INSTANCE BASED TABLE INTEGRATION ALGORITHM FOR MULTILINGUAL TABLES ON THE WEB

Ikeda, Daisuke
Computing and Communications Center, Kyushu University

INSTANCE BASED TABLE INTEGRATION ALGORITHM FOR MULTILINGUAL TABLES ON THE WEB

By

Daisuke IKEDA*

Abstract

In this paper, we define the table integration problem which is, given two tables, to determine the correct mapping between fields of the tables. A table is a set of instances of a record which consists of fields. A field is a pair of an attribute name and a sequence of attribute values of the same type. We present an algorithm for the problem which uses only instance values of tables instead of schema and attribute names. Given tables, the algorithm calculates two numerical features for each field using character codes and then finds correspondence between fields among tables. The novelty of the algorithm is that it uses the character code chart for the language in which the contents of the tables are written. This enables a field to be represented by only two types of features. The algorithm requires neither an attribute value contained in all input tables nor attribute names. So, the algorithm is suitable for tables obtained from Web data, as long as they are written in the same language. Applying the algorithm for real Web data written in many languages, we demonstrate that the algorithm yields the accurate results and is robust for errors. The languages are Chinese, English, Germany, Japanese, and Korean.

Key Words and Phrases: Database integration, Information extraction from the Web, Probabilistic distribution, Multilingual data, Character code.

1. Introduction

An important feature of the Web is its diversity. There are a lot of types, purposes, and qualities of Web pages and they are written in many different kinds of languages. The diversity causes difficulties when we try to find, compare, and utilize wide variety of informations on the Web.

In the chaotic World Wide Web, however, many sites provide a series of informations of the same type. For example, search facilities, which are provided at many Web sites, return a list search results. An online news outlet publishes article pages on the Web with the same style and structure. Many efforts have been paid to extract contents from such sites semi- or full-automatically in the literatures such as Ashish and Knoblock (1997), Kushmerick et al. (1997), Lerman et al. (2001), Yamada et al.(2001), Yamada et al. (2002). A wrapper is a procedure which extracts the contents and generates tables from Web pages. We call such a table a *wrapper table*.

---

* Computing and Communications Center, Kyushu University, Hakozaki 6-10-1, Fukuoka 812-8581, Japan. Tel: +81-92-642-2296, Email: daisuke@cc.kyushu-u.ac.jp

Once a wrapper table is created, the next problem is to aggregate wrapper tables even though they are created from different sources. We can use integrated tables as a single database. Imagine that we have an integrated table of cars of various makers and that you want to buy a sedan. All you need is just to search the table.

In this paper, we define the table integration problem and present an algorithm for the problem. The algorithm is very simple and utilizes only basic techniques, but ample enough to integrate diverse tables on the Web written in many languages.

1.1. Related Works

A table can be seen as a simple database. The database integration problem has been paid many attention in the field of database over the years. However, there are several essential difference between databases and wrapper tables. A database is well-structured and a schema is explicitly given. Therefore, most integration algorithms are based on or require schema such as those developed in Cohen (1998), Doan et al. (2001), Madhavan et al. (2001), Miller (2000). On the other hand, a wrapper table does not have explicit schema. The original sources of the tables are markuped in HTML and do not have schema or attribute names. Therefore, methods developed in the database field are not applicable to this situation.

Many query based integration systems have been proposed in the field of database such as Genesereth et al. (1997), Levy et al. (1996), Miller (2000) as well. Such a system receives a query, decomposes it into sub-queries for individual database, and joins results. Therefore, a schema of each database must be given to the system. Knoblock et al. (1998) focused on usual Web sites instead of relational database, but the schema of a site must be described by a user. The main problem to integrate results is to determine whether two instances from different database are the same entity or not. Similarity measures are proposed in Cohen (1998), Monge and Elkan (1996) based on the edit distance, vector space model, etc.

The recent upsurge of semi-structured data poses the necessity of database integration in the new context. It is the integration of databases on the Web. The purpose seems to be similar to this paper, but the target and setting of their integration is completely different. In Chidlovskii (2002) and Popa et al. (2002), the main target is XML collections. A user can view HTML files whose original structured are destroyed even if their original sources are well-structure XML files (or relational database). In Yoshida et al. (2001), the target is a usual HTML file but the proposed algorithm requires for input files to contain attribute names and for the contents to be formatted in table tags. However, there are many pages which looks like a list or table without table tags and does not have attributes names. If pages have attribute names, notations might be different from other sources.

