Extracting ontologies from World Wide Web via HTML tables

Minoru Yoshida[1], **Kentaro Torisawa**[2,3] **and Jun'ichi Tsujii**[1,4]

[1] Department of Computer Science, Graduate school of Information Science and Technology,

[2] School of Information Science, Japan Advanced Institute of Science and Technology

[3] Information and Human Behavior, PRESTO, Japan Science and Technology Corporation

[4]CREST, JST(Japan Science and Technology Corporation)

Postal address:

Department of Computer Science, Graduate school of Information Science and Technology, University of Tokyo, 7-3-1 Hongo, Bunkyo-ku, Tokyo, 113-0033, Japan

Telephone: +81 3 5803 1697 Facsimile: +81 3 5802 8872

{mino, tsujii}@is.s.u-tokyo.ac.jp, torisawa@jaist.ac.jp

Summary

This paper describes a method to extract ontologies from tables in the World Wide Web (WWW). A table can be seen as a device to describe relating objects by attribute-value pairs. The attributes specify the important information that we need to know for *identification* and *utilization* of the described objects. This property is the same as the requirement on generic ontologies. So, by properly processing a wide range of tables, we can construct ontologies. We proposed an unsupervised method for this task. The method utilizes the EM algorithm and can be seen as an unsupervised learning method. The effectiveness of our method is confirmed by a series of experiments.

keywords: WWW, tables, EM algorithm, ontology

1 Introduction

This paper describes a method to extract ontologies from tables in the World Wide Web (WWW). A table can be seen as a device to describe relating objects by attribute-value pairs. The attributes specify the important information that we *need to know* for *identification* and *utilization* of the described objects. For example, we can identify each CD by its values for the “Title”, “Composer” and “Price” attributes, and can utilize the same information for certain purposes.

This property of attributes is the same as the requirement on generic ontologies. More precisely, ontologies, or some part of them, can be described by attribute-value pairs as exemplified in Figure 2 and those attributes express what we *need to know* for identification and utilization of the described class of objects. So, by properly processing a wide range of tables, we can construct ontologies.

Title	Composer	Price
Sonata No. 1	Mozart	2500
Symphony No. 9	Beethoven	1800
Nocturne No. 1	J.S.Bach	1500

Figure 1: A Table Describing CDs

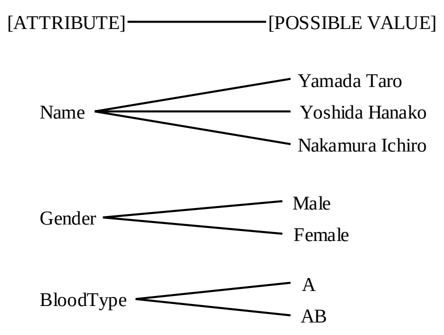


Figure 2: A Sample Ontology of Human

The proposed method classifies the tables according to the objects described by each table,

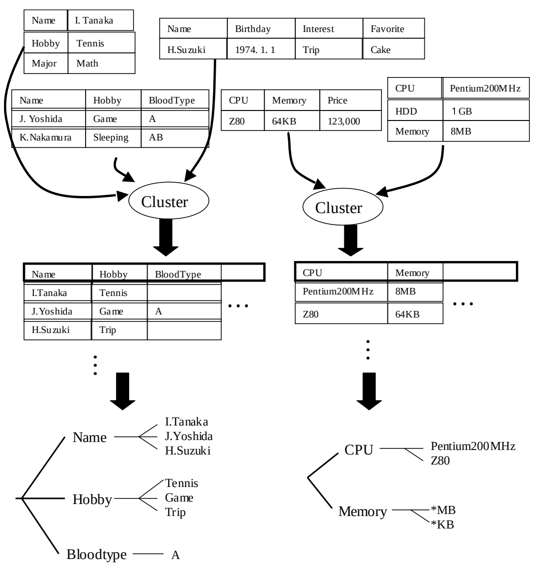


Figure 3: Sample Extraction of Ontologies

and collects the attributes and their possible values from the tables describing a class of objects which is illustrated in Figure 3.

