the training data is treated as a sequence of elements ignoring all details of the length of each element and the tokens it contains. These sequences are used to train the outer HMM. In the second stage, the inner HMMs are learnt using the procedure described next.

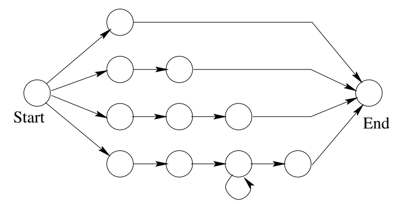

Figure 2: A four length Parallel Path structure

In general, the number of tokens in each element is variable. We handle such variability automatically by choosing a parallel path structure (Figure 2) for each nested HMMs. In the Figure, the start and end states are dummy nodes to mark the two end points of a tag. They do not output any token. All records of length one will pass through the first path, length two will go through the second path and so on. The last path captures all records with four or more tokens.

We next describe how the structure of the HMM in terms of the number of paths and the position of the state with the self loop is chosen from the training data. Initially, we create as many paths as the number of distinct token counts. This might create some paths that have few training examples leading to a poorly trained dictionary. We merge such paths to its neighboring path as follows. Suppose a k state path needs to be merged with a $k-1$ state path to produce a single merged path. In the k-path, $k-1$ of its state will be merged with those of the $(k-1)$-path. We need to decide which of the k states will not be merged. We try out all possible k choices of this state and choose the one that gives the gives best result on a separate validation dataset. Such merging of paths is continued until a merge does not cause any improvement in accuracy.

3 Results

We measured the efficacy of the proposed techniques on real-life address datasets. We consider three different address sources:

US addresses: A set of 740 US addresses downloaded from an internet directory partitioned into 6 elements as shown in Table 4.

Student address: 2388 home addresses of students in the author's university campus partitioned into 16 elements as described in Figure 6.

Company address: 769 addresses of customers of a major national bank in a large Asian metropolitan broken into 6 elements as shown in Table 5.

In each case one-third of the data was used for training and the remaining two-thirds used for testing. We obtained accuracies of 99.6%,89.3% and 83% on the US, student and company dataset respectively. The non-US addresses have a much higher complexity compared to the US addresses. In Table 7 we show the accuracy partitioned into precision and recall values for individual elements. The table shows that there is a wide variance on the precision of each tag. Fortunately, the tags on which accuracy is low also happen to be the less important tags and occur infrequently in both the training and test instances.

We compared the performance of our approach with a rule learner, Rapier [3]. Rapier is a bottom-up induc-

Element	Tokens present	Precision (%)	Recall (%)
House No.	427	99.3	99.765
Po Box	10	57.142	40.0
Road Name	1268	99.449	99.763
City	611	100.0	99.509
State	490	100.0	100.0
Zip code	490	100.0	100.0
Overall	3296	99.605	99.605

Table 4: US addresses

Element	Tokens present	Precision (%)	Recall (%)
Care Of	278	90.604	48.561
House Address	2145	77.343	88.484
Road Name	1646	78.153	73.025
Area	808	91.7	83.415
City	527	99.81	100.0
Zip Code	519	100.0	100.0
Overall	5923	83.656	83.656

Table 5: Company addresses

Element	Tokens present	Precision (%)	Recall (%)
Care Of	281	98.091	91.459
Department	99	100.0	20.202
Designation	98	45.098	23.469
House No.	3306	95.681	90.471
Bldg. Name	1702	73.204	77.849
Society	3012	76.554	85.856
Road Name	2454	84.815	88.997
Landmark	443	88.338	68.397
Area	2364	93.277	89.805
P O	231	86.473	77.489
City	1785	96.561	97.535
Village	40	100.0	13.333
District	138	93.333	81.159
State	231	96.38	92.207
Country	20	100.0	45.0
Zip code	3042	99.967	99.934
Overall	19246	88.901	88.901

Table 6: Student addresses

Table 7: Precision and recall values for different datasets shown broken down into the constituent elements.

tive learning system for finding information extraction rules. It has been tested on several domains and found to be competitive. It extracts each tag in isolation of the rest. The accuracy of Rapier was found to be considerably lower than our approach. Rapier leaves many tokens untagged by not assigning them to any of the elements. Thus it has low recall — 98%, 50%, 60% on the US, Student and Company datasets respectively. However, the precision of Rapier was found to be competitive to our method. The overall accuracy is acceptable only for US addresses where the address format is regular enough to be amenable to rule-based processing. For the complicated sixteen-element Student dataset such rule-based processing could not successfully tag all elements.

The size of the training data is an important concern in all extraction tasks that require manual effort in tagging the instances. In most such information extraction problems, untagged data is plentiful but tagged data to serve as training records is scarce and requires human effort. We therefore study the amount of training effort needed to achieve peak performance in our approach. The results show that HMMs are fast learners. For US addresses (Figure 3), just 50 addresses achieved the peak accuracy of 99.5% on 690 test instances and just 10 addresses yielded an accuracy of 91%. For the Student dataset (Figure 4) with 150 training addresses we get 85% accuracy on 2238 addresses and with 300 addresses reach 89% accuracy on the remaining 2088 addresses. Further increasing the training size only slightly boosts the accuracy. A similar trend was observed for the Company dataset.

4 Conclusion

Address elementization is a key step of the warehouse data cleaning process and consequently of great practical importance. Surprisingly, research effort on the topic has been limited in spite of the many technical challenges that arise because of the multitudes of ways in which addresses are written. Existing commercial approaches rely on hand-coded rule-based systems that are hard to set up and evolve. In this paper we presented an automated approach for elementizing postal addresses using the powerful technique of Hidden Markov Modeling. We developed practical methods of choosing the best structure of the HMM and ways of incorporating a partial

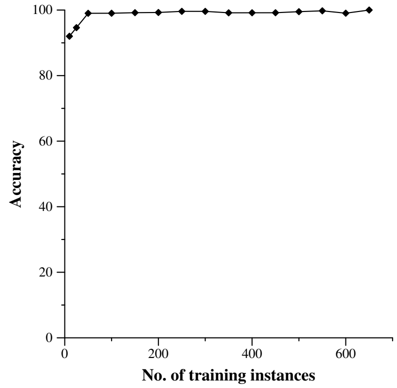

Figure 3: US addresses

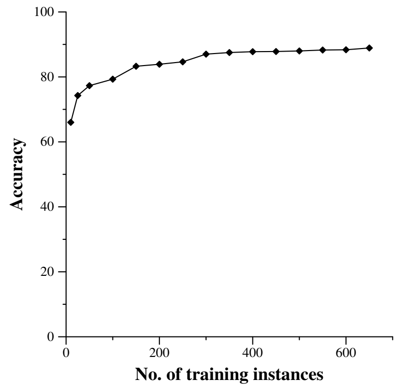

Figure 4: Student Addresses

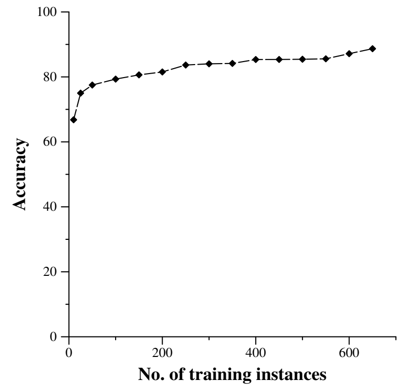

Figure 5: Company Addresses

Figure 6: Effect of training data size on accuracy for different datasets

hierarchical database into the HMM. Experiments on real-life address datasets yield very encouraging results. Given the encouraging experimental results and the intuitive, human-interpretable nature of the model we believe that the HMM approach is an excellent choice for the address elementization problem.

We are further refining our approach by adding support for automated feature selection, incorporating a partial database and reducing amount of training data needed by active learning. Other future work on the topic include correcting and allowing for spelling mistakes in the data and automatically segmenting a large heterogeneous training dataset into subsets to be trained on separate HMMs.

