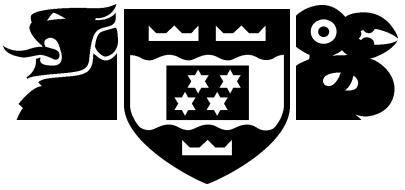
VICTORIA UNIVERSITY OF WELLINGTON
Te Whare Wananga o te Upoko o te Ika a Maui



School of Mathematical and Computing Sciences
Computer Science

Learning Knowledge Bases for Information Extraction from Multiple Text Based Web Sites

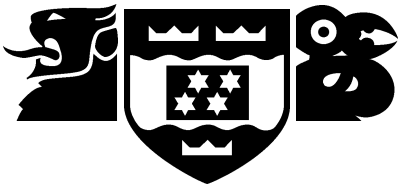
Xiaoying Gao and Mengjie Zhang

Technical Report CS-TR-03/2
Feb 2003

School of Mathematical and Computing Sciences
Victoria University
PO Box 600, Wellington
New Zealand

Tel: +64 4 463 5341
Fax: +64 4 463 5045
Email: Tech.Reports@mcs.vuw.ac.nz
http://www.mcs.vuw.ac.nz/research

VICTORIA UNIVERSITY OF WELLINGTON
Te Whare Wananga o te Upoko o te Ika a Maui



School of Mathematical and Computing Sciences
Computer Science

PO Box 600
Wellington
New Zealand

Tel: +64 4 463 5341, Fax: +64 4 463 5045
Email: Tech.Reports@mcs.vuw.ac.nz
http://www.mcs.vuw.ac.nz/research

Learning Knowledge Bases for Information Extraction from Multiple Text Based Web Sites

Xiaoying Gao and Mengjie Zhang

Technical Report CS-TR-03/2
Feb 2003

Abstract

This paper describes a learning/adaptive approach to automatically building a knowledge base for information extraction from text based web pages. A frame based representation is introduced to represent domain knowledge as knowledge unit frames. A frame learning algorithm is developed to automatically learn knowledge unit frames from training examples. Some training examples can be obtained by automatically parsing a number of tabular web pages in the same domain, which greatly reduced the time consuming manual work. This approach was investigated on ten web sites of real estate advertisements and car advertisements and nearly all the information was successfully extracted with very few false alarms. These results suggest that both the knowledge unit frame representation and the frame learning algorithm work well, domain specific knowledge base can be learned from training examples, and the domain specific knowledge base can be used for information extraction from flexible text-based semi-structured Web pages on multiple Web sites. The investigation of the knowledge representation on five other domains suggests that this approach can be easily applied to other domains by simply changing the training examples.

Keywords Information extraction; learning; knowledge unit fram; text-based web sites; semi-structured data.

Author Information

The authors are lecturers in computer science, Victoria University of Wellington, New Zealand.

Learning Knowledge Bases for Information Extraction from Multiple Text Based Web Sites

Xiaoying Gao and Mengjie Zhang

School of Mathematical and Computing Sciences
Victoria University of Wellington
PO Box 600, New Zealand
{*xgao,mengjie*}*@mcs.vuw.ac.nz*

Abstract

This paper describes a learning/adaptive approach to automatically building a knowledge base for information extraction from text based web pages. A frame based representation is introduced to represent domain knowledge as knowledge unit frames. A frame learning algorithm is developed to automatically learn knowledge unit frames from training examples. Some training examples can be obtained by automatically parsing a number of tabular web pages in the same domain, which greatly reduced the time consuming manual work. This approach was investigated on ten web sites of real estate advertisements and car advertisements and nearly all the information was successfully extracted with very few false alarms. These results suggest that both the knowledge unit frame representation and the frame learning algorithm work well, domain specific knowledge base can be learned from training examples, and the domain specific knowledge base can be used for information extraction from flexible text-based semi-structured Web pages on multiple Web sites. The investigation of the knowledge representation on five other domains suggests that this approach can be easily applied to other domains by simply changing the training examples.

1 Introduction

Since the late 1990s, the World Wide Web (WWW) has been serving a huge, widely distributed, global information service centre for news, advertisements, consumer information, financial management, education, government, e-commerce, and many other information services. Almost all the information is presented for "human reading" only, as text in natural language, marked up using HTML or other markup languages. While keyword search can retrieve a lot of relevant documents from the Web, information extraction techniques and information agents are needed for finding exact information from one or more particular Web documents (Web pages or sites).