Let us see the above processes more concretely using the example. In this case, a set of self-introduction tables and PC specification tables in various format are clustered into the two clusters. Tables in each cluster are merged into one large table called a *class table*. After that, attributes in each class table, such as "Name" and "Hobby" attributes in a human class table (made from self-introduction tables), are extracted as attributes for the corresponding ontology. Possible values of the attributes like "I.Tanaka" and "Tennis" are also extracted as the values of those attributes.

Note that there are three major problems in doing this task. The first is that extracting attribute-value pairs from a given table is not a trivial task. The second is that we have to develop a method to classify tables according to the objects described by the tables. The third is that there are much variations in the representations of the attributes that means almost the same. For instance, we can write "Date of Birth" instead of "Birthday" both of which have the same meaning. So, in order to construct useful ontologies from tables, we need to identify the set of expressions that have the same meaning. In this paper, we provide solutions for tackling these problems.

Our method consists of the following three processes:

- Table Structure Recognition (for the first problem)
- Table Clustering (for the second problem)
- Attribute Clustering (for the third problem)

where the *table structure* means the set of attributes and their possible values for a given table. In order to extract attribute-value pairs from tables in various forms, we first recognize table structures of tables since they are not given explicitly in web pages. We call this task *table structure recognition*. *Table clustering* classifies tables and *attribute clustering* finds the attributes in the same meaning.

Our method can be seen as a learning method.The method is unsupervised one in the sense that no human intervention and no manually tailored "table corpus" are required. The method utilizes the EM algorithm.

1.1 Table Structure Recognition

Table structure recognition detects which part of a given table is an attribute or a value. There have been some researches on this task. Most of them utilized surface features such as the number of numeric words, the length of strings (Hurst and Douglas, 1997) (Chen et al., 2000) or HTML tags (Itoh et al., 1999). Although these methods achieved significant success, we view this task in a different perspective. Table interpretation by human requires ontological knowledge, as stated above. For instance, if a table describes people, it should have attributes such as "Name", "Birthday" and "Hobby". In addition, there should be strings which are likely to be values of these attributes. We think that these insights are based on generic ontological knowledge. Our approach is to recover a part of such ontological knowledge from tables about various objects in various formats, and to use it in table recognition. Of course, previous works can be seen as special cases of this approach. The utilization of surface clues such as numbers can be regarded as an attempt to use ontological knowledge related to numbers. However, our approach treats larger classes of strings. As another type of study, a machine learning approach (Ng et al., 1999) or a model-based algorithm (Green and Krishnamoorthy, 1995) were proposed. But they focused on the tables in free texts or printed tables with visual cues such as the thickness of lines, not on HTML tables.

In this paper, we provide the algorithm to perform this task which doesn't require much prior-given knowledge. The algorithm is an unsupervised method in the sense that it does not require any *training samples* in which table structures are given by a human. This is achieved by utilizing the Expectation Maximization algorithm (Dempster et al., 1977) . The algorithm estimates the probabilities that a string appears as attributes (or values). After this estimation is performed we obtain a set of probabilities which tells us that the strings such as "Hobby" or "Birthday" has a high probability to be an attribute, and a number or a person's name has a high probability to be a value. Then, the algorithm determines a table structure according to the estimated probability distribution. Through a series of experiments we show that our algorithm achieved about 80% recognition accuracy.

1.2 Table Clustering

After obtaining table structures for many tables, our procedure gathers tables describing objects in the same class into a single cluster. In order to solve this problem, we define the concept of *unique attributes*, which are the attributes appearing particularly in some specific tables, such as the "Hobby" attribute in the tables about people. Our algorithm estimates the *uniqueness* for each attribute and uses the attribute with the high uniqueness for clustering.

1.3 Attribute Clustering

Different attributes can be used with the same meaning, such as the attributes "Birthday" and "Day of birth" in self-introduction tables. To align such attributes we need a method for *attribute clustering*, which classifies attributes. We define a similarity measure between each pair of attributes in terms of the frequency of values appearing with each attribute and cluster them based on this similarity measure. After that, we obtain the set of table clusters each of which is represented by a single large table called a *class table*, as shown in Figure 3. Finally, each class table is mapped to the corresponding ontology.