References

[1] Brad Aldelberg. Nodose: A tool for semi-automatically extracting structured and semistructured data from text documents. In *SIGMOD*, 1998.

[2] Vinayak R. Borkar, Kaustubh Deshmukh, and Sunita Sarawagi. Automatic text segmentation for extracting structured records. Submitted for publication, 2000.

[3] Mary Elaine Califf and Raymond J. Mooney. Relational learning of pattern-match rules for information extraction. In *Proceedings of the Sixteenth National Conference on Artificial Intelligence (AAAI-99)*, pages 328–334, July 1999.

[4] Arturo Crespo, Jan Jannink, Erich Neuhold, Michael Rys, and Rudi Studer. A survey of semi-automatic extraction and transformation. http://www-db.stanford.edu/ crespo/publications/.

[5] Helena Galhardas. *http://caravel.inria.fr/ galharda/cleaning.html*.

[6] Mauricio A. Hernandez and Salvatore J. Stolfo. The merge/purge problem for large databases. In *Proceedings of the ACM SIGMOD*, 1995.

[7] Ralph Kimball. Dealing with dirty data. *Intelligent Enterprise*, September 1996. http://www.intelligententerprise.com/.

[8] N. Kushmerick, D.S. Weld, and R. Doorenbos. Wrapper induction for information extraction. In *Proceedings of IJCAI*, 1997.

[9] Alvaro E. Monge and Charles P. Elkan. The field matching problem: Algorithms and applications. In *Proceedings of the Second International Conference on Knowledge Discovery and Data Mining (KDD-96)*, 1996.

[10] Ion Muslea. Extraction patterns for information extraction tasks: A survey. In *The AAAI-99 Workshop on Machine Learning for Information Extraction*, 1999.

[11] Ion Muslea, Steve Minton, and Craig A. Knoblock. A hierarchical approach to wrapper induction. In *Proceedings of the Third International Conference on Autonomous Agents*, Seattle, WA, 1999.

[12] Lawrence Rabiner. A tutorial on Hidden Markov Models and selected applications in speech recognition. In *Proceedings of the IEEE, 77(2)*, 1989.

[13] Lawrence Rabiner and Biing-Hwang Juang. *Fundamentals of Speech Recognition*, chapter 6. Prentice-Hall, 1993.

Accurately and Reliably Extracting Data from the Web: A Machine Learning Approach

Craig A. Knoblock
University of Southern California
and Fetch Technologies

Kristina Lerman
University of Southern California

Steven Minton
Fetch Technologies

Ion Muslea
University of Southern California

Abstract

A critical problem in developing information agents for the Web is accessing data that is formatted for human use. We have developed a set of tools for extracting data from web sites and transforming it into a structured data format, such as XML. The resulting data can then be used to build new applications without having to deal with unstructured data. The advantages of our wrapping technology over previous work are the the ability to learn highly accurate extraction rules, to verify the wrapper to ensure that the correct data continues to be extracted, and to automatically adapt to changes in the sites from which the data is being extracted.

1 Introduction

There is a tremendous amount of information available on the Web, but much of this information is not in a form that can be easily used by other applications. There are hopes that XML will solve this problem, but XML is not yet in widespread use and even in the best case it will only address the problem within application domains where the interested parties can agree on the XML schema definitions. Previous work on wrapper generation in both academic research [4, 6, 8] and commercial products (such as OnDisplay's eContent) have primarily focused on the ability to rapidly create wrappers. The previous work makes no attempt to ensure the accuracy of the wrappers over the entire set of pages of a site and provides no capability to detect failures and repair the wrappers when the underlying sources change.

We have developed the technology for rapidly building wrappers for accurately and reliably extracting data from semistructured sources. Figure 1 graphically illustrates the entire lifecycle of a wrapper. As shown in the Figure, the wrapper induction system takes a set of web pages labeled with examples of the data to be extracted. The user provides the initial set of labeled examples and the system can suggest additional pages for the user to label in order to build wrappers that are very accurate. The output of the wrapper induction system is a set of extraction rules that describe how to locate the desired information on a Web page. After the system creates a wrapper, the wrapper verification system uses the functioning wrapper to learn patterns that describe the data being extracted. If a change is detected, the system can automatically repair a wrapper by using the same patterns

Bulletin of the IEEE Computer Society Technical Committee on Data Engineering

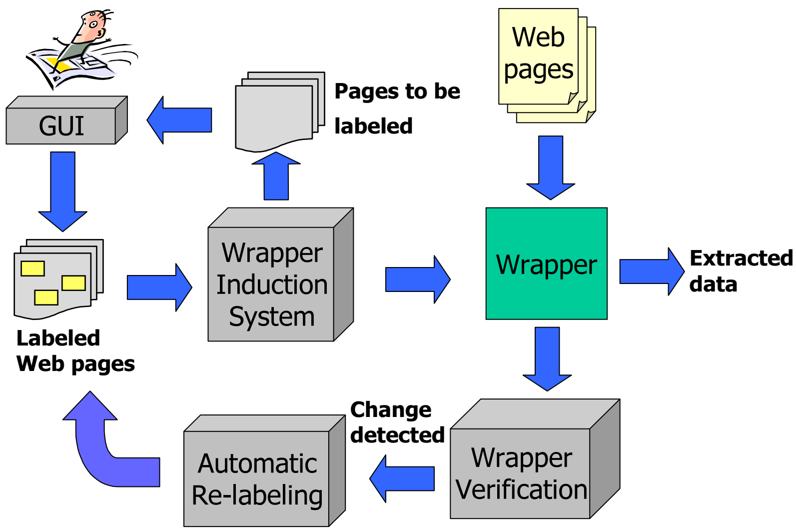

Figure 1: **The Lifecycle of a Wrapper**

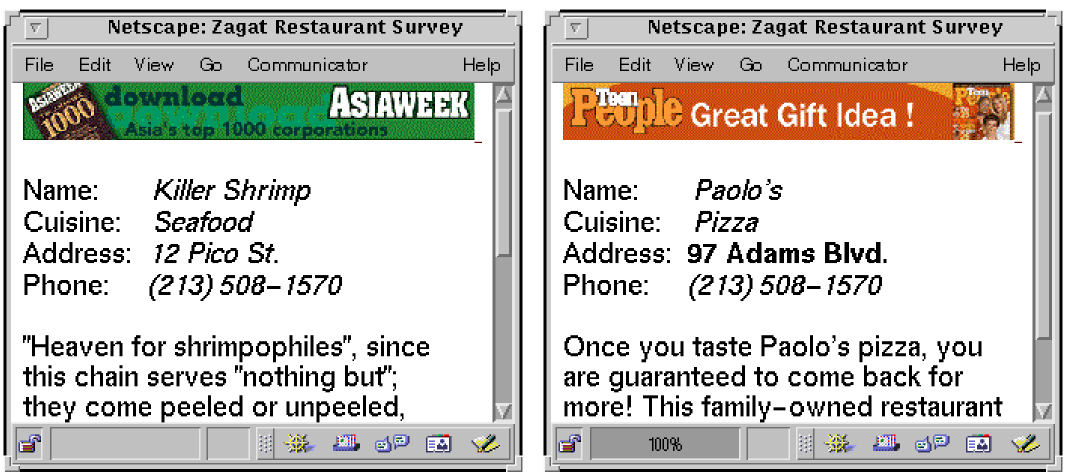

Figure 2: **Two Sample Restaurant Documents From the Zagat Guide.**

to locate examples on the changed pages and re-running the wrapper induction system. The details of this entire process are described in the remainder of this paper.

2 Learning Extraction Rules

A wrapper is a piece of software that enables a semistructured Web source to be queried as if it were a database. These are sources where there is no explicit structure or schema, but there is an implicit underlying structure (for example, consider the two documents in Figure 2). Even text sources, such as email messages, have some structure in the heading that can be exploited to extract the date, sender, addressee, title, and body of the messages. Other sources, such as online catalogs, have a very regular structure that can be exploited to extract the data automatically.