However, information extraction from Web is a difficult task. More and more online documents are becoming available and each has a different data format. The number of Web sites and their domains is huge and is growing very fast. The existing Web pages are continuously being updated and their data formats may be changed or updated at any time without warning. While it might be easy to hand craft an information extraction system for one particular Web site in one specific domain at a particular time, how to maintain the system and make it effective, efficient, and adaptive for multiple or new Web sites is a big challenge. There is an urgent need to develop methods and tools to ease the system generation and adaptation.

Recently, a number of information extraction systems have been developed on semi-structured web pages, including manual [6], semi-automatic [2, 3], interactive [1, 11] and automatic [4, 7] systems. However, the wrappers generated by these systems are site specific, that is, an individual wrapper has to be built for each Web site and could not be applied to other Web pages/sites in the same domain.

In addition, most of these systems are developed and applied to Web pages with a strict data format. In Doorenbos [4], it is assumed that data are presented in a uniform format and related data are presented in one single line. The algorithm introduced in Kushmerick [7] works on Web pages where data are presented in a fixed order without any missing items. The

system by Ashish [3] is based on heuristics on font size and indentation and works on pages organized hierarchically. More flexible semi-structured Web sites such as those with free text, with missing units or with free order units are beyond the scope of such systems.

The goal of this paper is to develop a learning/adaptive approach to automatically building a knowledge base for information extraction. Rather than being applied to relatively structured tabular web pages, this approach is designed for information extraction from more flexible, relatively unstructured text-based Web pages such as pages with missing data or data written in free order or even in free text. Instead of building site specific wrappers, this approach will build a single wrapper that can work on multiple Web sites in the same domain. To investigate the effectiveness and adaptability of this approach, flexible text based web sites such as classified real estate advertisements and car advertisements are used as the test beds in the experiments.

In the rest of the paper, we first briefly analyse the tasks and overview the entire system in section 2. Section 3 describes the frame representation developed for representing knowledge and section 4 describes out learning algorithm for building a knowledge base for information extraction. After presenting the experimental results in section 5, we conclude this paper and give future research directions in section 6.

2 Task Analysis and System Overview

2.1 Task Analysis

This research focuses on information extraction from flexible text based web sites, which belong to semi-structured data, an intermediate point between data in free text and structured data in databases. Compared with relatively structured tabular Web pages, the text based web pages are relatively unstructured and generally more difficult to deal with. Typical examples are Web pages provided by online services such as online classified advertisements, online product categories, and online telephone books. Figure 1 shows an example of the kind. These data are written in a relatively unstructured format. For example, the text is telegraphic with a lot of abbreviations, which is much more flexible than data in tabular Web pages. In addition, these web pages usually contain missing data, the data might be presented in free order, even written in a format very close to free text.

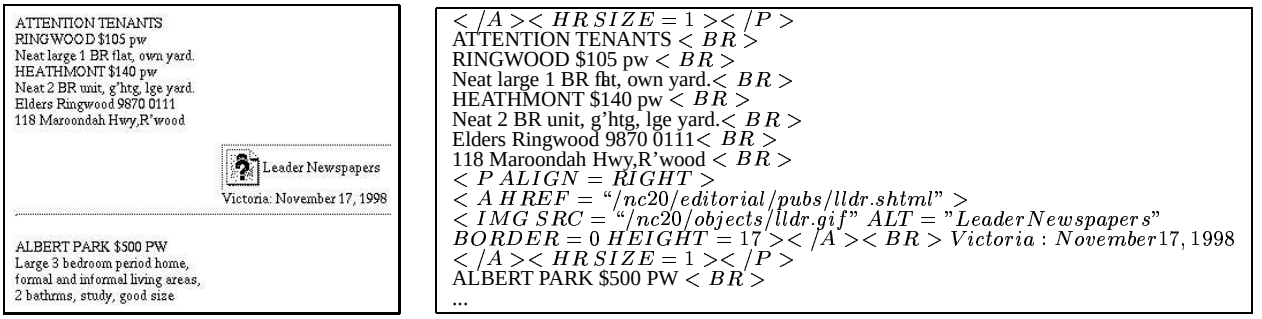


Figure 1. A Fragment of a Web page. (a) Postscript format; (b) HTML format.

