TCS-TR-A-07-24

TCS Technical Report

Flexible Wrappers for Keyword-Related Information

by

Atsuyoshi Nakamura, Hiroyuki Hasegawa, Tomoya Saito, Mineichi Kudo

Division of Computer Science

Report Series A

February 13, 2007

Hokkaido University
Graduate School of
Information Science and Technology

Email: atsu@ist.hokudai.ac.jp

Phone: +81-011-706-6806
Fax: +81-011-706-7832

Abstract

We propose an approach for realizing flexible wrappers, the keyword-related target item extraction based on structure and contents. Our approach composed of four phases, the (flexible) structure-based extraction phase, the content-based filtering phase, the decision phase and the error correction phase. We show one implementation of our approach and report encouraging results on our preliminary experiments that indicate practical usefulness of our method for information exploration.

1 Introduction

It is easy to make a wrapper program that extracts city weather from a site providing weather information of fixed format. By virtue of wrapper induction techniques [5], without coding extraction rules, you can get such a wrapper by only labeling what you want to extract. Even without such labeling, you can display city weather in a table format by analyzing repetition structure [3] of the weather information page. However, you should not depend on a specific site too much because your wrapper could not extract information correctly if the format used in the weather site changed and also because weather of some cities may not appear in that site but may appear in another site. If you could create a wrapper that does not depend on specific sites but can extract target information from non-specific sites like sites found using keyword search, more information could be extracted more robustly using such a wrapper.

Most wrappers proposed so far are classified into either a class of structure-base wrappers or that of content-based wrappers. Structure-based wrappers [5, 3] extract lists of target items by matching of structures around target items. Generally, structure-based wrappers do not have flexibility and can extract information only from specific sites. Content-based wrappers [9] extract lists of target items by identifying those terms themselves using their contents (or attributes). Content-based wrappers have some flexibility but cannot extract information correctly from some structured pages.

In this paper, we propose an approach for realizing a flexible wrapper, which is the keyword-related target item extraction using the four-phase extraction based on structure and contents. In our approach, our wrapper extracts lists of only two items, a key item and target item. The key item extracted by our wrapper must contain a keyword inputted by a user. These simplification and restriction are necessary for enabling information extraction from arbitrary pages. In our four phase extraction, many candidates are generated first using flexible structure-based extraction, some of them are filtered out second using content-based filtering, then at most one candidate is selected for each key position by total evaluation, and finally correcting candidates by analyzing repetition structures. We also provide detailed description with respect to one implementation of our approach.

We conducted preliminary experiments on two different tasks, the review search for an inputted restaurant name and the restaurant name search for an inputted city name. Our experimental results on both tasks are encouraging ones and indicating practical usefulness of our method for information exploration.

2 Our Approach

2.1 Keyword-Related Target Item Extraction

A flexible wrapper we consider should extract information from any page, but flexibility may cause many garbage information to be extracted. Generally, a wrapper extracts a collection of related specific item lists, for example, a collection of (arbitrary restaurant name A, A's address, A's telephone number). Component items in each collected list appear in a variety of structures, and an ideal flexible wrapper is the one that can extract related item lists specified for the wrapper from a page of any structure.

Extraction task are decomposed to two subtasks, component item identification and recognition of related component items. In component item identification, a wrapper has to identify which parts of a web page are the component items specified for the wrapper. For example, a wrapper for list (arbitrary restaurant name A, A's address, A's telephone number) has to identify restaurant-name parts, address parts and telephone-number parts. In recognition of related component items, a wrapper has to recognize which component items are related, namely, components in the same list. For example, the above wrapper has to recognize which address and telephone number are of restaurant name A for each identified restaurant name A.

For extraction from an arbitrary page, both subtasks are difficult, so extraction with high precision and recall cannot be expected.

In order to make the problem easy, we only consider problems of extracting lists of two items, a *key item* and a *target item*. Furthermore, we assume that a keyword which must be appeared in key item is inputted by a user at every extraction. Constraint that an inputted keyword must appear in key item makes key item identification task easier. Restriction that lists of only two items can be extracted may be seemed not to be so useful, but joining results extracted by different wrappers makes it possible to give a view that is composed of lists of more than two items.

2.2 Four-Phase Extraction Based on Structure and Contents

To realize flexible wrappers for the keyword-related target item extraction, we propose the following four-phase extraction based on structure and contents.

1. **Structure-based Extraction Phase**

Extract key item candidates by string matching for an inputted keyword. Then, extract related target item candidates for each key item candidate using structural relations between key item and target item, which have been already learned from training data.