1.2. Instance Based Algorithms

We need an instance based integration algorithm because wrapper tables do not have any schema and attribute names. However, very few effort have been paid for instance based integration algorithms in the field of database. The algorithm in Doan et al.(2001) uses attribute values to train a learning algorithm which learns the correspondence of schemata. SEMINT developed in Li Clifton (2000) uses partly numerical features of attribute values. Numerical features are calculated by the number of up-

per/lower case letters, digit, punctuation, and so on. The algorithm concludes that two fields contains the same type of data if their attribute values look like similar.

Ideally, for an instance based algorithm, it would be the best way to find semantics of each field first and then to match fields using semantics instead of numerical features. For example, an algorithm judges the field containing "Daisuke Ikeda" to be the field for the persons names and the field containing "daisuke@cc.kyushu-u.ac.jp" to be one of their mail addresses. It seems, however, to be difficult especially in multilingual situations. Therefore, like SEMINT, it is reasonable for instance based algorithms to treat attribute values as just strings even if some of them are digit, and utilize numerical features of strings. One superiority of such numerical features is the independence from the domain of target databases.

In the fields of clustering and machine learning, there are many algorithms based on numerical features. Numerical features are used to learn classes of fields (Lerman and Minton (2000)) and cluster documents (Ma et al. (2003)). Lerman and Minton (2000) introduced the syntactic token hierarchy, whose root has three children "PUNCT," "AL-PHANUM," and "HTML." And, for example "ALPHANUM" have two children, "AL-PHA" which is the class for usual letter and "NUMBER" which is the class of digits. Kushmerick (1999) formulate the wrapper verification problem which is, given two wrapper tables, to decide whether they contains data of the same type. If not, the algorithm concludes that the source site changed the format. To solve the problem, Kushmerick adopted nine features, such as digit/letter density, upper/lower-case density, punctuation density, length, and so on. As an an application of the algorithm, the wrapper maintenance problem is discussed. A Web site often changes its page style, so we need to find correspondence of fields between before and after the change. Ma et al. (2003) also calculate a vector for each input document using the features, such as isUpperCaseLine (all alphabetics are upper case), isFirstUpperCaseLine (each word starts with upper case), startsWithDigit, and so on. They cluster documents with vector representation in a domain independent manner. These algorithms using numerical features are in fact domain independent but obviously heavily depend on languages. In described algorithms, only two types of letters, upper and lower case letters, are considered. However, in Japanese for example, we have two types of letters, Hiragana and Katakana, and one type of Chinese characters instead of upper and lower cases.

1.3. Our Contribution

In this paper, we define the table integration problem. The problem is, given tables, to find a mapping of fields among them. Then, we present an instance based table integration algorithm to solve the problem. We represent a field of a table as a vector in two dimensional space, which we call the *feature space*. The novelty of the representation is its low dimensionality. This gives a unified view of heterogeneous tables.

SEMINT used a feature space to represent a field by a vector in 20 statistical characteristics of a database. Only 5 of them are instance based. They are minimum, maximum, average, coefficient of variance, standard deviation for non-blanks for sizes[1] of all attribute values in a field. Using these characteristics, a field is converted to a vector in 20 dimensional space. Then, the algorithm clusters vectors in the space with

---

[1] A size is in byte.

respect to their Euclidean distance.

Like SEMINT, the proposed algorithm also calculates a feature vector of a field. But, in calculation, we make use of the character code chart[2] of the language used in input tables instead of sizes of attribute values. Any attribute value is treated as a string even if it is a digit and represented by the sum of all code numbers of it. "daisuke" is, for example in the ASCII (American Standard Code for Information Interchange) code, represented by 64, 61, 69, 73, 75, 6B, and 65 in hexadecimal. This representation is very natural. In fact, the programming language C does not distinguish a char (character) and an int (integer). Therefore, any languages are acceptable for input tables if the wrapper tables are written in the same language (character code chart) at a time.