One of the critical problems in building a wrapper is defining a set of extraction rules that precisely define how to locate the information on the page. For any given item to be extracted from a page, one needs an extraction rule to locate both the beginning and end of that item. Since, in our framework, each document consists of a sequence of tokens (e.g., words, numbers, HTML tags, etc), this is equivalent to finding the first and last tokens of an item. The hard part of this problem is constructing a set of extraction rules that work for *all* of the pages in the source.

A key idea underlying our work is that the extraction rules are based on "landmarks" (i.e., groups of consecutive tokens) that enable a wrapper to locate the start and end of the item within the page. For example, let us consider the three restaurant descriptions `E1`, `E2`, and `E3` presented in Figure 3. In order to identify the beginning of the address, we can use the rule

```
E1:   ...Cuisine:<i>Seafood</i><p>Address:<i>  12 Pico St. </i><p>Phone:<i>...
E2:   ...Cuisine:<i>Thai   </i><p>Address:<i> 512 Oak Blvd.</i><p>Phone:<i>...
E3:   ...Cuisine:<i>Burgers</i><p>Address:<i> 416 Main St. </i><p>Phone:<i>...
E4:   ...Cuisine:<i>Pizza</i><p>Address:<b> 97 Adams Blvd. </b><p>Phone:<i>...
```

Figure 3: **Four sample restaurant documents.**

R1 = $SkipTo$(`Address`) $SkipTo$(`<i>`)

which has the following meaning: start from the beginning of the document and skip every token until you find a landmark consisting of the word `Address`, and then, again, ignore everything until you find the landmark `<i>`. **R1** is called a *start rule* because it identifies the beginning of the address. One can write a similar *end rule* that finds the end of the address; for sake of simplicity, we restrict our discussion here to start rules.

Note that **R1** is by no means the only way to identfy the beginning of the address. For instance, the rules

R2 = $SkipTo$(`Address : <i>`)
R3 = $SkipTo$(`Cuisine : <i>`) $SkipTo$(`Address : <i>`)
R4 = $SkipTo$(`Cuisine : <i>` *_Capitalized_* `</i> <p> Address : <i>`)

can be also used as start rules. **R2** uses the 3-token landmark that immediately precedes the beginning of the address in examples `E1`, `E2`, and `E3`, while **R3** relies on two 3-token landmarks. Finally, **R4** is defined based on a 9-token landmark that uses the wildcard *_Capitalized_*, which is a placeholder for any capitalized alphabetic string (other examples of useful wildcards are *_Number_*, *_AllCaps_*, *_HtmlTag_*, etc).

To deal with variations in the format of the documents, our extraction rules allow the use of *disjunctions*. For example, let us assume that the addresses that are within one mile from your location appear in bold (see example `E4` in Figure 3), while the other ones are displayed as italic (e.g., `E1`, `E2`, and `E3`). We can extract all the names based on the disjunctive start rule

either $SkipTo$(`Address : <b>`)
or $SkipTo$(`Address`) $SkipTo$(`<i>`)

Disjunctive rules are *ordered lists* of individual disjuncts (i.e., decision lists). Applying a disjunctive rule is a straightforward process: the wrapper successively applies each disjunct in the list until it finds the first one that matches. Even though in our simplified examples one could have used the nondisjunctive rule

$SkipTo$(`Address :` *_HtmlTag_*),

there are many real world sources that cannot be wrapped without using disjuncts.

We have developed STALKER [11], a *hierarchical* wrapper induction algorithm that learns extraction rules based on examples labeled by the user. We have a graphical user interface that allows a user to mark up several pages on a site, and the system then generates a set of extraction rules that accurately extract the required information. Our approach uses a greedy-covering inductive learning algorithm, which incrementally builds the extraction rules from the examples.

In contrast to other approaches [4, 6, 8], a key feature of STALKER is that it is able to efficiently generate extraction rules from a small number of examples: it rarely requires more than 10 examples, and in many cases two examples are sufficient. The ability to generalize from such a small number of examples has a two-fold explanation. First, in most of the cases, the pages in a source are generated based on a fixed template that may have only a few variations. As STALKER tries to learn landmarks that are part of this template, it follows that for templates with little or no variations a handful of examples usually will be sufficient to induce reliable landmarks.

Second, STALKER exploits the *hierarchical* structure of the source to constrain the learning problem. More precisely, based on the schema of the data to be extracted, we *automatically* decompose one difficult problem (i.e., extract all items of interest) into a series of simpler ones. For instance, instead of using one complex rule that extracts all restaurant names, addresses and phone numbers from a page, we take a hierarchical approach. First we apply a rule that extracts the whole list of restaurants; then we use another rule to break the list into tuples that correspond to individual restaurants; finally, from each such tuple we extract the name, address, and phone number of the corresponding restaurant. Our hierarchical approach also has the advantage of being able to extract data from pages that contain complicated formatting layouts (e.g., lists embedded in other lists) that previous approaches could not handle (see [11] for details).

STALKER is a sequential covering algorithm that, given the training examples E, tries to learn a minimal number of *perfect disjuncts* that cover *all* examples in E. By definition, a perfect disjunct is a rule that covers at least one training example and on any example the rule matches it produces the correct result. STALKER first creates an initial set of candidate-rules C and then repeatedly applies the following three steps until it generates a perfect disjunct:

- select most promising candidate from C
- refine that candidate
- add the resulting refinements to C

Once STALKER obtains a perfect disjunct P, it removes from E all examples on which P is correct, and the whole process is repeated until there are no more training examples in E. STALKER uses two types of refinements: *landmark refinements* and *topology refinements*. The former makes the rule more specific by adding a token to one of the existing landmarks, while the latter adds a new 1-token landmark to the rule.

For instance, let us assume that based on the four examples in Figure 3, we want to learn a start rule for the address. STALKER proceeds as follows. First, it selects an example, say `E4`, to guide the search. Second, it generates a set of *initial candidates*, which are rules that consist of a single 1-token landmark; these landmarks are chosen so that they match the token that *immediately precedes* the beginning of the address in the guiding example. In our case we have two initial candidates:

R5 = $SkipTo$(<b>)

R6 = $SkipTo$(*_HtmlTag_*)

As the <b> token appears only in `E4`, **R5** does not match within the other three examples. On the other hand, **R6** matches in all four examples, even though it matches *too early* (**R6** stops as soon as it encounters an HTML tag, which happens in all four examples *before* the beginning of the address). Because **R6** has a better generalization potential, STALKER selects **R6** for further refinements.

While refining **R6**, STALKER creates, among others, the new candidates **R7**, **R8**, **R9**, and **R10** shown below. The first two are obtained via landmark refinements (i.e., a token is added to the landmark in **R6**), while the other two rules are created by topology refinements (i.e., a new landmark is added to **R6**). As **R10** works correctly on all four examples, STALKER stops the learning process and returns **R10**.

R7 = $SkipTo$(: *_HtmlTag_*)

R8 = $SkipTo$(*_Punctuation_ _HtmlTag_*)

R9 = $SkipTo$(**:**) $SkipTo$(*_HtmlTag_*)

R10 = $SkipTo$(`Address`) $SkipTo$(*_HtmlTag_*)

By using STALKER, we were able to successfully wrap information sources that could not be wrapped with existing approaches (see [11] for details). In an empirical evaluation on 28 sources proposed in [8], STALKER had to learn 206 extraction rules. We learned 182 *perfect* rules (100% accurate), and another 18 rules that had an accuracy of at least 90%. In other words, only 3% of the learned rules were less that 90% accurate .

3 Identifying Highly Informative Examples

STALKER can do significantly better on the hard tasks (i.e., the ones for which it failed to learn perfect rules) if instead of $random$ examples, the system is provided with carefully selected examples. Specifically, the most informative examples illustrate exceptional cases. However, it is unrealistic to assume that a user is willing and has the skills to browse a large number of documents in order to identify a sufficient set of examples to learn perfect extraction rules. This is a general problem that none of the existing tools address, regardless of whether they use machine learning.