2. **Content-based Filtering Phase**

   Score each key item candidate by its feature values using a function which measures appropriateness as a key item specified for a wrapper. This function is also learned from training data. Similarly, score each target item candidate using a function specialized for it. Filter out candidates of which scores for key or target items are less than a predefined threshold.

3. **Decision Phase**

   Select only one target item candidate for each key item candidate depending on its score, scores of its related target item candidates and structural measures like distance between the key and its related target item candidates.

4. **Error Correction Phase**

   If relation between a key item candidate and its target item candidate is inconsistent with the repetition structure of the page, correct it if possible and reject it otherwise.

3 One Inprementation of Our Approach

3.1 Document Structure and Extraction Unit

As a structure of a web page, we deal with a DOM(Document Object Model)-tree structure. A DOM tree is an *ordered labeled* tree in which each node is labeled with an HTML (or XML) tag except for a node called a *text node*, which is labeled with a special label, '#text'. Each text node has a *text attribute* value that is a corresponding text appearing in an HTML (or XML) document. Our DOM tree does not have any other attributes such as the attribute 'bgcolor' for the node labeled with the tag 'body'[1]. We call a node labeled inline tag or '#text' an *inline node*, and call a node labeled block-level tag as a *block-level node*. An id of a node is its position in the preorder traversal of a tree.

A *span*, which is represented by a pair of two leaf nodes (N_l, N_r), is defined as all nodes from leaf N_l to leaf N_r in the preorder traversal of a tree[2]. An extraction unit we adopt is a span, namely, we consider the problem of extracting a pair of a key item span and a target item span, a *key span* and a *target span* for short.

[1] TreeMiner [7] uses those attributes.

[2] Note that this definition is different form the one used in [4].

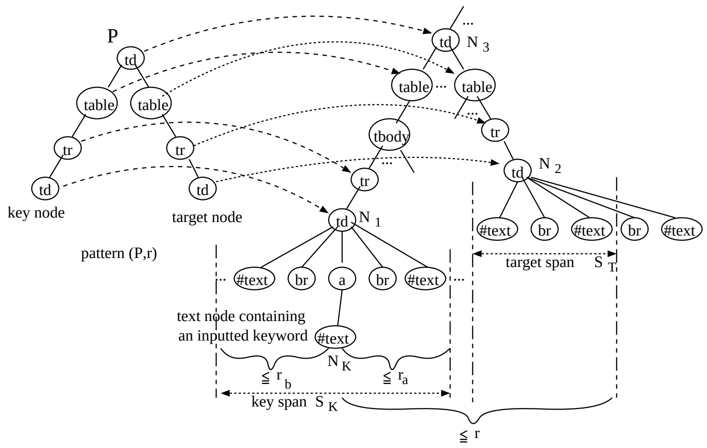

Figure 1: An example of a triple (N_K, S_K, S_T) matching pattern (P, r).

3.2 Structure-based Extraction Phase

In this phase, we extract all candidates represented by triples (N_K, S_K, S_T) using structural pattern matching, where N_K is a text node whose text attribute value contains an inputted keyword, S_K is a candidate key span that includes N_K, and S_T is a target span that matches a pattern together with N_K and S_K. Patterns used in pattern matching are a member of pattern set $\mathcal{P}$ that are learned from positive instances in training data. (See Sec. 4.2) For flexibility, all triples that match one of patterns in $\mathcal{P}$ are extracted as candidates. A pattern (P, r) in $\mathcal{P}$ is composed of a pattern tree P and limit distance r from N_k to the farthest node in S_T. Here, the distance between two nodes are difference between those ids. A pattern tree P is a DOM tree which is composed of a root node, two leaf nodes called *key node* and *target node* and two paths connecting the root node with the two leaf nodes. Note that all nodes in pattern trees are block-level nodes, namely, pattern matching is basically done for structures of block-label nodes. Whether a target node is located after or before a key node depends on each pattern.

A triple (N_K, S_K, S_T) is said to *match pattern* (P, r) for inputted keyword K if the following conditions are satisfied. (See Figure 1.)

1. Node N_K is a text node whose text attribute value contains inputted keyword K. Span S_K contains node N_K. The distance between N_K and the first node in S_K is at most r_b, and the distance between N_K and the last node in S_K is at most r_a, where r_b and r_a are common constants for all patterns, which are learned from training data. The distance between N_K and the farthest node in span S_T is at most r.

2. The lowest block-level node N_1 among the common ancestors of the two boundary nodes in S_K has the same label as the key node of pattern tree P has.

Similarly, the lowest block-level node N_2 among the common ancestors of the two boundary nodes in S_T has the same label as the target node of P has.