A character code chart is a gold mine of information. The characters of the same type occupy a successive area. The areas for different types are completely distinct. In the ASCII code, the digits occupy the area from 31 to 39, upper case letters from 41 to 5A, and lower case letters from 61 to 7A. Hiragana letters occupy from 3040 to 309F in Unicode[3] and Katakana from 30A0 to 30FF. Therefore, character codes are expressive for characteristics of fields. Thus, our representation of field even in a low dimensionality works well to integrate wrapper tables.

If we have large amount of instances, we can find appropriate mapping using methods based on probabilistic distributions of instances' occurrences. However, we can not assume enough instances. Moreover, there is no instance which appears in both input tables in many cases. Therefore, we use character code charts instead of the probabilistic distributions.

We evaluate the effectiveness and limit of the presented algorithm by experiments on 10 tables. 9 of them are wrapper tables created from real data on the Web. The other table is created from an XML collection. They are written in 5 languages, Chinese, English, Germany, Japanese, and Korean.

For most datasets, the presented algorithm achieves 100% accuracy if enough instances are given. In our experiments, less than 50 instances are enough. Experiments show that the accuracy value is not affected by the contents language. Some experiments show that a limit of the algorithm. That is, the algorithm fails to match fields whose attribute values are semantically different but look similar. For example, in the list of tennis players, we have two such fields "Name" which is the field for tennis player names and "Coach" which contains also person's (coach for a player) names.

This paper is organized as follows: First of all, we define notations and the key representation of fields in Section 2.. Then, we present the main algorithm in Section 3.. In Section 4., we show experiments using wrapper table created from practical data written in Chinese, English, German, Japanese, and Korean.

2. Table Integration Problem

A table consists of instances of a *record*. A record instance contains different types of data, each of which is called a *field*. A field is a pair of an *attribute name* and a sequence[4] of *attribute values* of the same type. For a field f, we denote the number of all attribute values in f by $|f|$. For a table and any two fields f_1 and f_2, $|f_1| = |f_2|$.

Table 1 is a sample of table. Each column contains data of the same type, and

---

[2] More precisely, a character code chart is a character encoding scheme.
[3] http://www.unicode.org/
[4] That is, multiple values with the same value are allowed.

Table 1: Sample Table

Name	Phone	Email
D. Ikeda	2298	daisuke@cc.kyushu-u.ac.jp
S. Hirokawa	2301	hirokawa@cc.kyushu-u.ac.jp
Y. Yamada	2296	yshiro@cc.kyushu-u.ac.jp
...	...	...

a column corresponds to a field. In this table, each field has an attribute name such as "Name," "Phone," and "Email." Our algorithm does not require attribute names. Attribute names in Table 1 are only for expository purpose.

In out algorithm, an attribute value is treated as a string even if it consists of only digits. For an attribute value e, $|e|$ denotes the length of e[5] and $[e]_i$ the i character of e, where $1 \leq i \leq |e|$.

Without loss of generality, we can assume that the number of input tables are restricted to be two. If three or more tables are given, first we apply an integration algorithm to any two of them, then the iteratively application to an integrated table and another input table leads to the integration of all input tables.

For a table T, $F(T)$ denotes the set of all fields in T. We do not require any restriction on $F(T_1)$ and $F(T_2)$. We do no assume that tables such that $F(T_1) \subset F(T_2)$ or that $F(T_1)$ and $F(T_2)$ are comparable might be input tables. We assume that for any field in T_1, there exists at most one correspondent field in T_2.

Now, we define the table integration problem formally as follows:

DEFINITION 2.1 TABLE INTEGRATION. The table integration problem is, given two tables T_1 and T_2, to find the correspondence (f_1, f_2) between $f_1 \in F(T_1)$ and $f_2 \in F(T_2)$, where f_1 and f_2 are fields containing data of the same type.

3. Algorithm

Our algorithm receives two tables T_1, T_2 and a character code chart $C(\cdot)$ as an input, where $C(\cdot)$ is a function from characters to integers. The algorithm first transforms all fields into vectors in 2 or 3 dimensional space using the code chart $C(\cdot)$. The space is called the *feature space*. Then, it calculates distances for all pairs of fields among the tables and finds iteratively nearest pairs of fields with respect to the Euclid distance between them.