To solve this problem we have developed an *active learning* approach that analyzes the set of unlabeled examples to automatically select examples for the user to label. Our approach, called *co-testing* [10], exploits the fact that there are often multiple ways of extracting the same information [1]. In the case of wrapper learning, the system can learn two different types of rules: *forward* and *backward* rules. All the rules presented above are *forward* rules: they start at the beginning of the document and go towards the end. By contrast, a *backward* rule starts at the end of the page and goes towards its beginning. For example, one can find the beginning of the addresses in Figure 3 by using one of the following backward rules:

R11 = $BackTo$(`Phone`) $BackTo$(*_Number_*)

R12 = $BackTo$(`Phone : <i>`) $BackTo$(*_Number_*)

The main idea behind co-testing is straightforward: after the user labels one or two examples, the system learns *both* a forward and a backward rule. Then it runs *both* rules on a given set of unlabeled pages. Whenever the rules disagree on an example, the system considers that as an example for the user to label next. The intuition behind our approach is the following: if both rules are 100% accurate, on *every* page they must identify *the same* token as the beginning of the address. Furthermore, as the two rules are learned based on different sets of tokens (i.e., the sequences of tokens that precede and follow the beginning of the address, respectively), they are highly unlikely to make the exact same mistakes. Whenever the two rules disagree, at least one of them must be wrong, and by asking the user to label that particular example, we obtain a highly informative training example. Co-testing makes it possible to generate accurate extraction rules with a very small number of labeled examples.

To illustrate how co-testing works, consider again the examples in Figure 3. Since most of the restaurants in a city are *not* located within a 1-mile radius of one's location, it follows that most of the documents in the source will be similar to `E1`, `E2`, and `E3` (i.e., addresses shown in italic), while just a few examples will be similar to `E4` (i.e., addresses shown in bold). Consequently, it is unlikely that an address in bold will be present in a small, randomly chosen, initial training set. Let us now assume that the initial training set consists of `E1` and `E2`, while `E3` and `E4` are *not labeled*. Based on these examples, we learn the rules

Fwd-R1 = $SkipTo$(`Address`) $SkipTo$(`<i>`)

Bwd-R1 = $BackTo$(`Phone`) $BackTo$(*_Number_*)

Both rules correctly identify the beginning of the address for all restaurants that are more than one mile away, and, consequently, they will agree on all of them (e.g., `E3`). On the other hand, **Fwd-R1** works *incorrectly* for examples like `E4`, where it stops at the beginning of the phone number. As **Bwd-R1** is correct on `E4`, the two rules disagree on this example, and the user is asked to label it.

To our knowledge, there is no other wrapper induction algorithm that has the capability of identifying the most informative examples. In the related field of information extraction, where one wants to extract data from free text documents, researchers proposed such algorithms [13, 12], but they cannot be applied in a straightforward manner to existing wrapper induction algorithms.

We applied co-testing on the 24 tasks on which STALKER fails to learn perfect rules based on 10 random examples. To keep the comparison fair, co-testing started with one random example and made up to 9 queries. The results were excellent: the average accuracy over all tasks improved from 85.7% to 94.2% (error rate reduced by 59.5%). Furthermore, 10 of the learned rules were 100% accurate, while another 11 rules were at least 90% accurate. In these experiments as well as in other related tests [10] applying co-testing leads to a significant improvement in accuracy without having to label more training data.

4 Verifying the Extracted Data

Another problem that has been largely ignored in past work on extracting data from web sites is that sites change and they change often. Kushmerick [7] addressed the wrapper verification problem by monitoring a set of generic features, such as the density of numeric characters within a field, but this approach only detects certain types of changes. In contrast, we address this problem by applying machine learning techniques to learn a set of patterns that describe the information that is being extracted from each of the relevant fields. Since the information for even a single field can vary considerably, the system learns the statistical distribution of the patterns for each field. Wrappers can be verified by comparing the patterns of data returned to the learned statistical distribution. When a significant difference is found, an operator can be notified or we can automatically launch the wrapper repair process, which is described in the next section.

The learned patterns represent the structure, or format, of the field as a sequence of words and wildcards [9]. Wildcards represent syntactic categories to which words belong — alphabetic, numeric, capitalized, etc. — and allow for multi-level generalization. For complex fields, and for purposes of information extraction, it is sufficient to use only the starting and ending patterns as the description of the data field. For example, a set of street addresses — **12 Pico St.**, **512 Oak Blvd.**, **416 Main St.** and **97 Adams Blvd.** — all start with a pattern **(***_Number_ _Capitalized_***)** and end with **(Blvd.)** or **(St.)**. We refer to the starting and ending patterns together as the *data prototype* of the field.

The problem of learning a description of a field (class) from a set of labeled examples can be posed in two related ways: as a classification or a conservation task. If negative examples are present, the classification algorithm learns a *discriminating* description. When only positive examples are available, the conservation task learns a *characteristic* description, *i.e.* one that describes many of the positive examples but is highly unlikely to describe a randomly generated example. Because an appropriate set of negative examples not available in our case, we chose to frame the learning problem as a conservation task. The algorithm we developed, called DataPro [9], finds all statistically significant starting and ending patterns in a set of positive examples of the field. A pattern is said to be significant if it occurs more frequently than would be expected by chance if the tokens were generated randomly and independently of one another. Our approach is similar to work on grammar induction [2, 5], but our pattern language is better suited to capturing the regularities in small data fields (as opposed to languages).

The algorithm operates by build building a prefix tree, where each node corresponds to a token whose position in the sequence is given by the node's depth in the tree. Every path through the tree starting at the root node is a significant pattern found by the algorithm.

The algorithm grows the tree incrementally. Adding a child to a node corresponds to extending the node's pattern by a single token. Thus, each child represents a different way to specialize the pattern. For example, when learning city names, the node corresponding to "New" might have three children, corresponding to the patterns "New Haven", "New York" and "New *_Capitalized_*". A child node is judged to be significant with respect to its parent node if the number of occurrences of the child pattern is sufficiently large, given the number of occurrences of the parent pattern and the baseline probability of the token used to extend the parent pattern. A pruning step insures that each child node is also significant given its more specific siblings. For example, if there are 10 occurrence of "New Haven", and 12 occurrences of "New York", and both of these are judged to be significant, then "New *_Capitalized_*" will be retained only if there are significantly more than 22 examples that match "New *_Capitalized_*". Similarly, once the entire tree has been expanded, the algorithm includes a pattern extraction step that traverses the tree, checking whether the pattern "New" is still significant given the more specific patterns "New York", "New Haven" and "New *_Capitalized_*". In other words, DataPro decides whether the examples described by "New" but not by any of the longer sequences can be explained by the null hypothesis.

The patterns learned by DataPro lend themselves to the data validation task and, specifically, to wrapper verification. A set of queries is used to retrieve HTML pages from which the wrapper extracts some training examples. The learning algorithm finds the patterns that describe the common beginnings and endings of each field of the training examples. In the verification phase, the wrapper generates a test set of examples from pages retrieved using the same or similar set of queries. If the patterns describe statistically the same (at a given significance level) proportion of the test examples as the training examples, the wrapper is judged to be extracting correctly; otherwise, it is judged to have failed.

Once we have learned the patterns, which represent our expectations about the format of data, we can then configure the wrappers to verify the extracted data before the data is sent, immediately after the results are sent, or on some regular frequency. The most appropriate verification method depends on the particular application and the tradeoff between response time and data accuracy.

We monitored 27 wrappers (representing 23 distinct Web sources) over a period of several months. For each wrapper, the results of 15-30 queries were stored periodically. All new results were compared with the last correct wrapper output (training examples). A manual check of the results revealed 37 wrapper changes out of the total 443 comparisons. The verification algorithm correctly discovered 35 of these changes. The algorithm incorrectly decided that the wrapper has changed in 40 cases. We are currently working to reduce the high rate of false positives.