Name	John	Mary
Gender	male	female
BloodType	A	AB

(a) A table representing human beings

Name	Tel.	Recommendation
Lake Restaurant	31-2456	special lunch
Cafe Bonne	12-3456	chicken curry
Metro Restaurant	56-7890	fried rice

(b)A table representing restaurants

John	Richard	Tom
Jude	Mary	Bill

(c)A list of names

Figure 4: Sample Tables

1.4 Outline

In Section 2 of this paper we describe our algorithm for table structure recognition. In Section 3 we will provide our table clustering algorithm. In Section 4 we will describe our attribute clustering algorithm. In Section 5 we will show the results of our experiments in which tables extracted from the WWW were used.

2 Table Structure Recognition

Given a set of tables, our table structure recognition algorithm assigns a table structure, which denotes the positions of attributes and those of values, to each table in the set. For example, for table (b) in Figure 4, the desired output is the table structure which places the attributes on the line of the first row.

First of all, we give definitions of tables and table structures. We begin by assuming that a table is a two-dimensional array of strings. For our convenience, we represent a table by a sequence of its items, along with the numbers of its rows and columns. Therefore, a table T is denoted as

$$T = (\langle s_1, s_2, ..., s_{xy} \rangle, x, y)$$

where x is the number of columns in T and y is the number of rows in T. $\langle s_1, s_2, ..., s_{xy} \rangle$ is a sequence of strings appearing in the table. We assume that the position of a string s in a sequence $\langle s_1, ., s, ., s_{xy} \rangle$ has a one-to-one correspondence with the position in the table. In other words, if we specify a position in a table, we can retrieve the string appearing in the position from the sequence.

We assume that each string in the table can be classified as an *attribute string* or a *value string*. In table (a) of Figure 4, the strings "Name", "Gender" and "BloodType" are attribute strings. "John", "Male", "A", etc. are value strings of these attributes. Note that an attribute string in one table can be a value string in another table. The table structure represents the layout of attribute strings and value strings in a table. It is a set of *labels*, which are assigned to each string of the table, where a label is a member of the *set of labels* $\{att, val\}$. The *att* label stands for *attributes* and the *val* label stands for *values*.

More formally, the table structure is defined as the function whose argument is a table

$$(\langle s_1, s_2, ..., s_n \rangle, x, y)$$

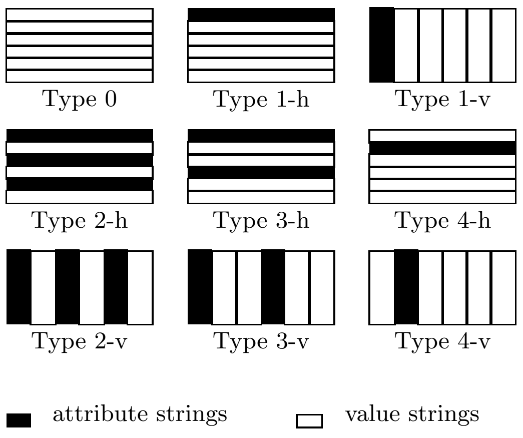
and has a value of the corresponding sequence

$$\langle (s_1, l_1), (s_2, l_2), ..., (s_n, l_n) \rangle$$

where l_i is the label that corresponds to the string s_i.

We also assume that table structures are categorized into the nine *types* illustrated in Figure 5. Figure 6 provides examples of tables using some of the types. When table (a) in Figure 6 has the table structure 1-h, *att* is assigned to the words "Title", "Composer" and "Price", while *val* is assigned to "Mozart", "Beethoven", and so on.

Note that if we are given the size and a type of a table, we can obtain a unique table structure. Since a table size is always given, we can regard a *type* as a table structure. Another important point is that the nine types in Figure 5 are corresponding to only *plausible* table structures, not all the possible table structures. This has the effect of preventing our algorithm from losing its way among various implausible table structures. Actually, we use types instead of table structures in the remaining of the paper for the sake of simplicity.