3.1. Field Representation

The algorithm utilizes a function fieldval() which returns the vector in the feature space for a given field. This function is the key representation of the proposed algorithm. fieldval() treats an attribute value e of a field as a string and calculates the sum of character codes $C(\cdot)$ for all characters in e. We call this sum the *feature value* for an attribute value. Now, we extend the domain of C from a character to a string. For a string x, $C(x)$ is defined to $\sum_{i=1}^{|x|} [x]_i$. Then the feature value for an attribute value e is $C(e)$.

---

[5] This notation is the same as the number of all attribute values of a field f. However, the difference is clear from the context.

```
function fieldval(f: filed, C: code chart): vector
begin
    for e in all instances of f
    do
        v_e := ∑_{i=1}^{|e|} C([e]_i)
    end
    a := average value of all v_e's
    s := standard deviation of all v_e's
    return (a, s)
end
```

$$v_e := \sum_{i=1}^{|e|} C([e]_i)$$

Figure 1: Pseudo code to calculate the vector of a field

The *vector* v for a field is $v = (x, y)$, where x and y are the average and standard deviation value of all feature values in the field.

This representation exhibits characteristics of an attribute value's appearance. For an attribute value e, v_e contains informations about both lengths and character types used in e, but omits the order of characters. Therefore, the substring information is ignored. Table 2 shows vectors for sample instances in Table 1. Note that special

Table 2: Attribute values and their feature values by the ASCII code chart

Name	$C(\cdot)$	Phone	$C(\cdot)$	Email	$C(\cdot)$
D. Ikeda	624	2298	213	daisuke@cc.kyushu-u.ac.jp	2399
S. Hirokawa	980	2301	198	hirokawa@cc.kyushu-u.ac.jp	2511
Y. Yamada	850	2296	211	yshiro@cc.kyushu-u.ac.jp	2811
...	...	...	...	...	...

characters, such as a space and punctuation, are used as-is. For example, "D. Ikeda" is translated as $C(D) + C(.) + C(\sqcup) + C(I) + \cdots + C(a) = 68 + 46 + 32 + 73 + \cdots + 97 = 624$[6]. Note that a feature value only depends on one attribute value, it is not necessary for an integration algorithm to see all attribute values in a field. In this sense, a vector in this 2D feature space is locally definable.

All attribute values in a field of Table 1 are basically different from each other. In general, however, there are fields containing some fixed optional values. For example, a sex is either male or female and a job title is selected from some fixed number of candidates. The standard deviation for such a field is expected to be low. It seems to be difficult distinguish such a field from another field which contains wide variety of attribute values but its standard deviation happens to be low.

Here, we introduce another numerical feature for experimentally purpose. Let m be the number of different attribute values in a field f. Then, the vector v for a field f is $v = (x, y, z)$, where x and y are the same as the 2D case, and $z = |f|/m$. We use $|f|/m$ instead of m as-is because the difference of m between fields are important especially when m is too small. Note that this new feature is not defined for an attribute value. So, a vector in this new feature space is *not* locally definable.

---

[6] A code number is in decimal.

3.2.   Mapping by Distance

The main part of the proposed algorithm is to calculate distances between vectors in the feature space and map close two vectors among two input tables as correspondent fields. The pseudo code for the algorithm is illustrated in Fig. 2.

```
procedure TableIntegrate(T₁, T₂: tables, C: code chart)
begin
  foreach f₁ ∈ F(T₁)
  do
      v₁ := fieldval(f₁, C)
      foreach f₂ ∈ F(T₂)
      do
          v₂ := fieldval(f₂, C)
          d := distance between v₁ and v₂
          h(v₁, v₂) := d
      end
  end
  l := sorted list of all pairs (v₁, v₂)
      by d(= h(v₁, v₂)) in increasing order
  for (v₁, v₂) ∈ l
      report (f₁, f₂) are correspondent
      delete (p′₁, p′₂) ∈ l if p′₁ = v₁ or p′₂ = v₂
  end
end
```

Figure 2: Pseudo code of the algorithm

For each field f_1 of T_1, the algorithm calculate the Euclidean distance between f_1 and f_2 for any f_2 of T_2, and then selects iteratively a pair (f_1, f_2) providing the smallest distance among all pairs of fields.

As the similarity of vectors, the cosine value of the vectors. If the vecoters are nomalized, we can see that the cosine value are equivalent to the Euclidian distance. But, by the preliminari experiments, normalization of vectors decrease accuracy values for matching fields. Thus, we do not adopt nomalization.