3. The lowest common ancestor N_3 of N_1 and N_2 has the same label as the root node of P has. Furthermore, the label sequence of the path from the root node to the key node in P is a subsequence of that from N_3 to N_1. (Matching like this is called *elastic matching* [10].) Similarly, the label sequence of the path from the root node to the target node in P is a subsequence of that from N_3 to N_2.

We use pattern set $\mathcal{P}$ in which the combination of the labels of the root, key and target nodes in each tree P is different from those in the other trees. Thus, for each triple (N_1, N_2, N_3) defined above, only one pattern P is matched, but there still remain some variation with respect to the ranges of two spans S_K and S_T. We only select the triples (N_K, S_K, S_T) with the widest ranges of S_K and S_T as candidates. Note that the range is narrowed to an appropriate one in the decision phase.

3.3 Content-based Filtering Phase

In this phase, spans S_K and S_T are scored by their appropriateness as key and target items, respectively, for each candidate (N_K, S_K, S_T) extracted in the structure-based extraction phase. Two scoring functions used here, one for key item candidate S_K and the other for target item candidate, are learned from training data.

Score is calculated for each leaf node in spans S_K and S_T. Score for a span S_K or S_T is also calculated as the maximum score among those of leaf nodes in the span.

The input of a scoring function is an attribute vector of a leaf node. In the case with extraction of text items, we use an *index term vector* [2] as an attribute vector. An inputted keyword K is different every time but score of N_K should be high, so we replace K with the same special word. As a scoring function, we use SVM (Support Vector Machine) which can be used for real vector inputs and of which score is the margin from the separation hyperplane.

Candidates (N_K, S_K, S_T) such that at least one of two spans S_K and S_T has score of less than θ_1 are filtered out in this phase, where θ_1 is a fixed threshold. In our experiment, we set θ_1 to 0.

3.4 Decision Phase

First, spans S_K and S_T in each candidate (N_K, S_K, S_T) are narrowed to the appropriate ranges. For conducting the task, we use another fixed threshold θ_2 in addition to θ_1. Span S_K is narrowed to the widest span among its subspans satisfying the following three conditions: 1) containing N_K, 2) whose boundary nodes have score of at least θ_1 and 3) not containing any nodes having score less than θ_2. Similarly, span

S_T is narrowed to the widest span among its subspans satisfying condition 1') below and conditions 2) and 3) above: 1') containing the node nearest to N_K among the nodes with score of at least θ_1 in S_T, In our experiment, we set θ_2 to the score for the empty text.

Among all remaining candidate (N_K, S_K, S_T) with the same N_K, we select only one candidate of which S_T has a node nearest to N_K.

3.5 Error Correction Phase

In the structure-based extraction phase, many target span candidates are extracted for the same N_K, and that seems inevitable for flexible extraction. Since some correct target spans are difficult to be recognized as a target item from their contents only, some correct candidates are filtered out in the content-based filtering phase. In such a case, finally survived candidates are not correct ones. Some of such errors can be detected by repetition structure analysis because the analysis enables estimation of data record boundaries. Furthermore, even error correction can be done by replacing a wrong survived target span with the span structurally corresponding to it in the estimated same data record as the key span belongs to. This is because data records of the same structure are considered to have the same type of item in the same position.

3.5.1 Rejection-Correction Algorithm

Two important concepts for our algorithm are a *data record* and a *data region* [6]. A data record is a kind of representation of a data object that is a group of its attribute values. A data region is a continuous region that is composed of repeated data records of the same format.

Assume that we have estimated data regions and data records, and also assume that all items in each estimated data records have estimated corresponding items in all the other estimated data records in the same estimated data region.

Then, we can reject or correct the final candidates (N_K, S_K, S_T) as follows.

1. If S_K and S_T are contained in the same estimated data region but different estimated data records, then correct S_T to the span corresponding to S_T in the same estimated data record as S_K is belonging to.
2. If S_K and S_T are contained in different data regions, then reject (N_K, S_K, S_T).

3.5.2 Analysis Using Repetition Representation trees

Zhai and Liu [11] have already developed a method for estimating data records and data regions and also developed a method for aligning data records. Thus, using their method we can reject or correct candidates by our algorithms explained above. Here, we propose a method that is simpler and easier to implement.

We use a representation called a *repetition representation tree* (*RRTs* for short) which was proposed by us [8]. An RRT is a compact representation of a labeled ordered tree having contiguous repetition structure, and it is also a labeled ordered tree that may include special nodes called *repetition information nodes.* A *repetition information node* v_1 is a node labeled number r, which represents such a repetition structure that the subtree sequence rooted by v_1 is repeated r times as a subtree sequence directly rooted by a v_1's parent node p. (See Figure 2.)