5 Automatically Repairing Wrappers

Most changes to Web sites are largely syntactic and are often minor formatting changes or slight reorganizations of a page. Since the content of the fields tend to remain the same, we exploit the patterns learned for verifying the extracted results to locate correct examples of the data field on new pages. Once the required information has been located, the pages are automatically re-labeled and the labeled examples are re-run through the inductive learning process to produce the correct rules for this site.

Specifically, the wrapper reinduction algorithm takes a set of training examples and a set of pages from the same source and uses machine learning techniques to identify examples of the data field on new pages. First, it learns the starting and ending patterns that describe each field of the training examples. Next, each new page is scanned to identify all text segments that begin with one of the starting patterns and end with one of the ending patterns. Those segments, which we call candidates, that are significantly longer or shorter than the training examples are eliminated from the set, often still leaving hundreds of candidates per page. The candidates are then clustered to identify subgroups that share common features. The features used in clustering are the candidate's relative position on the page, adjacent landmarks, and whether it is visible to the user. Each group is then given a score based on how similar it is to the training examples. We expect the highest ranked group to contain the correct examples of the data field.

Figure 4 shows a actual example of a change to Amazon's site and how our system automatically adapts to the change. The top frame shows an example of the original site, the extraction rule for a book title, and the extracted results from the example page. The middle frame shows the source and the incorrectly extracted result after the title's font and color were changed. And the bottom frame shows the result of the automatic reinduction process with the corrected rule for extracting the title.

We evaluated the algorithm by using it to extract data from Web pages for which correct output is known.

Figure 4: **An Example of the Reinduction Process**

The output of the extraction algorithm is a ranked list of clusters for every data field being extracted. Each cluster is checked manually, and it is judged to be correct if it contains only the complete instances of the field, which appear in the correct context on the page. For example, if extracting a city of an address, we only want to extract those city names that are part of an address.

We ran the extraction algorithm for 21 distinct Web sources, attempting to extract 77 data fields from all the sources. In 62 cases the top ranked cluster contained correct complete instances of the data field. In eight cases the correct cluster was ranker lower, while in six cases no candidates were identified on the pages. The most closely related work is that of Cohen [3], which uses an information retrieval approach to relocating the information on a page. The approach was not evaluated on actual changes to Web pages, so it is difficult to assess whether this approach would work in practice.

6 Discussion

Our work addresses the complete wrapper creation problem, which includes:

- building wrappers by example,
- ensuring that the wrappers accurately extract data across an entire collection of pages,
- verifying a wrapper to avoid failures when a site changes,

- and automatically repair wrappers in response to changes in layout or format.

Our main technical contribution is in the use of machine learning to accomplish all of these tasks. Essentially, our approach takes advantage of the fact that web pages have a high degree of regular structure. By analyzing the regular structure of example pages, our wrapper induction process can detect landmarks that enable us to extract desired fields. After we developed an initial wrapper induction process we realized that the accuracy of the induction method can be improved by simultaneously learning "forward" and "backward" extraction rules to identify exception cases. Again, what makes this possible is the regularities on a page that enable us to identify landmarks both before a field and after a field.

Our approach to automatically detecting wrapper breakages and repairing them capitalizes on the regular structure of the extracted fields themselves. Once a wrapper has been initially created, we can use it to obtain numerous examples of the fields. This enables us to profile the information in the fields and obtain structural descriptions that we can use during monitoring and repair. Essentially this is a bootstrapping approach. Given the initial examples provided by the user, we first learn a wrapper. Then we use this wrapper to obtain many more examples which we then analyze in much greater depth. Thus by leveraging a few human-provided examples, we end up with a highly scalable system for wrapper creation and maintenance.

Acknowledgements

The research reported here was supported in part by the Rome Laboratory of the Air Force Systems Command and the Defense Advanced Research Projects Agency (DARPA) under contract number F30602-98-2-0109, in part by the United States Air Force under contract number F49620-98-1-0046, and in part by the Integrated Media Systems Center, a National Science Foundation Engineering Research Center, under Cooperative Agreement No. EEC-9529152. The views and conclusions contained herein are those of the authors and should not be interpreted as necessarily representing the official policies or endorsements, either expressed or implied, of any of the above organizations or any person connected with them.

References

[1] A. Blum and T. Mitchell. Combining labeled and unlabeled data with co-training. In *Proc. of the 1988 Conference on Computational Learning Theory*, pages 92–100, 1998.

[2] R. Carrasco and J. Oncina. Learning stochastic regular grammars by means of a state merging method. In *Lecture Notes In Computer Science*, page 862, 1994.

[3] W. Cohen. Recognizing structure in web pages using similarity queries. In *Proc. of the 16th National Conference on Artificial Intelligence AAAI-1999*, pages 59–66, 1999.

[4] D. Freitag and N. Kushmerick. Boosted wrapper induction. In *Proc. of the 17th National Conference on Artificial Intelligence AAAI-2000*, pages 577–583, 2000.

[5] T. Goan, N. Benson, and O. Etzioni. A grammar inference algorithm for the world wide web. In *Proc. of the AAAI Spring Symposium on Machine Learning in Information Access*, 1996.

[6] C. Hsu and M. Dung. Generating finite-state transducers for semi-structured data extraction from the web. *Journal of Information Systems*, 23(8):521–538, 1998.

[7] N. Kushmerick. Regression testing for wrapper maintenance. In *Proc. of the 16th National Conference on Artificial Intelligence AAAI-1999*, pages 74–79, 1999.

[8] N. Kushmerick. Wrapper induction: efficiency and expressiveness. *Artificial Intelligence Journal*, 118(1-2):15–68, 2000.

[9] K. Lerman and S. Minton. Learning the common structure of data. In *Proc. of the 17th National Conference on Artificial Intelligence AAAI-2000*, pages 609–614, 2000.

[10] I. Muslea, S. Minton, and C. Knoblock. Co-testing: Selective sampling with redundant views. In *Proc. of the 17th National Conference on Artificial Intelligence AAAI-2000*, pages 621–626, 2000.

[11] I. Muslea, S. Minton, and C. Knoblock. Hierarchical wrapper induction for semistructured information sources. *Journal of Autonomous Agents and Multi-Agent Systems*, 2001. (to appear).

[12] S. Soderland. Learning extraction rules for semi-structured and free text. *Machine Learning*, 34:233–272, 1999.

[13] C. Thompson, M. Califf, and R. Mooney. Active learning for natural language parsing and information extraction. In *Proc. of the 16th International Conference on Machine Learning ICML-99*, pages 406–414, 1999.

ARKTOS: A Tool For Data Cleaning and Transformation in Data Warehouse Environments

Panos Vassiliadis Zografoula Vagena Spiros Skiadopoulos Nikos Karayannidis
Timos Sellis
Knowledge and Database Systems Laboratory
Dept. of Electrical and Computer Engineering
National Technical University of Athens
{pvassil, zvagena, spiros, nikos, timos}@dbnet.ece.ntua.gr

Abstract

Extraction-Transformation-Loading (ETL) and Data Cleaning tools are pieces of software responsible for the extraction of data from several sources, their cleaning, customization and insertion into a data warehouse. To deal with the complexity and efficiency of the transformation and cleaning tasks we have developed a tool, namely ARKTOS*, capable of modeling and executing practical scenarios, by providing explicit primitives for the capturing of common tasks.* ARKTOS *provides three ways to describe such a scenario, including a graphical point-and-click front end and two declarative languages: XADL (an XML variant), which is more verbose and easy to read and SADL (an SQL-like language) which has a quite compact syntax and is, thus, easier for authoring.*

1 Introduction

A data warehouse is a heterogeneous environment where data must be integrated both at the schema and at the instance level [CGL+98]. Practice has shown that neither the accumulation nor the storage process of the information seem to be completely credible. Errors in databases have been reported to be up to 10% range and even higher in a variety of applications [WRK95]. [WSF95] report that more than $2 billion of U.S. federal loan money had been lost because of poor data quality at a single agency; manufacturing companies spent over 25% of their sales on wasteful practices. The number came up to 40% for service companies. Clearly, as a decision support information system, a data warehouse must provide high level quality of data and service. In various vertical markets (e.g., the public sector) data quality is not an option but a strict requirement for the proper operation of the data warehouse. Thus, data quality problems seem to introduce even more complexity and computational burden to the loading of the data warehouse.