Suffix h: the attribute words are arranged horizontally.
Suffix v: the attribute words are arranged vertically.

Figure 5: Types

[Title]	[Composer]	[Price]
Sonata No. 1	Mozart	2500
Symphony No. 9	Beethoven	1800
Nocturne No. 1	J.S.Bach	1500

(a) Type 1-h

[Name]	Hanako	[BloodType]	A
[Gender]	Female	[Birthday]	22 Feb.
[Nationality]	Japanese	[Tel.]	12-3456

(b) Type 2-v

Cafe Hongo	
[Item]	[Price]
Coffee	300 Yen
Curry Rice	700 Yen
Toast	350 Yen

(c) Type 4-h

Figure 6: Examples of Tables with Various Types. Bracketed Strings Represent Attributes.

2.1 Algorithm

The algorithm chooses the most plausible sequence of types $\mathcal{M} = \langle m_1, m_2, ..., m_n \rangle$ for input sequence of tables $\mathcal{T} = \langle T_1, \cdots, T_n \rangle$, according to the estimated probabilities as follows.

$$\begin{aligned}
\mathcal{M} &= \arg\max_{\mathcal{M}} P(\mathcal{M}|\mathcal{T}) \\
&= \arg\max_{\mathcal{M}} P(\mathcal{M}, \mathcal{T}) \\
&= \arg \max_{\langle m_i \rangle_{i=1}^n \in \mathcal{M}} \prod_i P(m_i, T_i)
\end{aligned}$$

Then, we express the probability $P(m_i, T_i)$ with the follwoing parameter set θ and denotes the probability by $P_\theta(m_i, T_i)$.

$$\theta = \{P_\theta(m|x,y)\} \cup \{P_\theta(s|l)\}$$

$$\begin{aligned}
P_\theta(m, T) &= P_\theta(m, \{\langle s_i \rangle_{i=1}^n, x, y\}) \\
&= P(x,y) P_\theta(m|x,y) P_\theta(\langle s_i \rangle_{i=1}^n | m, x, y) \\
&\approx P(x,y) P_\theta(m|x,y) P_\theta(\langle s_i \rangle_{i=1}^n | m) \\
&\approx P(x,y) P_\theta(m|x,y) \prod_{(s,l)\in m(T)} P_\theta(s|l)
\end{aligned}$$

Note that $P(x,y)$ is constant because it is not dependent on the value of $\mathcal{M}$. In the last transformation, we made an approximation such that $P_\theta(\langle s_i \rangle_{i=1}^n | m) = \prod_{(s,l)\in m(T)} P_\theta(s|l)$. Roughly, this says that $P_\theta(\langle s_i \rangle_{i=1}^n | m)$ is the product of $P(s|l)$ for all the pairs (s,l) in $m(T)$ where s is a string and l is a label.

The EM algorithm improves the value of $P_\theta(\mathcal{M}, \mathcal{T})$ by repeatedly adjusting the parameter set θ. The algorithm outputs $\mathcal{M}$ which locally maximizes the value of $\log P_\theta(\mathcal{M}, \mathcal{T})$ as the best sequence of types. At each iteration, the parameters are adjusted according to the following formulae. These formulae are easily derived according to the standard derivation steps in the EM algorithm:

$$P_\theta(m|x,y) = \frac{1}{|\mathcal{T}_{xy}|} \sum_{T \in \mathcal{T}_{xy}} P_{\theta'}(m|T)$$

$$P_\theta(s|l) = \frac{1}{Z_{\theta'}(l)} \cdot \sum_{i,k} \sum_{k(m(T_i))=(s,l)} P_{\theta'}(m|T_i) \quad (1)$$

where

$$P_{\theta'}(m|T_i) = \frac{P_{\theta'}(m, T_i)}{P_{\theta'}(T_i)}$$

Here $k(m(T))$ means the kth element of the ordered set $m(T)$ and $\sum_{k(m(T_i))=(s,l)}$ means the summation over all possible values of m such that the kth element of $m(T)$ is (s,l). $Z_{\theta'}(l)$ is the normalizing factor such that $\sum_s P(s|l) = 1$. $\mathcal{T}_{xy}$ is a sequence of tables with the size of (x,y).