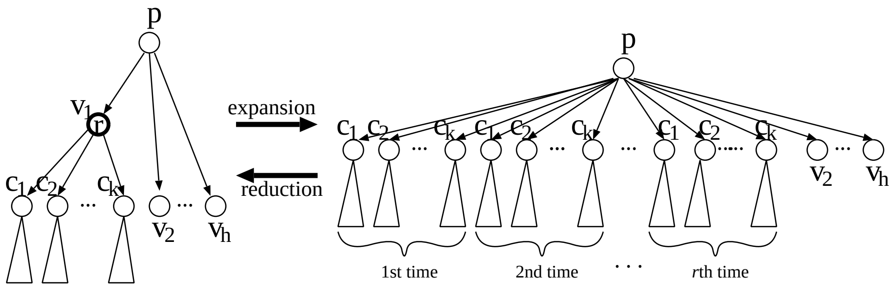

Figure 2: Repetition information node and its expansion

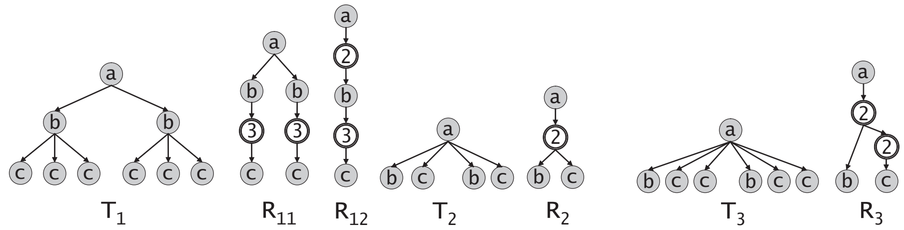

Figure 3: Examples of RRTs

The operation of transforming a tree with repetition information node v into a tree representing the same tree without node v is called *expansion* of v, and the opposite operation is called *reduction* to v. When all the repetition information nodes in RRT R are expanded, a general labeled ordered tree T without repetition information nodes is obtained. Then, we say that R represents T.

Our method for analyzing repetition structure of a labeled ordered tree T is to construct an RRT with the minimum number of nodes among RRTs that represent T. Note that there are various interpretation of repetition structure for a tree, but one interpretation appropriate for human beings exists. From the viewpoint of Occam's Razor, namely, preference for simpler explanation, a minimum-sized representation is expected to capture the real structure well.

We developed an $O(n^3)$ algorithm for minimizing RRTs [8], where n is the number of nodes in an original labeled ordered tree. By restricting RRTs to the ones in which each repetition information node represents a repetition of a sequence of at most k subtrees, we can use an $O(k^2 n)$ algorithm. (We call such an RRT *k-RRT*. See Appendix A for the detailed algorithm.)

One shortcoming of our method comparing to the one developed by Zhai and Liu is that a repetition of subtrees with a small difference cannot be found by our method because repetitions found using RRTs are only repetitions of exactly the same subtrees. However, most such small differences occurs at inline nodes near leafs, and a tree we use in repetition analysis is made from the original tree by removing all inline nodes near leafs, repetitions that cannot be found by our method seem not so many practically. By virtue of capturing exact repetitions, no alignment is needed, which is a merit for practical use.

3.5.3 Data Record Estimation Using a Minimum RRT

Data records in a web page represented by a labeled ordered tree T can be identified using a minimum RRT R representing T. We estimate span S in T satisfying the following conditions as a data record.

1. S corresponds to the subtree sequence rooted by a repetition information node N_R in R.

2. N_R has no ancestor repetition information node.

3. S contains at least two *content* nodes such as text nodes and image nodes[3].

An estimated data region is a span corresponding to the whole region represented by the above N_R.

For use of Rejection-Correction algorithm, we can save computation time for RRT construction a little. For every candidate (N_K, S_K, S_T) rejected or corrected by the algorithm, S_K and S_T are contained in different estimated data records, which is possible only when those data records exist in the minimum subtree containing both S_K and S_T. Thus, first, we calculate a minimum RRT for the minimum subtree containing both S_K and S_T. Then, we also calculate a minimum RRT for the whole tree only when S_K and S_T are contained in different estimated data records in the above minimum subtree. This reduces computation time a lot because in most cases the minimum tree containing the two spans is very small.