Let N be the maximum number of attribute values for a field, n be the maximum length of an attribute value, and $m = \max\{|F(T_1)|, |F(T_2)|\}$. Then, fieldval() runs in $O(nN)$ time. It takes $O(m^2)$ time to calculate distances for all pairs and $(m^2 \log m)$ time to sort all distances. Therefore, the algorithm totally runs in $O(nN + m^2 \log m)$ time.

4.   Experiments

We implement the algorithm described in the previous section in Python and apply it to tables written in various languages. Table 3 lists all datasets with brief descriptions, where the fourth and fifth columns show the number of instances and fields in dataset, respectively.

Table 3: Datasets used for experiments

ID	Description	Language	# instances	# fields
1	Search results from 4 major search engines.	English	4721	3
2	Search results Yahoo! Korea.	Korean	671	4
3	Search results Yahoo! China.	Chinese	1773	3
4	XML database of academics in Kyushu univ.	Japanese	497	6
5	Yellow Pages.	Japanese	1000	3
6	Yahoo! Auction Korea.	Korean	2908	4
7	List of tennis players.	German	227	13

For the implemented program, tables written in any language are applicable if they are encoded in UTF-8[7].

4.1. Search Engines

Dataset 1 are collected from 4 major search engines, AltaVista[8], Excite[9], Lycos[10], and Yahoo![11]. A page produced by a search engine contains multiple instances. An instance consists of titles of Web pages, URLs, and short descriptions for the pages. Therefore, the result of search result looks like a table containing 3 fields. From each of 4 search engines, we gather about 1,000 search results (about 100 pages), and totally 4721 record instances are obtained.

We collected 410 category names in advance from Yahoo! and the pages linked from the top page. We use these names as query words. When we gathered search results, we randomly chose keywords from the category names.

For any combination of 4 search engines, we made a series of experiments as the number of instances are increasing. We evaluate the experiments by accuracy, that is, the number of matched field to the number all fields. The total number of fields is 18 since there are 6 combinations and each dataset has 3 fields. Table 4 shows all accuracy

Table 4: Accuracy for integration tables among any two search engines in Dataset 1

size	accuracy	size	accuracy
10	16/18	60	18/18
20	14/18	70	18/18
30	16/18	80	18/18
40	18/18	90	16/18
50	18/18	100	18/18

values, where "size" denotes the the number of files. Basically, a file in this dataset contains 10 found pages, so 10 in the size column means that 100 instances are included for both search engines of a combination.

---

[7] This is an character encoding scheme of Unicode.
[8] http://www.altavista.com/
[9] http://www.excite.com/
[10] http://www.lycos.com/
[11] http://www.yahoo.com/

The algorithm fails to match fields when the size is small and it almost succeed for larger sizes. However, when the size is 90, the algorithm fails to match between URL and title fields. Apparently, the difference between URLs and page titles is easy for us because most URLs of found pages begin with "http://." The algorithm folds all character codes of a string into one value when it calculates a feature value. This means that the algorithm ignores any properties of substrings appearance and their appearance orders. Nevertheless, the algorithm works very well for many data sizes.

This dataset does not show any difference between 2 and 3 dimensional feature spaces.

For Dataset 1, we integrate wrapper tables created from different sites. We need to see source files and define the correct mapping in advance. However, it is impossible to construct such correct mapping for multilingual inputs. So, in the sequel, we use two tables whose instances are randomly selected from the same table. In this case, it is easy to check the correctness.

Dataset 2 and 3 are also search result pages gathered from Yahoo! Korea[12] and China[13], respectively. Both datasets have the same 3 fields of Dataset 1. Dataset 2 has another field[14].

Table 5 shows accuracy values as the number of instances are increasing, where "size" is the number of files. Basically, each file contains 20 instances. Each size, we

Table 5: Accuracy for integration tables of Dataset 2

size	accuracy	size	accuracy
1	11/20	6	20/20
2	14/20	7	20/20
3	18/20	8	20/20
4	20/20	9	20/20
5	20/20	10	20/20

size	accuracy	size	accuracy
1	8/20	6	20/20
2	12/20	7	20/20
3	20/20	8	20/20
4	20/20	9	20/20
5	18/20	10	20/20

do experiments iteratively 5 times[15]. Now, accuracy is defined to be $m/(N \times 5)$, where m is the number of matched fields during all 5 times iterations and N is the number of fields in the dataset.