2.2 Other elements of the algorithm

Some further elements of our algorithm are described in this subsection.

Final Word Let us describe our use of final words to canonicalize each string of all tables. We assume that all strings s in a table are noun phrases. We remove all words from s other than the head nouns, and thus generalize the meaning of s (i.e. to avoid problems with data sparseness.)

In Japanese, the *final word* of a noun phrase is often the head noun. For example, "関数名 (Function Name)" is generalized to "名 (Name)" as this is the final word of "関数名". This assigns "関数名" a high frequency, even if the frequency of the appearance of this particular phrase is low. We therefore use the final word from each string s in place of s.

Threshold We cluster the final words with a frequency which is lower than some threshold (currently 20). Such strings are treated as one string.

Initial Parameter Finally let us explain how to give the initial parameter set θ_0. The only parameters to which we manually assign values are the $P(m)$ for all values of m. (This $P(m)$ is equal to $P(m|x,y)$ for all (x,y)). Note that because the number of kinds of m is 9, we have to give only 9 parameters a priori. After that, the following formula, which is obtained by substituting $P_{\theta_0}(m)$ for $P(m|T_i)$ in the formula (1), is used to estimate each $P(s|l)$.

$$P_{\theta_0}(s|l) = \frac{1}{Z_{\theta_0}(l)} \cdot \sum_{i,k} \sum_{k(m(T_i))=(s,l)} P_{\theta_0}(m)$$

Currently $P_{\theta_0}(m)$ is set to 0.3 if m is a Type 0, 1-h or 1-v, and $P_{\theta_0}(m)$ is set to 0.03 if m is another type in the parameter θ_0. These parameters are set to realize $P(0) : P(\text{1-h}) : P(\text{1-v}) =$

1 : 1 : 1 and $P(0) : P(m) = 10 : 1$ for another m. This setting is based on the fact that type 2–4 tables are rare in comparison with tables of the type 1-h, 1-v or 0.

Each $P_{\theta_0}(x, y)$ is given as follows and not changed in the algorithm.

$$P_{\theta_0}(x, y) = \frac{|\mathcal{T}_{xy}|}{|\mathcal{T}|}$$

3 Table Clustering

As mentioned before, we use *unique attributes* for clustering tables. The unique attributes are attributes *peculiar* to certain class of objects (or tables describing certain class of objects.) For instance, the attribute "Hobby" is peculiar to a table about self-introduction and the attribute CPU is peculiar to catalogues of personal computers. We express the degree of peculiarity of an attribute a by the function $uniq(a)$ defined below. Assume that we are given a set of tables R and the set A of all the attributes appearing in R. $U(a)$ is a set of tables in R in which a appears. $V(a)$ is the set of attributes appearing in $U(a)$. $Freq(b, a)$ is the frequency of attribute b in the tables in $U(a)$.

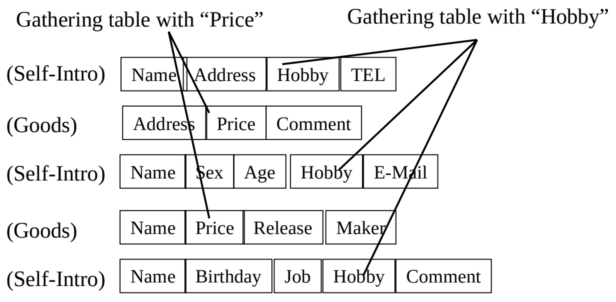
$$uniq(a) =_{def} cooc(a) \cdot excl(a)$$

where

$$cooc(a) =_{def} \frac{1}{|V(a)|} \sum_{b \in V(a)} Freq(b, a)$$

$$excl(a) =_{def} \frac{1}{|V(a)|} \sum_{b \in V(a)} \frac{Freq(b, a)}{|U(b)|}$$

Intuitively, if $uniq(a)$ is large, $U(a)$ is likely to be a set of tables describing similar objects. $cooc(a)$ expresses how consistently the attributes in $V(a)$ co-occurs with a. When $V(a)$ consists of only the attributes which rarely co-occurs with a, then a is considered to appear in R rather randomly. This means that a is not peculiar to tables describing certain consistent class of objects. On the other hands, $excl(a)$ represents the degree of exclusiveness of the attributes in $V(a)$. If attributes in $V(a)$ appear



- "Hobby" and "Price" are taken as the unique attributes.
- Note that only attributes of each table are indicated.