4 Training Stage

4.1 Training Data

A training instance is a quintuple (i, K, N_K, S_K, S_T), where i is a page ID, K is an inputted keyword, N_K is a text node in the DOM tree for page i that containing K in

[3]I our experiments, we only checked the number of text nodes.

its text attribute value, and S_K and S_T are a key span and a target span, respectively. One instance is created for every appearance of K in page i. When no appropriate key span for N_K exists, S_K and S_T are set to null. Even when an appropriate key span exists, there may be no appropriate target span. In such a case, S_T is also set to null.

4.2 Pattern Set Generation

Pattern set $\mathcal{P}$ used in the structure-based filtering phase is generated from all training instances with $S_T \neq$null. (We call such instances *positive instances*.) For each instance (i, K, N_K, S_K, S_T), let N_1 and N_2 be the lowest block-level nodes in the DOM tree T for page i among the common ancestors of the boundary nodes of S_K and S_T, respectively, and let N_3 be the lowest common ancestor of N_1 and N_2. (See Figure 1.) Extract tree U from T that is composed of root node N_3 and the two paths from N_3 to N_1 and N_2.

Classify every extracted tree U by structural relation among N_1, N_2 and N_3. Two trees are classified into the same class when their labeled ordered trees composed of only nodes N_1, N_2 and N_3 are equivalent, where those trees are created by replacing the paths from N_3 to the two leaf nodes N_1 and N_2 with the edges between them.

Patterns (P, r) are created for each class C of trees. Pattern tree P is a tree composed of paths from N_3 to N_1 and N_2 whose label sequences are a maximal common label subsequence of the corresponding paths of the trees in C. Such a maximal common label subsequence can be calculated using Apriori type algorithm for sequences [1]. Limit Distance r is set to the maximum distance between N_K and its farthest node in S_T among those of all instances whose extracted tree U belong to the class..

Constants r_b and r_a, that do not depend on individual patterns, are set to the maximum distance between N_K and the first and the last node of S_K, respectively, among those of all instances with $S_K \neq$null.

4.3 Scoring Function Generation

Two scoring functions used in the content-based filtering phase are generated from training instances and pattern set $\mathcal{P}$ that have been already generated from training instances. If learning methods are established for the function used for scoring, and if sets of positive and negative instances and each instance representation appropriate for the function's input are given, then an established learning method produces an appropriate function.

Positive instances for scoring functions of key and target spans are all leaf nodes in non-null S_K and S_T, respectively. For negative instance generation, we execute the structure-based filtering phase using $\mathcal{P}$ to get set of candidates (N_K, S'_K, S'_T). Negative instances for scoring function of key spans are N_K for instances with S_K =null and all leaf nodes contained in some S'_K but not in any S_K. Negative instances for scoring function of target spans are all leaf nodes contained in some S'_T but nodes are restricted to ones between N_K and S_T when $S_T \neq$null. This restriction is necessary because there may be many items appropriate for target spans judging only from there contents when a web page has more than one data records of the similar format.

5 Experiment

We conducted preliminary experiments on two tasks. One is review search for an inputted keyword of a restaurant name, and the other is restaurant name search for an inputted keyword of a city name.

5.1 Data

For each task, we created four datasets as shown in Table 1. HTML files in each dataset are collected by keyword search using Google. Just 10 keywords are used for each dataset. Keywords used for Ramen R, Curry R and Sushi R are names of famous restaurants in category Ramen, curry and sushi, respectively, in Sapporo. Keywords used in NY Sushi R are names of famous Sushi restaurants in New York. Keywords used for Ramen N, Curry N and Sushi N are names of big cities in Hokkaido. Keywords used for America Sushi N are names of big cities in North America. All used keywords are shown in Table 2. Language of input keywords and collected web pages is Japanese except for NY Sushi R and America Sushi N, and language for them is English. As an additional keyword for Google keyword search, we used the telephone number of the restaurant for the datasets of review search and we used a restaurant category name for the datasets of restaurant name search. Only top 50 pages are collected for each keyword in the datasets of restaurant name search.

The numbers of HTML files and keyword appearances in each dataset are shown in the columns named #html and #N_K, respectively. The number of key and target spans appropriate for our tasks are shown in the columns named #non-null-S_K and #non-null-S_T, respectively.