To deal with the complexity of the data warehouse loading process, specialized tools are already available in the market, under the general title *Extraction-Transformation-Loading* (ETL) tools [Evo00, Ard00, Dat00]. ETL as well as *Data Cleaning* tools are pieces of software responsible for the extraction of data from several sources, their cleaning, customization and insertion into a data warehouse.

A study for Merrill Lynch [ST98] reports some very interesting facts about the situation in the area of ETL and Data Cleaning tools, by the end of 1998. These tools cover a labor-intensive and complex part of the data warehouse processes, which is estimated to cost at least one third of effort and expenses in the budget of the data

Bulletin of the IEEE Computer Society Technical Committee on Data Engineering

warehouse. [Dem97] mentions that this number can rise up to 80% of the development time in a data warehouse project. Still, due to the complexity and the long learning curve of these tools [ST98], many organizations prefer to turn to in-house development to perform ETL and data cleaning tasks. The major problems with data cleaning tools are complexity and price. Moreover, due to the nature of the IT departments, which are constrained in time and budget, tools for off-line tasks like data cleaning are pushed aside from the list of products to purchase.

Several commercial tools exist in the market [Evo00, Ard00, Dat00]; a detailed list can be found in [LGI00]. Research efforts include the AJAX prototype developed at INRIA [GFSS00, GFSS99]. A discussion of the research literature on data quality is found in [WSF95]. Finally, a description of our personal engagement in practical aspects of ETL and cleaning in data warehouses can be found in [Vas00]. In the sequel, we will not discriminate between the tasks of ETL and Data Cleaning and adopt the name ETL for both kinds of activities.

In order to pursue research in the field of data transformation and cleaning, we have developed a tool, namely ARKTOS, to achieve the following goals:

- graphical and declarative facilities for the definition of data warehouse transformation and cleaning tasks;
- measurement of the quality of data through specific quality factors;
- optimized execution of complex sequences of transformation and cleaning tasks.

In the rest of this paper we will present the basic ideas behind the implementation and functionality of ARKTOS as well as the declarative languages for the specification of transformation and cleaning scenarios.

2 Model and Functionality of ARKTOS

ARKTOS is based on a simple metamodel to achieve the desired functionality. The main process type of the model is the entity *Activity*. An activity is an atomic unit of work and a discrete step in the chain of data processing. Since activities in a data warehouse context are supposed to process data in a data flow, each activity is linked to *Input* and *Output* tables of one or more databases. An SQL statement gives the *logical*, declarative description of the work performed by each activity (without obligatorily being identical to the actual, *physical* code that performs the execution of the activity). A *Scenario* is a set of processes to be executed all together. A scenario can be considered as the outcome of a design process, where the designer tailors the set of activities that will populate the data warehouse. Each activity is accompanied by an *Error Type* and a *Policy*. Since data are expected to encounter quality problems, we assume that several activities should be dedicated to the elimination of these problems (e.g., the violation of the primary or the foreign key constraint). The error type of an activity identifies the problem the process is concerned with. The policy, on the other hand, signifies the way the offending data should be treated. Around each activity, several *Quality Factors* can be defined. The quality factors are measurements performed to characterize the quality of the underlying information. For the moment, quality factors in ARKTOS are implemented through the use of SQL queries. ARKTOS is also enriched with a set of "template" generic entities that correspond to the most popular data cleaning tasks (like primary or foreign key violations) and policies (like deleting offending rows, reporting offending rows to a file or table).

Connectivity. ARKTOS uses JDBC to perform its connections to the underlying data stores.

Transformation and cleaning primitives. ARKTOS offers a rich variety of primitive operations to support the ETL process. More specifically, the cleaning primitives include: (a) *Primary key violation*, (b) *Reference violation*, (c) *NULL value existence*, (d) *Uniqueness violation* and (e) *Domain mismatch*. Moreover, the tool offers *Propagation* and *Transformation* primitive operations. The propagation primitive simply pushes data to the next layer of storage. The transformation primitive transforms the data to the desired format, according to some pattern (which can be either built-in or user-defined). For example, a transformation primitive can be used to transform a date field from dd/mm/yy to dd/mm/yyyy format. These primitives are customized by the user (graphically or declaratively). The customization includes the specification of input and output (if necessary) data stores, contingency policy and quality factors. For example, in the current version of ARKTOS, the format transformation functions are performed programmatically using a freely available Java Package [ORO00] that simulates the functionality of Perl regular expressions.

Contingency policy. Once a primitive filter is defined in the context of a scenario, it is possible that some rows fulfill its criteria at runtime. For example, a particular row might violate the foreign key constraint for one of its attributes. For each such filter, the user is able to specify a policy for the treatment of the violating rows. For the moment, the policies supported by ARKTOS are: (a) *Ignore* (i.e., do not react to the error and let the row pass), (b) *Delete* (from the source data store), (c) *Report to a contingency file* and (d) *Report to a contingency table*. It is possible that the user is requested to supply some additional parameters (for example, the name of the file where the rows should be reported, or the format to which the values should be changed).

Trace Management. The physical properties of the execution of the ARKTOS scenarios are captured by detailed log information kept for this reason. The *status*, *initialization*, *commit* or *abort* information for each execution of an activity is traced.

Scheduling. ARKTOS uses a freely available Java package [Bra99] to schedule the execution of scenarios that have already been created and saved by the user. To perform this task, the user has to specify, in the correct order, the name(s) of the files where the appropriate scenarios reside. Each of the scenarios participating in a schedule can be executed either once, at a specific time point, or on the basis of a specified repetition frequency (e.g., every Monday, or every 23rd day of each month, at 11:30 am).

3 Declarative Languages for ETL Processes

As we have already pointed out, that the major obstacles the ETL tools have to overcome are the issues of user-friendliness and complexity. To this end, ARKTOS offers two possible ways for the definition of activities: *graphically* and *declaratively*. The graphical definition is supported from a palette with all the possible activities currently provided by ARKTOS. The user composes the scenario from these primitives, links them in a serial list and resolves any pending issues of their definition. In the rest of this section we will focus on the declarative definition of data warehouse processes in the ARKTOS environment.

There is a classical problem with declarative languages and formal specifications: the languages which are easy to read are hard to write and vice-versa. To overcome the problem we resort to two declarative definition languages:

- XADL (*XML-based Activity Definition Language*), an XML language for data warehouse processes, on the basis of a well-defined DTD;
- SADL (*Simple Activity Definition Language*), a declarative definition language motivated from the SQL paradigm.

The former language is rather verbose and complex to write; yet it is more comprehensible. The latter is more compact and resembles SQL; thus it is suitable mostly for the trained designer. We next give an informal presentation of the two languages, by using a motivating example (Figure 1) based on the former TPC-D standard (now TPC-H and TPC-R) [Tra00].

1. Push data from table LINEITEM of source database S to table LINEITEM of the DW database.
2. Perform a referential integrity violation checking for the foreign key of table LINEITEM in database DW, which is referencing table ORDER. Delete violating rows.
3. Perform a primary key violation check to the table LINEITEM. Report violating rows to a file.