Figure 7: Clustering Example

frequently in $U(a)$ and rarely in $R - U(a)$, then $excl(a)$ has a large value.

The algorithm selects an attribute a with a large value of $uniq(a)$ and takes $U(a)$ as a cluster. In the example in Figure 5, self-introduction tables are gathered by the unique attribute "Hobby" and goods tables are gathered by the unique attribute "Price". On the other hand, "Comment" is assigned a low value of *uniq* and not used for clustering because if $a =$"Comment", all other attributes in $V(a)$ ("Address", "Birthday", and so on) appear only once in $U(a)$ and frequently appears in $R{-}U(a)$.

4 Attribute Clustering

Attribute clustering classifies similar attributes into the same cluster. Attributes in the same cluster are aligned as the same attribute in a class table. The similarity between attributes a and b is calculated using a simple cosine measure where each attribute is represented by a vector whose element is the frequency of each value appearing with that attribute.

5 Experiments

5.1 Table Structure Recognition

We implemented our algorithm and applied it to S, a set of HTML tables gathered from WWW. S contained 35232 tables. Most of these tables were in Japanese.

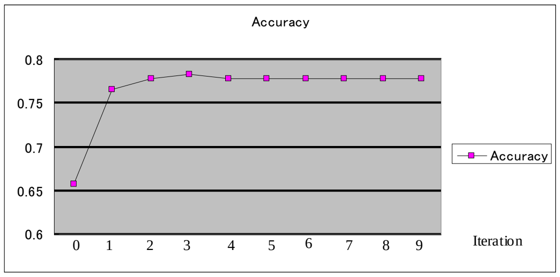


Figure 8: Improvement of Accuracy with Iteration

After the parameters were estimated from all over S, we estimated the accuracy with which types were assigned to tables by randomly selecting 175 tables from S and applying the algorithm to them. Note that therefore accuracy was evaluated in a closed test. The degree of accuracy was calculated as

$$\frac{n}{175}$$

where n is the number of tables to which correct types were assigned. Of these 175 tables, the number of tables with type 0, 1-h, 1-v and others, were 76, 61, 35 and 3, respectively.

Figure 8 shows the accuracy on each iteration. The accuracy increased from 0.66 to 0.78. Note that this accuracy is significant if we consider the distribution of types.

Table 1 shows the more detailed results, with figures for recall and precision. The recall of a type m is defined as

$$\frac{N_c(m)}{N_h(m)}$$

where $N_h(m)$ denotes the number of tables assigned type m by a human. The precision of a type m is defined as

$$\frac{N_c(m)}{N_m(m)}$$

where $N_m(m)$ denotes the number of tables assigned type m by our algorithm, and $N_c(m)$ denotes the number of tables assigned the type m by both the human and our algorithm. We can see that the recall for Type 0 was dramatically improved by iteration. This means that iteration has a good effect in terms of discarding two-dimensional arrays of strings which do not contain attribute words.

Iteration=0			
Type	N_h	Correct answer	Recall
0	76	32	0.42
1-h	61	52	0.85
1-v	35	31	0.89
2–4	3	0	0.00
Total	175	115	0.66
Iteration=0			
Type	N_m	Correct answer	Precision
0	41	32	0.78
1-h	69	52	0.75
1-v	64	31	0.48
2–4	1	0	0.00
Total	175	114	0.66
Iteration=10			
Type	N_h	Correct answer	Recall
0	76	54	0.71
1-h	61	53	0.87
1-v	35	28	0.80
2–4	3	1	0.33
Total	175	136	0.78
Iteration=10			
Type	N_m	Correct answer	Precision
0	68	54	0.79
1-h	60	53	0.88
1-v	44	28	0.64
2–4	3	1	0.33
Total	175	136	0.78