5.2 Methodology

Tasks we address are text item extraction, so we use an index term vector as an attribute vector inputted to scoring functions in the content-based filtering phase. The index terms we used in our experiments are standard forms of words for Japanese texts and stems of words for English texts. We obtained a set of index terms from a Japanese text using MeCab ver. 0.91[4], one of the Japanese morphological analysis

[4]mecab.sourceforge.jp

Table 1: Datasets used in our experiments

name	language	key	target	#html	$\#N_K$	#non-null-S_K	#non-null-S_T
Ramen R	Japanese	restaurant name	review	301	575	386	202
Curry R	Japanese	restaurant name	review	414	948	470	249
Sushi R	Japanese	restaurant name	review	285	531	353	117
NY Sushi R	English	restaurant name	review	328	647	263	98
Ramen N	Japanese	city name	restaurant name	497	2954	2222	1478
Curry N	Japanese	city name	restaurant name	498	3369	2098	803
Sushi N	Japanese	city name	restaurant name	498	4022	3264	1098
America Sushi N	English	city name	restaurant name	498	7810	2656	1282

Table 2: Inputted keywords used in each dataset

name	used keywords
Ramen R	GOZYOGEN, TETSUYA, SUMIRE, ZYUNREN, IKKOKUDO, SAKURAJIMA, SHINA-SOBAYA, APPARETEI, KATORAMEN, MUSASHI
Curry R	Curry Leaves, Gandara, KITARO, KOKORO, Magic Spice, Miruti, Picante, PorePore, Savoy, TENJIKU
Sushi R	AZUMAZUSHI, HANAMARU, KANIGEN, KIKUZUSHI, NAGOYAKATEI, ONI-GASHIMA, OTARUMASAZUSHI, SAIKAKU, SUSHIZEN, TORITON
NY Sushi R	Aki, Avenue A Sushi, BONDST, Bar Masa, Ebisu, Forbidden City, Ikeno Hana, Sushi Hatsu, Tsuki, Kasadela
Ramen N, Curry N, Sushi N	ASAHIKAWA-SHI, OTARU-SHI, CHITOSE-SHI, TOMAKOMAI-SHI, HAKODATE-SHI, SAPPORO-SHI CHUO-KU, SAPPORO-SHI KITA-KU, SAPPORO-SHI NISHI-KU, SAPPORO-SHI SHIROISHI-KU, SAPPORO-SHI TOYOHIRA-KU
America Sushi N	Boston, Chicago, Honolulu, Los Angeles, Miami, New York, Philadelphia, San Francisco, Seattle, Vancouver

Table 3: Extraction performance of our method

dataset	precision [%]	recall [%]	#replace (#correct,#wrong)	#reject (#correct,#wrong)
Ramen R	52.5(73.8)	47.5(66.8)	1 (1,0)	1 (0,1)
Curry R	43.7(55.6)	47.2(60.0)	1 (0,0)	0 (0,0)
Sushi R	39.3(55.6)	39.0(55.1)	1 (1,0)	0 (0,0)
NY Sushi R	33.7(62.9)	30.3(56.6)	0 (0,0)	0 (0,0)
Ramen N	78.2(80.4)	62.0(63.7)	135 (102,0)	7 (7,0)
Curry N	69.4(71.1)	35.2(36.1)	79 (53,0)	4 (4,0)
Sushi N	77.2(80.7)	62.9(65.8)	107 (74,0)	22 (22,0)
America Sushi N	70.4(76.0)	37.4(40.3)	26 (26,0)	14 (14,0)

systems. As MeCab's dictionary, we used IPA dictionary. A set of index terms is obtained from an English text using Porter stemmer (Porter's algorithm in [2]). We used SVMs as scoring functions. An SVM program we used is SVM-Torch II[5]. We used the polynomial kernel with option string "-c 1.0", where the option -c is the parameter C in the non-separable SVM optimization formula (the trade-off between minimizing the margin and the slack variables).

For each dataset, we conducted cross validation by using the instances for 9 keywords as training data and the instances for the rest one keyword as test data.

5.3 Results

We show our experimental results in Table 3. Precision and recall were calculated by judging that an extracted triple (N_K, S_K, S_T) is correct only when both S_K and S_T are extracted correctly . We also calculate precision and recall by loose criteria

[5]www.idiap.ch/~bengio/projects/SVMTorch.html

in which (N_K, S_K, S_T) is correct even when S_K or S_T is a part of the correct span. The numbers calculated using loose criteria are shown in parentheses. As for the restaurant-name search tasks, precisions were 69.4-78.2 and recalls were 35.2-62.9. Recalls are a little low for Curry N and America Sushi N, but these performance results indicate possibility of practical use of our method. For the review search tasks, we had relatively low precisions (33.7-52.5) and recalls (30.3-47.5). In most cases, a target review span is composed of several text nodes, so extracting its exact range seems difficult. Since a part of review can help people's decision, loose criterion seems not a bad measure in this case. By applying loose criterion to review search tasks, precisions and recalls become 55.6-73.8 and 55.1-66.8, respectively. Furthermore, there are non-serious errors such as slightly wider spans, target spans having more than one appropriate key spans (only one key span is correct in our settings), spans difficult to be labeled. By regarding such non-serious errors as correct extractions, precisions and recalls become 72.6-85.2 and 72.0-79.6, respectively, for the review search tasks, which also seems to be performance enough for information exploration.