Figure 1: Description of the scenario of the motivating example

3.1 XADL (XML-based Activity Definition Language)

In Figure 2 we illustrate a subset of the XADL definition for the scenario of the motivating example. Lines 67-102 describe a simple activity. First, in Lines 68-85 the structure of the input table is given. Lines 86-92 describe the error type (i.e., the functionality) of the activity which involves foreign key constraint violations. The target column and table are specifically described. Lines 93-95 deal with the policy followed for the identified records and declare that in this case, we simply delete them. The quality factors of the activity are described in Lines

96-101. Each quality factor is characterized by the SQL query that computes its value and the report file where this value will be stored. The only quality factor in Figure 2 is the absolute number of violating rows and is characterized by the SQL query of Line 97. For any valid scenario that we load in ARKTOS, its XADL description can be automatically generated by the tool.

```
1. <?xml version="1.0" encoding="ISO-8859-1" standalone="yes"?>
...
67. <transformtype>
68.     <input_table table_name="lineitem" database_url="jdbc:informix-sqli:
         //kythira.dbnet.ece.ntua.gr:1500/dbs3:informixserver=ol_milos_tcp">
69.     <column> l_orderkey </column>
70.     <column> l_partkey </column>
...
85.     </input_table>
86.     <errortype>
87.         <reference_violation>
88.             <target_column_name> l_orderkey </target_column_name>
89.             <referenced_table_name> Informix.tpcd.tpcd.tpcd.order </referenced_table_name>
90.             <referenced_column_name> o_orderkey </referenced_column_name>
91.         </reference_violation>
92.     </errortype>
93.     <policy>  <delete/>  </policy>
94.     <quality_factor qf_name=No_of_reference_violations qf_report_file="H:\path\scenario3.txt">
95.         <sql_query> select l_orderkey from lineitem t1 where not exists
                     (select o_orderkey from order t2 where t1.l_orderkey = t2.o_orderkey)
            </sql_query>
96.     </quality_factor>
97. </transformtype>
...
140. </scenario>
```

Figure 2: XADL definition of the scenario of the motivating example, as exported by ARKTOS

3.2 SADL (Simple Activity Definition Language)

The SADL language is composed of four definition statements: the `CREATE SCENARIO`, `CREATE CONNECTION`, `CREATE ACTIVITY` and `CREATE QUALITY FACTOR` statements. A `CREATE CONNECTION` statement specifies the details of each database connection. A `CREATE ACTIVITY` statement specifies an activity and a `CREATE QUALITY FACTOR` statement specifies a quality factor for a particular activity. The `CREATE SCENARIO` statement ties all the elements of a scenario together. In Figure 3 we depict the syntax of these four statements.

```
CREATE SCENARIO <scenario_name>
WITH CONNECTIONS <con_1,...,con_m>
ACTIVITIES <act_1,...,act_n>

CREATE CONNECTION <connection_name>
WITH DATABASE <url> ALIAS <db_alias>
[USER <user_name> PASSWORD <password>]
DRIVER <class_name>
```

```
CREATE QUALITY FACTOR <qf_name> WITH ACTIVITY <activity_name>
REPORT TO <file_name>
SEMANTICS <SQL query>

CREATE ACTIVITY <activity_name> WITH TYPE <error_type>
POLICY <policy_type>
[OUTPUT <output_name>(<attr_1,...,attr_m>)]
SEMANTICS <SQL query>
```

Figure 3: The syntax of SADL for the `CREATE SCENARIO`, `CONNECTION`, `ACTIVITY`, `QUALITY FACTOR` statements

Connections and activities are the first-class citizens within the context of a scenario. Thus, to declare a `CREATE SCENARIO` statement one has simply to provide the names of the respective connections and the activities of the scenario. The definition of a connection, through the `CREATE CONNECTION` statement is equally simple: the database URL and the class name of the respective driver are required. Since the database URL is quite big in size for the user to write down, an `ALIAS` clause is introduced. All table names are required to be in the form `<table_name>@<database alias>` to distinguish between synonym tables in different databases. The username and password are optional (in order to avoid storing them in the file). `CREATE QUALITY FACTOR` is also a simple statement: one has to specify the activity in the context of which a quality factor is defined, the report to which the value of the quality factor will be saved and the semantics of the quality factor, expressed by an SQL statement (in practice any SQL statement that the database driver and JDBC can support).

The CREATE ACTIVITY statement is somewhat more complex. One has to specify first the functionality of the activity in the TYPE clause. The <error_type> placeholder can take values from the set {PUSH, UNIQUESS VIOLATION, NULL EXISTENCE, DOMAIN MISMATCH, PRIMARY KEY VIOLATION, REFERENCE VIOLATION, FORMAT MISMATCH}. The POLICY clause determines the treatment of the rows affected by the activity. The <policy_type> belongs to the set {IGNORE, DELETE, REPORT TO FILE, REPORT TO TABLE}. The OUTPUT clause specifies the target table or file (if there exists one). If a table is to be populated, all the relevant attributes are specified too. The order of the attributes is important (it must be in one-to-one correspondence with the attributes of the input tables as specified in the SQL query, which will be described later). The SEMANTICS clause is filled with an arbitrary SQL query. Several issues should be noted for this clause:

- the order of the attributes in the OUTPUT clause should be in accordance with the order of the attributes in the SELECT clause of the SQL query;
- the input tables are described in the FROM clause of the SQL statement;
- the order of the activities in the CREATE SCENARIO statement is important, because it denotes the flow of the activities.

Primitive Operation	SQL statement	SEMANTICS clause shortcuts
UNIQUENESS VIOLATION	SELECT * FROM <table> GROUP BY <attribute> HAVING COUNT(*) > 1	<table>.<attribute>
NULL EXISTENCE	SELECT * FROM <table> WHERE <attribute> IS NULL	<table>.<attribute>
DOMAIN MISMATCH*	SELECT * FROM <table> WHERE <attribute> NOT IN <domain specification>	<table>.<attribute> NOT IN <domain specification>
PRIMARY KEY VIOLATION	SELECT * FROM <table> GROUP BY (<attribute_1,...,attribute_n>) HAVING COUNT(*) > 1	<table>.(<attribute_1, ..., attribute_n>)
REFERENCE VIOLATION	SELECT * FROM <table> WHERE <attribute> NOT IN (SELECT <target_attribute) FROM <target_table>)	<table>.<attribute> NOT IN <target_table>.<target_attribute>
FORMAT MISMATCH**	SELECT APPLY(<reg_exp>,<attribute>) FROM <table> WHERE APPLY(<reg_exp>,<attribute>)	TARGET APPLY(<reg_exp>,<attribute>) SOURCE APPLY(<reg_exp>,<attribute>)
PUSH	Arbitrary SQL query	Arbitrary SQL query

* works for intervals of numbers and strings

** where <reg_exp> is PERL regular expression acting as a formatting function

Figure 4: The SADL specification for the basic primitives offered by ARKTOS.

There are standard CREATE ACTIVITY statements for all the primitives (i.e., the specialized activities) offered by ARKTOS. In Figure 4 we list them along with syntactic sugar shortcuts which make the life of the designer much easier (remember that the type of the operation is given in the TYPE clause of the CREATE ACTIVITY statement). Note again that in XADL and SADL we refer only to the logical semantics of the activities and not to the way they are actually executed within the DBMS, which is hard-coded in the subclasses of the ARKTOS architecture.

Figure 5 expresses our motivating example in SADL. In Lines 1-4 we define our scenario, which consists of three activities. The order of the activities appearing in the figure is in descending execution priority. The connection characteristics for connecting to the data warehouse are declared in Lines 6-9. An example of the SADL description of an activity can be seen in Lines 11-16 for the reference violation checking activity. Finally, in Lines 18-22 we give the declaration of a quality factor, which reports to a file the number of foreign key violating rows.