Table 1: Results with recall and precision

Iteration=0			
	Number	Correct answer	
Recall	15	5	0.33
Precision	7	5	0.71
F-measure			0.45
Iteration=10			
	Number	Correct answer	
Recall	15	10	0.67
Precision	20	10	0.50
F-measure			0.57

Table 2: Results for Type 2–4

Type 2–4 only applied to one table, so we cannot properly estimate the recall and precision for Type 2–4 from Table 1. To estimate the recall and precision for Type 2–4, we investigated results for a larger set of 4033 tables, including 15 tables with Type 2–4. The result is shown in Table 2. This result shows that our algorithm can correctly assign Type 2–4 to a limited extent, although the F-measure is less than that for Type 1-h or Type 1-v.

Next we compared the performance of our

Cluster	Objects	# of Tables
趣味 (Hobby)	Person(10)	811
名前 (Person's Name)	Person(10)	668
住所 (Address)	Building(10)	271
所在地 (Address)	Building(7), Noted Place(3)	243
氏名 (First/Last Name)	Person(8), Univ.-Course(1), Record(1)	425
順位 (Rank)	Ranking(10)	351
タイトル (Title)	Game(5), Movie(3), Song(2)	623
内容 (Contents)	TV-Program(7), Lecture(1), Rugby Game(1), Schedule(1)	292
NAME	Person(10)	159
日時 (Date/Time)	Schedule(8), Record(2)	253
月日 (Date)	Schedule(5), Record(5)	149
全長 (Length)	Car(6), Machine(2), Pig(1), Fishing Rod(1)	41
CPU	PC(10)	95
なまえ (Person's Name)	Person(10)	25
資本金 (Capital Money)	Company(10)	40

Table 3: Result of Table Clustering

algorithm with the algorithm of (Chen et al., 2000). They reported that their algorithm filtered out non-tables (i.e. Type 0 tables) with a precision of 92.92% and a recall of 80.07% . According to their criteria, the precision of our result was 79.44% and the recall was 85.86%.

Although a precise comparison is not possible, it would at first appear that our method does not perform as well as Chen's. However, Chen's experiments were performed on a set of tables selected from airline information pages. We therefore believe that these tables were more suitable for their method because these tables most likely contain numbers. We therefore expect that our method can outperform Chen's if the performances are evaluated on a set containing a greater variety of tables.

5.2 Table Clustering

Table clustering was evaluated on a set of 44691 tables, which is larger than S.

We selected clusters with the top 15 values of $|V(a)|$, assuming that if $|V(a)|$ is large it means that tables in that cluster came from various kinds of locations and were therefore appropriate for evaluation. Thus we can avoid the cluster which contains many similar tables written by only one or a few persons. We investigated the clustering result by checking 10 randomly-selected objects in each cluster . Table 3 shows the clustering result. We can observe that some attributes, such as "Hobby", "CPU" and "Capital Money", which are we think appropriate as unique attributes, were properly used for clustering. On the other hand, some attributes such as "Contents" and "Title" are ambiguous because it is hard to express the type of objects in the cluster by one word (as "People" in "Hobby" cluster.) Therefore there is still a room for improvement of clustering result by adjusting the definition of *uniq* function or relying on other clustering techniques.

5.3 Attribute Clustering

The performance of attribute clustering was evaluated by using the resulting class tables. The algorithm selects top 7 highest-occurring attributes to present class tables. We call them *main attributes* of each cluster and use them for evaluation. For example, main attributes in the "Hobby" cluster were Name, Birthday, Address, Bloodtype, Job, Hobby and Favorite-foods. We evaluated table merging for 3 clusters whose unique attributes were "Hobby", "CPU", "Capital Money."