Our error correction method performed well for the restaurant-name search tasks. The numbers of replacement and rejection are also shown in Table 3. We also wrote the numbers of correct and wrong replacements (rejections) in parentheses. Here, a correct replacement means that a wrong span is replaced with a correct one, and a wrong replacement means that a correct span is replaced with a wrong one. As for rejections, a correct rejection means that a wrong candidate is rejected, and a wrong rejection means that a correct candidate is rejected. Many correct replacements and rejections occurred for the restaurant-name search tasks. However, no effect can be seen for the review search tasks. This is because one whole page is used for one restaurant in many review sites, and most such pages containing reviews of many restaurants do not have firm structures.

6 Concluding Remark

We proposed an approach for realizing flexible wrapper, and showed one implementation following the approach. In our preliminary experiments, we applied our implementation to two different kinds of extraction problems and obtained encouraging results on the both problems. Using both structural and content information, even error correction becomes possible. Current method cannot extract information correctly from a table composed of column data records, but it also seems to become possible on the basis of repetition structure and content information.

References

[1] R. Agrawal and R. Srikant. Mining sequential patterns. In *Proc. of 11th Int'l Conf. on Data Eng.*, pages 3–14, 1995.

[2] R. Baeza-Yates and B. Ribriro-Neto. *Modern Information Retrieval.* ACM Press, New York, NY, 1999.

[3] C.-H. Chang and S.-C.Lui. IEPAD: Information extraction based on pattern discovery. In *Proc. of 10th Int'l World Wide Web Conf.*, pages 4–15, 2001.

[4] W. W. Cohen, M. Hurst, and L. S. Jensen. A flexible learning system for wrapping tables and lists in html documents. In *Proc. of 11th Int'l World Wide Web Conf.*, pages 232–241, 2002.

[5] N. Kushmerick. Wrapper induction: Efficiency and expressiveness. *Artificial Intelligence*, 118:15–68, 2000.

[6] B. Liu, R. L. Grossman, and Y. Zhai. Mining data records in web pages. In *Proc. of 9th ACM SIGKDD Int'l Conf. on Knowledge Discovery and Data Mining*, pages 601–606, 2003.

[7] Y. Murakami, H. Sakamoto, H. Arimura, and S. Arikawa. Extracting text data from html documents. *The Information Processing Society of Japan (IPSJ) Transactions on Mathematical Modeling and its Applications (TOM)*, 42(SIG 14(TOM 5)):39–49, 2001. In Japanese.

[8] T. Saito, A. Nakamura, and M. Kudo. Algorithm for minimizing repetition representation trees. In *Proc. of FIT2006*, pages LA–001, 2006. In Japanese.

[9] K. Tateishi, Y. Ishiguro, and T. Fukushima. A reputation search engine that collects people's opinions using information extraction technology. *IPSJ Transactions on Databases (TOD)*, 45(SIG 07):115–123, 2004.

[10] M. J. Zaki. Efficiently mining frequent trees in a forest. In *Proc. of 8th ACM SIGKDD Int'l Conf. on Knowledge Discovery and Data Mining*, pages 71–80, 2002.

[11] Y. Zhai and B. Liu. Web data extraction based on partial tree alignment. In *Proc. of 14th Int'l World Wide Web Conf.*, pages 76–85, 2005.

A Algorithm for Minimizing k-RRTs

ConsMin$k$RRTree is an algorithm using dynamic programming that returns the root node N_R of a minimum k-RRT representing T with root node N_T.

The problem of constructing a minimum k-RRT can be reduced to a problem of repeatedly solving the following problem.

Problem 1 *Let Σ denote a set of labels and assume that each element l in Σ has real weight $w(l)$. Let T be an arbitrary height-1 ordered tree whose nodes are labeled elements in Σ. Construct a k-RRT R that represents T and has the minimum sum of node weights.*

Function ConsMin$k$RRTH1 is a function that solves Problem 1, namely, the function that returns the root node of a k-RRT with the minimum sum of node weights which represents an inputted node-weighted height-1 tree.

Note that Labeling function used in ConsMin$k$RRTree is a function that labels all the same subtrees rooted by N_i with the same label l_i, and also note that TreeConst function used in ConsMin$k$RRTH1 is a function that constructs and returns a minimum k-RRT using information on the best structure.