The above table of Table 5 is the result when the algorithm uses 2 dimensional feature space and the bellow one for 3 dimensional case.

For both feature spaces, the algorithm increases accuracy values and achieves 100%. For this dataset, the new added feature slow the converge speed.

For Dataset 3, the algorithm shows 100% accuracy if the input file size is greater

---

[12] http://kr.search.yahoo.com/
[13] http://cn.search.yahoo.com/
[14] But, the author does not know what the field is.
[15] In the sequel, we make experiments in a similar manner for other datasets.

D. IKEDA

than 3 if in the 2D feature space (see the above table of Table 6) and 2 in the 3D feature space (see the below of Table 6).

Table 6: Accuracy for integration tables of Dataset 3

size	accuracy	size	accuracy
1	12/15	6	15/15
2	13/15	7	15/15
3	15/15	8	15/15
4	15/15	9	15/15
5	15/15	10	15/15

size	accuracy	size	accuracy
1	12/15	6	15/15
2	15/15	7	15/15
3	15/15	8	15/15
4	15/15	9	15/15
5	15/15	10	15/15

Table 7 shows all points for fields in two tables randomly extracted from Dataset 3 when the file is 10.

Table 7: Points and distances for Dataset 3

Field	Point	
URL =	(2618.95,	948.87)
Summary =	(582382.92,	222902.47)
Title =	(207781.26,	114893.04)
URL =	(2595.70,	1018.10)
Summary =	(585820.57,	266668.74)
Title =	(225597.73,	104127.19)

4.2. XML Database of Academics

Dataset 4 consists of 497 XML files, each of which is a record instance of an academic in Kyushu University. A record contains "ID" which is a unique number, email address, phone, URL, job title, academic degree, and so on. Basically, contents are written in Japanese. An ID number is automatically generated and a job title is selected from options, so data in these two fields are clean. On the other hand, other fields are basically filled by hand. There are some blank fields, irregular data, and multiple values.

Important differences from Dataset 1 are that contents are basically written in Japanese and selective fields exist such as the job title. Table 8 shows accuracy values as the number of instances are changing, where "size" is the number of instances.

As the number of instances are increasing, the accuracy are increasing and converges to the perfect when the number of instances is 50.

In Dataset 4, several outliers of feature values are found. They are included in the fields of email address and phone number. Although almost all attribute values in these fields are written in the ASCII code, few of them are written in double-byte

Table 8: Accuracy for integration tables with the same number of fields extracted from Dataset 4

size	accuracy	size	accuracy
10	20/30	60	30/30
20	26/30	70	30/30
30	28/30	80	30/30
40	28/30	90	30/30
50	30/30	100	30/30

characters, like "d a i s u k e [16]." The feature value, for example, for "daisuke" is $100 + 97 + 105 + 115 + 117 + 107 + 101 = 742$[17], while one for "d a i s u k e" is $65348 + 65345 + 65353 + 65363 + 65365 + 65355 + 65349 = 457478$ in Unicode. The values for such outliers are very huge. Thus, both the average and the standard deviation heavily affected by these outliers. In the above experiments, we use data cleaned by hand. We need some algorithm that eliminates automatically such outliers.

We compare accuracy values between 2 and 3 dimensional feature spaces (see Table 9). The convergence size is a bit smaller than 2D case. But, on the whole, the

Table 9: Experiments using 3 dimensional feature space on Dataset 4

size	accuracy	size	accuracy
10	26/30	60	30/30
20	26/30	70	30/30
30	28/30	80	30/30
40	30/30	90	30/30
50	30/30	100	30/30

dimensionality does not affect accuracy values.

4.3.   Yellow Pages

Dataset 5 are HTML files each of which contains 20 instances. A record instance has the company name in Japanese, its phone number, and its address in Japanese. We gather randomly 50 files from Yahoo! PHONEBOOK[18] and totally this dataset has 1,000 instances. The accuracy converges to 100% after the number of instances is just 20 and after that the algorithm keeps the accuracy value as 100%.

4.4.   Auction Lists

Dataset 6 are HTML files gathered from the auction site of Yahoo! Korea[19]. An instance consists of 4 fields[20].