4 Conclusions and Future Work

In this paper, we have presented ARKTOS, a tool we have developed for modeling and executing practical data management scenarios by providing explicit primitives for the capturing of common tasks (like data cleaning, scheduling and data transformations). Within ARKTOS, we provide three ways to describe such a scenario: a

```
1. CREATE SCENARIO Scenario3 WITH
2. CONNECTIONS S3,DW
3. ACTIVITIES Push_lnitem, Fk_lnitem, Pk_lnitem
4. ...
5. CREATE CONNECTION DW WITH
6. DATABASE "jdbc:informix-sqli://kythira.dbnet.ece.ntua.gr:1500/
    dbdw:informixserver=ol_milos_tcp" ALIAS DBDW
7. DRIVER "com.informix.jdbc.IfxDriver"
8. ...
9. CREATE ACTIVITY Fk_lnitem WITH
10. TYPE REFERENCE VIOLATION
11. POLICY DELETE
12. SEMANTICS "select l_orderkey from lineitem@DBDW t1 where not exists
     (select o_orderkey from order@DBDW t2 where t1.l_orderkey=t2.o_orderkey)"
13. ...
14. CREATE QUALITY FACTOR "# of reference violations" WITH
15. ACTIVITY fk_lnitem
16. REPORT TO "H:\path\scenario3.txt"
17. SEMANTICS "select l_orderkey from lineitem@DBDW t1 where not exists
           (select o_orderkey from order@DBDW t2 where t1.l_orderkey = t2.o_orderkey)"
```

Figure 5: Part of scenario 3 expressed in SADL

graphical point-and-click front end and two declarative languages. The first one, XADL (an XML variant) is oriented towards an easily understood description of a scenario. The second one, SADL is tailored to support the declarative definition of the ETL scenario in an SQL-like style.

In the future we plan to add more functionality to ARKTOS, in order to provide the users with richer transformation primitives. Several research issues remain open, such as (a) the development of an impact analyzer, based on the results of [VQVJ00], showing how changes in the definition of a table or an activity affect other tables or activities in the data warehouse; (b) the linkage to a metadata repository, and specifically ConceptBase [JGJ+95], in order to exploit its enhanced query facilities, and (c) the construction of an optimizer to attain improved efficiency during the execution of composite scenarios.

References

[Ard00] Ardent Software. DataStage Suite, 2000. See also www.ardentsoftware.com.

[Bra99] Branch Cut Software. JTask: Java Task Scheduler, 1999. Available at www.branchcut.com/jTask.

[CGL+98] D. Calvanese, G. De Giacomo, M. Lenzerini, D. Nardi, and R. Rosati. Information integration: Conceptual modeling and reasoning support. In *Proceedings of CoopIS'98*, pages 280–291, 1998.

[Dat00] DataMirror Corporation. Transformation Server, 2000. See also www.datamirror.com.

[Dem97] M. Demarest. The politics of data warehousing, 1997. Available at www.hevanet.com/demarest/marc/dwpol.html.

[Evo00] Evolutionary Technologies International. ETI*EXTRACT, 2000. See also www.eti.com.

[GFSS99] H. Galhardas, D. Florescu, D. Shasha, and E. Simon. An extensible framework for data cleaning. Technical Report RR-3742, INRIA, 1999.

[GFSS00] H. Galhardas, D. Florescu, D. Shasha, and E. Simon. Ajax: An Extensible Data Cleaning Tool. In *Proceedings of ACM SIGMOD-2000*, June 2000.

[JGJ+95] M. Jarke, R. Gallersdorfer, M.A. Jeusfeld, M. Staudt, and S. Eherer. ConceptBase - a deductive objectbase for meta data management. *Intelligent Information Systems*, 4(2), 1995.

[LGI00] LGI Systems Inc. The Data Warehousing Information Center, 2000. See also www.dwinfocenter.org.

[ORO00] ORO Inc. PerlTools 1.2, 2000. Available at www.savarese.org/oro.

[ST98] C. Shilakes and J. Tylman. Enterprise Information Portals. Enterprise Software Team, November 1998. Available at www.sagemaker.com/company/downloads/eip/indepth.pdf.

[Tra00] Transaction Processing Performance Council. TPC-H and TPC-R, 2000. Available at www.tpc.org.

[Vas00] P. Vassiliadis. Gulliver in the land of data warehousing: practical experiences and observations of a researcher. In *Proceedings of DMDW'2000*, June 2000.

[VQVJ00] P. Vassiliadis, C. Quix, Y. Vassiliou, and M. Jarke. A Model for Data Warehouse Operational Processes. In *Proceedings of CAiSE'00*, June 2000.

[WRK95] R.Y. Wang, M.P. Reddy, and H.B. Kon. Towards Quality Data: An attribute-based Approach. *Decision Support Systems*, 13, 1995.

[WSF95] R.Y. Wang, V.C. Storey, and C.P. Firth. A Framework for Analysis of Data Quality Research. *IEEE Transactions on Knowledge and Data Engineering*, 7(4), 1995.

ICDE 2001

in the Heart of Europe

The 17th International Conference on Data Engineering

April 2–6, 2001

Heidelberg, Germany

The 17th International Conference on Data Engineering (ICDE 2001) will be held in a beautiful old town – Heidelberg. This is at the center of Germany's most active high-tech region where you find world renowned companies, numerous software startups, many world-class research centers such as the ABB Research Center, the Deutsche Telekom Research and Development Center, the European Media Laboratory, the European Molecular Biology Laboratory, the Fraunhofer Gesellschaft Institute for Graphical Data Processing, the German Center for Cancer Research, the GMD National Center for Information Technology, the Computer Science Research Center, the Karlsruhe Research Center, and a number of universities with strong database research groups (amongst others Darmstadt University of Technology, International University in Germany, University of Karlsruhe, University of Mannheim and not far away the University of Stuttgart). Such combination of strong industry, ground breaking research institutions, economic prosperity, and a beautiful host town provide an ideal environment for a conference on Data Engineering.

With tutorials, panels and industrial program.

Website & Registration at
http://www.congress-online.de/ICDE2001

Early Registration until the 2nd of March 2001

Topics Include:

XML, METADATA, and SEMISTRUCTURED DATA
DATABASE ENGINES & ENGINEERING
QUERY PROCESSING
DATA WAREHOUSES, DATA MINING, AND KNOWLEDGE DISCOVERY
ADVANCED IS MIDDLEWARE
SCIENTIFIC AND ENGINEERING DATABASES
EXTREME DATABASES
E-COMMERCE and E-SERVICES
WORKFLOW and PROCESS-ORIENTED SYSTEMS
EMERGING TRENDS
SYSTEM APPLICATIONS AND EXPERIENCE

Conference Officers

General Chairs:	
Andreas Reuter	European Media Laboratory and International University in Germany
David Lomet	Microsoft Research, USA
Program Co-chairs:	
Alex Buchmann	University of Darmstadt, Germany
Dimitrios Georgakopoulos	Telcordia Technologies, USA
Panel Program Chair:	
Erich Neuhold	GMD-IPSI, Germany
Tutorial Program Chair:	
Guido Moerkotte	University of Mannheim, Germany
Eric Simon	INRIA, France
Industrial Program Co-chairs:	
Peter Lockemann	University of Karlsruhe, Germany
Tamer Ozsu	University of Alberta, Canada
Steering committee liaison:	
Erich Neuhold	GMD-IPSI, Germany
Marek Rusinkiewicz	MCC, USA
Organizing Chair:	
Isabel Rojas	European Media Laboratory, Germany
Demos & Exhibits:	
Wolfgang Becker	International University in Germany
Andreas Eberhart	
Local Arrangements:	
Bärbel Mack	European Media Laboratory, Germany
Claudia Spahn	
Public Relations:	
Peter Saueressig	European Media Laboratory, Germany

IEEE COMPUTER SOCIETY

Sponsored by the IEEE Computer Society

Financial Support: Special support for travel expenses and conference fees will be available for participants from Eastern Europe.

IEEE Computer Society
1730 Massachusetts Ave, NW
Washington, D.C. 20036-1903

Non-profit Org.
U.S. Postage
PAID
Silver Spring, MD
Permit 1398