ConsMin$k$RRTree %[Construct the Minimum RRT]
⟨Input⟩ N_T: the root node of labeled ordered tree T
⟨Output⟩ N_R: the root node of R,
m_R: the number of nodes in R, where R is an RRT that represents T with the minimum number of nodes

if N_T has no child node **then**
  return $(N_T, 1)$
else
  Let l be the label of N_T and let $C_1, C_2, ..., C_k$ be the sequence of child nodes of N_T. Execute STEP 1∼4 sequentially.
  STEP 1. $(N_i, m_i) \leftarrow$ ConsMinRRTree(C_i) for $i = 1, 2, ..., k$
  STEP 2. $(l_1, l_2, ..., l_k) \leftarrow$ Labeling$(N_1, N_2, ...N_k)$
  STEP 3. $(N_R, m_R) \leftarrow$ ConsMin$k$RRTH1$(l, l_1, l_2, ..., l_k, m_1, m_2, ..., m_k)$
  STEP 4. In tree R, replace the node labeled l_i with node N_i.
end if
return (N_R, m_R)

ConsMin$k$RRTH1 %[Construct the Minimum k-RRT for a Height-One Tree]
⟨Input⟩ l: root label, $(l_1, m_1), (l_2, m_2), ..., (l_n, m_n)$: (label, weight)-sequence of leaves of height-1 tree T
⟨Output⟩ (N_R, m_R): (root node of R, weight sum of nodes in R), where R is an RRT that represents T with the minimum sum of node weights

$(m, \mathrm{type}) \leftarrow$ CreateRRTSizeTable$(l, (l_1, m_1), ..., (l_n, m_n))$
$h[0, 0] = 0$, $\mathrm{best\_idx}[0] = 0$
for $i = 1$ to n **do**
  $h[i, 0] = h[i-1, \mathrm{best\_idx}[i-1]] + m[i, i]$, $\mathrm{best\_idx}[i] = 0$
  for $j = 1$ to $\min\{i, k\}$ **do**
    if $j \le i/2$ and $(l_{i-j+1}, l_{i-j+2}, ..., l_i) = (l_{i-2j+1}, l_{i-2j+2}, ..., l_{i-j})$ **then**
      if $h[i-j, j] \le h[i-2j, \mathrm{best\_idx}[i-2j]] + m[i-j+1, i] + 1$ **then**
        $h[i, j] = h[i-j, j]$
        $\mathrm{rep\_num}[i, j] = \mathrm{rep\_num}[i-j, j] + 1$
      else
        $h[i, j] = h[i-2j, \mathrm{best\_idx}[i-2j]] + m[i-j+1, i] + 1$
        $\mathrm{rep\_num}[i, j] = 2$
      end if
    else
      $h[i, j] = \infty$
    end if
    if $h[i, j] < h[i, \mathrm{best\_idx}[i]]$ **then** $\mathrm{best\_idx}[i] = j$
  end for
end for
return (TreeConst$(l, l_1, ..., l_n, \mathrm{type}, \mathrm{best_idx}, \mathrm{rep_num})$,$h[n, \mathrm{best\_idx}[n]] + 1$)

CreateRRTSizeTable %[Create $R[i, j]$ size table]

⟨Input⟩ $(l_1, m_1), (l_2, m_2), ..., (l_n, m_n)$: (label, weight)-sequence of leaves of height-1 tree T
⟨Output⟩ $(m[i, j], \text{type}[i, j])$ for $i, j \in \{1, 2, ..., n\}$ and $0 \leq j - i < k$: (size, structure) of $R[i, j]$

```
m[i,i] ← m_i for i = 1,2,...,n
m[i,j] ← ∞ for i = 1,2,...,n, j = 1,2,...,n, 0 < j − i < k
for h = 2 to k do
    for i = 1 to n − h do
        for j = 0 to h − 2 do
            if m[i,i+j] + m[i+j+1,i+h−1] < m[i,i+h−1] then
                m[i,i+h−1] ← m[i,i+j] + m[i+j+1,i+h−1]
                type[i,i+h−1] ← (+, j)
            end if
        end for
        for s = i + h to i + k − h by h do
            if (l_s, l_{s+1}, ..., l_{s+h−1}) ≠ (l_i, l_{i+1}, ..., l_{i+h−1}) then break
            if m[i,i+h−1] + 1 < m[i,s+h−1] then
                m[i,s+h−1] ← m[i,i+h−1] + 1
                type[i,s+h−1] ← (×, h)
            end if
        end for
    end for
end for
return (m, type)
```