---

[16] For all alphabets and digits in the ASCII code, we have correspondent double-byte characters in Japanese major character code charts.
[17] A code number is in decimal.
[18] http://phonebook.yahoo.co.jp/
[19] http://kr.list.auctions.yahoo.com/
[20] The author does not read Korean Hangeul sentences, but maybe, they are "Title," "Price," "Bids," and "Time Left" like http://list.auctions.shopping.yahoo.com/.
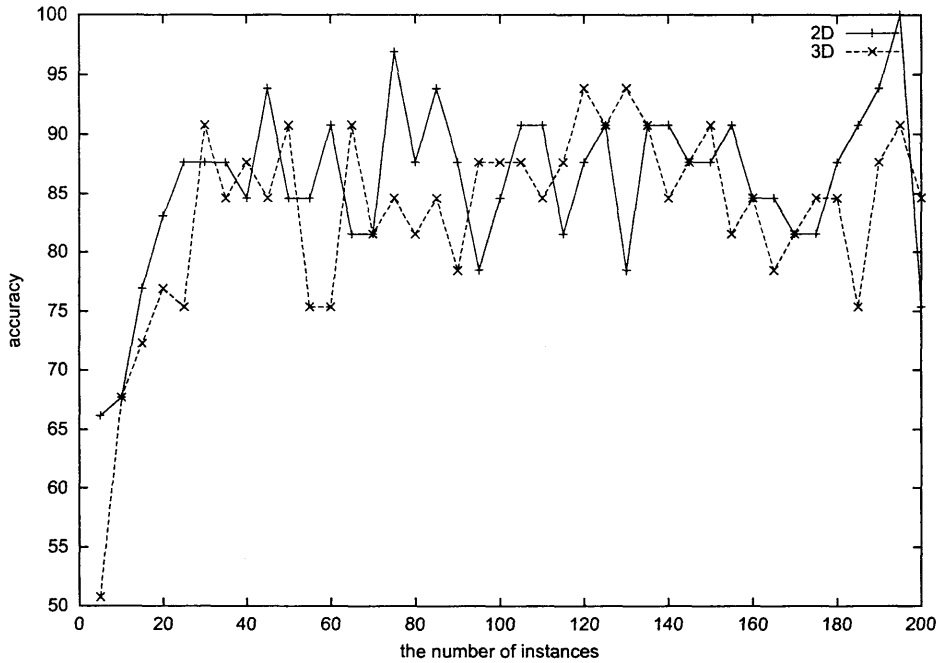
Figure 3: Accuracy for integration tables extracted from Dataset 7

For most cases, less than 50 instances are enough for the algorithm to achieve 100% accuracy.

Tables used in experiments have simple and flat structures. The simplicity does not decrease the difficulties to integrate tables because they do not have any schema and attribute names, and contents are written in many languages. Instead of schema and attribute names, presented algorithm utilized only attribute values. This shows that it is applicable for tables on the Web, especially those created by wrapper (semi-)automatically.

Obviously, there are tables which are not available for the presented algorithm. In fact, the algorithm failed to find the correct mapping for some tables. There are two types of such failures including potential ones: One is the case that seems to be impossible for instance based algorithms. Consider two fields, one contains phone numbers and the other do fax numbers. For us human beings, it is impossible to distinguish them without attribute names.

The other one is the case that would be improved by adding other features, that is increasing the dimensionality, or using other methods. Especially, we can expect that to find common substrings in a field would drastically improve the accuracy since our representation ignores the order of characters in a string. For a string s and its shuffled strings, they are transformed into the same integer. Thus, the proposed algorithm can not distinguish these strings. However, we must be cautious to add ad-hoc functions to the algorithm because to find common substrings or patterns is one of the main target of the Web mining and requires high computational costs. Instead, it is an interesting and important future work to find another numerical future that preserves (partially)

information of the characters' order.

In the current implementation, the algorithm always maps fields even if two tables have nothing to do with each other. Therefore, another important future work is to find a good threshold for distances to refute some correspondence of fields if they do not contain the same data.

References

Ashish, N. and Knoblock, C. A. (1997).   Wrapper Generation for Semi-structured Internet Sources, *SIGMOD Record*, Vol. 26, No. 4, pp. 8–15.

Chidlovskii, B. (2002).   Schema Extraction from XML Collections, In *Proceedings of the second ACM/IEEE-CS joint conference on Digital libraries*, pp. 291–292.