First, we evaluated the accuracy of attribute clustering by checking if each attribute gathered as a main attribute is plausible or not. Table 4 shows the accuracy for main attributes. Here N_c means the number of gathered attributes and N_a means the number of correct attributes among them. Figure 9 shows the 23 attributes gathered as "Present Address" attribute in a human class. Of these attributes, 16 were really plausible as "Present Address". In this case, a precision was calculated as $N_c/N_a = 16/23$. Note that the recall was not evaluated in this experiment. Although attributes whose values have a standard expression such as "Bloodtype" or "Employee" were fairly gathered, the accu-

Human Class			
Attribute	N_a	N_c	Precision
Name	8	4	0.50
Birthday	27	9	0.33
Present Address	23	16	0.70
Bloodtype	15	14	0.93
Job	25	4	0.16
Hobby	1	1	1.00
Favorite Food	39	8	0.21
PC Class			
Attribute	N_a	N_c	Precision
CPU	1	1	1.00
Main Memory	7	5	0.71
Cache	3	1	0.33
HDD	5	3	0.60
VideoCard	5	4	0.80
OS	3	1	0.33
CD-ROM	4	4	1.00
Company Class			
Attribute	N_a	N_c	Precision
Established Date	6	5	0.83
Opening Date	5	3	0.60
President	2	1	0.50
Capital Money	1	1	1.00
Main Address	17	5	0.29
Business	2	1	0.50
# of Employees	5	5	1.00

Table 4: Result of Attribute Clustering: 1

住所地, 所在地, 住所※, 住んでるとこ, 巣穴, 原産地, すんでいるところ, 生まれた場所, 住んでいる所, すみか, 在住, 生息場所, 住処, 住んだ所, 住んでる所, 生まれ, お住まい (PresentAddress),
誕生地と生息地, 誕生地＆生息地, 原産地＆生息地, (Birthplace and PresentAddress),
都道府県 (Prefecture),
生年及び出生地 (Birthday and Birthplace),
出身地, (Birthplace),

Figure 9: Attributes Gathered as "Present Address"

racy for attributes with no such standard value expression was rather low.

Next, we randomly selected 10 objects[1] for each class table and checked if their values in the original table appeared in the merging result (recall), and if their values in the merging result are correct (precision), for each main attribute in the cluster. The result is shown in Table 5. Although currently experiments was done on rather small set of tables, we can see that the merging was done properly to some extent.

[1]All of the objects came from different tables.

Cluster	Precision	Recall	F-measure
趣味 (Hobby)	0.98	0.79	0.87
CPU	0.90	0.82	0.86
資本金 (Capital)	0.94	0.77	0.85

Table 5: Result of Attribute Clustering: 2

6 Conclusion

In this paper, we have presented a method to extract ontologies by integrating the tables describing objects in the same class. Our method integrates tables by utilizing the automatically recognized table structures on the basis of probabilistic models where parameters are estimated with no training samples. We showed that our algorithm produced ontologies of a person class or a PC class automatically. However, our algorithm still leave room for improvement. We must enhance each process in our algorithm, especially in table clustering and attribute clustering, to improve the quality of ontologies. We also intend to apply our method to the tables other than Japanese ones.

References

H. H. Chen, S. C. Tsai, and J. H. Tsai. 2000. Mining tables from large scale HTML texts. *18th International Conference on Computational Linguistics (COLING)*, pages 166–172.

A.P. Dempster, N.M. Laird, and D.B. Rubin. 1977. Maximum likelihood from incomplete data via the EM algorithm. *J. Royal Statistical Soc., B*, 39:1–38.

E. Green and M. Krishnamoorthy. 1995. Model-based analysis of printed tables. *In Proceeding of International Conference on Document Analysis and Recognition*, pages 214–217.

M. Hurst and S. Douglas. 1997. Layout and language: Preliminary investigations in recognizing the structure of tables. *Fourth International Conference on Document Analysis and Recognition (ICDAR)*, pages 1043–1047.

F. Itoh, N. Otani, T. Ueda, and Y. Ikeda. 1999. Two-way navigation system for the information space and the real space using extraction and integration of attribute ontologies. *Journal of the Japanese Society for Artificial Intelligence*, 14 (In Japanese):1001–1009.

H.T. Ng, C.Y. Lim, and J.L.T. Koo. 1999. Learning to recognize tables in free text. *In Proceeding of the 37th Annual Meeting of the Association for Computational Linguistics (ACL 99)*, pages 443–450.