Cohen, W. W. (1998).   Integration of Heterogeneous Databases without Common Domains Using Queries Based on Textual Similarity, In *Proceedings of the 1998 ACM SIGMOD International Conference on Management of Data*, pp. 201–212.

Doan, A., Domingos, P. and Halevy, A. Y. (2001).   Reconciling Schemas of Disparate Data Sources: A Machine Learning Approach, In *Proceedings of the 2001 ACM SIGMOD International Conference on Management of Data*, pp. 509–520.

Genesereth, M. R., Keller, A. M. and Duschka, O. (1997).   Infomaster: An Information Integration System, In *Proceedings of the 1997 ACM SIGMOD International Conference on Management of Data*, pp. 539–542.

Knoblock, C. A., Minton, S., Ambit, J., et al. (1998).   Modeling Web Sources for Information Integration, In *Proceedings of the 15th National Conference on Artificial Intelligence*, pp. 211–218.

Kushmerick, N., Weld, D. S. and Doorenbos, R. B. (1997).   Wrapper Induction for Information Extraction, In *Proceedings of the 15th Intenational Joint Conference on Artificial Intelligence*, pp. 729–737.

Kushmerick, N. (1999).   Regression Testing for Wrapper Maintenance, In *Proceedings of the 16th National Conference on Artificial Intelligence (AAAI-99)*, pp. 74–79.

Lerman, K., Knoblock, C. A. and Minton, S. (2001).   Automatic Data Extraction from Lists and Tables in Web Sources, Adaptive Text Extraction and Mining Workshop.

Lerman, K. and Minton, S. (2000).   Learning the Common Structure of Data,   In *Proceedings of the 17th National Conference on Artificial Intelligence (AAAI-2000)*, pp. 26–30.

Levy, A., Rajaraman, A. and Ordille, J. (1996).   Querying Heterogeneous Information Sources Using Source Descriptions, In *Proceedings of the 22nd International Conference on Very Large Data Bases (VLDB)*, pp. 251–262.

Li, W. -S. and Clifton, C. W. (2000).   SEMINT: A Tool for Identifying Attribute Correspondences in Heterogeneous Databases Using Neural Networks, *Data and Knowledge Engineering*, Vol. 33, No. 1, pp. 49–84.

Ma, L., Shepherd, J. and Nguyen, A. (2003).   Document Classification via Structure Synopses, In *Proceedings of the 14th Australasian Database Conference*, pp. 59–65.

Madhavan, J., Bernstein, P. and Rahm, E. (2001).   Generic Schema Matching with Cu-

pid, In *Proceedings of the 27th International Conference on Very Large Data Bases (VLDB)*, pp. 49–58.

Miller, R. J., Haas, L. M. and Hernández, M. A. (2000). Schema Mapping as Query Discovery, In *Proceedings of the 26th International Conference on Very Large Data Bases (VLDB)*, pp. 77–88.

Monge, A. E. and Elkan, C. P. (1996). The Field Matching Problem: Algorithms and Applications In *Proceedings of the Second International Conference on Knowledge Discovery and Data Mining*, pp. 267–270.

Popa, L., Velegrakis, Y., Miller, R. J., Hernandez, M. A. and Fagin, R. (2002). Translating Web Data, In *Proceedings of the 28th International Conference on Very Large Data Bases (VLDB)*, pp. 598–609.

Yamada, Y., Ikeda, D. and Hirokawa, S. (2001). SCOOP: A Record Extractor without Knowledge on Input, In *Proceedings of the Fourth International Conference on Discovery Science*, Lecture Notes in Artificial Intelligence, Vol. 2226, pp. 428–487.

Yamada, Y., Ikeda, D. and Hirokawa, S. (2002). Automatic Wrapper Generation for Multilingual Web Resources, In *Proceedings of the 5th International Conference on Discovery Science*, Lecture Notes in Computer Science, Vol. 2534, pp. 332–339.

Yoshida, M., Torisawa, K. and Tsujii, J. (2001). A Method to Integrate Tables of the World Wide Web, In *Proceedings of the 1st International Workshop on Web Document Analysis (WDA 2001)*, pp. 31–34.

Yoshida, M. (2002). Extracting Attributes and Their Values from Web Pages, In *Proceedings of the ACL-02 Student Research Workshop*, pp. 72–77.