We hypothesize that domain knowledge (and common sense knowledge) is important for guiding information extraction from text based web pages. For example, in figure 1, the real estate advertisement domain, a *suburb* is a text string in a local suburb database and usually printed in all capital letters; some *suburbs* start with or end with "east", "north", "south", "west"; a *price* is the number after $ and before a unit such as "per week", "per month" or their abbreviations such as "PW", "p/week", "pwe" and "p/w"; for renting properties, the *price* is usually between $100 PW and $1000PW; a *size* is a number before "bedrooms" or its abbreviations such as "br", "bd", "brs", "bdm" and "brm"; a *type* can be identified by keywords such as flat, apartment, house, unit, home and their synonyms and abbreviations.

Acquisition/gathering/collection of sufficient domain knowledge for information extraction is rather difficult. Manual coding these kinds of knowledge is tedious and error-prone and could not catch up with the growth of the Internet. There are numerous domains on the Internet and the number keeps growing. Even in a particular domain, a hand coded, static knowledge base usually has a lot of limitations. For example, for online real estate advertisements, a system that works in one city (e.g. Melbourne) will not work in another city (e.g. Sydney) because the suburb names are different. In different

countries, the price may be presented as $ or £, “per week” or “per month”. Some features (e.g. central heating) are not a issue in one place (e.g. Melbourne), but might be a key issue in another (e.g. Moscow). Accordingly, ways of automatically, incrementally learning domain knowledge for information extraction should be considered in order to help ease the problem of structure diversity and complexity.

Since more and more online services are became available online, it is not difficult to find some web sites that present the same kind of data in the same domain but in a tabular format. One of our survey in car advertisement domain shows that half of the sites are tabular and half are non-tabular. We hypothesized that information extraction results obtained from the tabular web sites in the same domain can be used as training examples in this approach and that knowledge learned from these training examples can form the knowledge base for the use of information extraction from text based web sites. While such a knowledge base might not contain complete knowledge required for information extraction from multiple flexible text based web sites in a domain, it would greatly reduce manual work.

To enable our system to learn, we develop a frame language for representing the domain knowledge and an algorithm for learning the frames from training examples.

2.2 System Overview

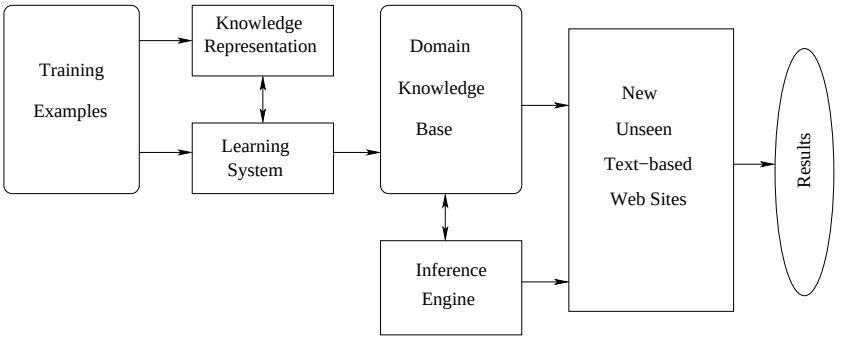


Figure 2. Overview of this approach.

An overview of this approach is shown in figure 2. The approach mainly consists of 4 parts: training example collection, knowledge representation, a learning system to automatically learn and construct a knowledge base, and an inference engine (a matching system) which applies the learned knowledge base to the new unseen text based web pages to extract data as the final results. In this paper, we focus on the first three parts.

3 Knowledge Representation

3.1 Knowledge Unit Frames

For convenience, we call the extracted information as *knowledge units*, denoted as *ku(Name, Value)*. For example, for the first advertisement in figure 1, the knowledge units to be extracted are: *suburb*, *price*, *size*, and *type* denoted by *ku(suburb, ringwood)*, *ku(price, 105)*, *ku(size, 1)*, and *ku(type, flat)*. Each knowledge unit has a name and a value. Its name suggests its content, which is the meaning of the data. Its value is the information to be extracted.

In this approach, we use *knowledge unit frames* to represent the domain knowledge that is useful for extracting knowledge units. The BNF format of our knowledge unit frames is as follows:

```
KnowledgeUnitFrame ::= FrameList
FrameList ::= Frame | FrameList
Frame ::= SlotList
SlotList ::= Slot | SlotList
Slot ::= OneSlot OR SubFrame
OneSlot ::= (SlotName, Slot Value)
SubFrame ::= Frame
```

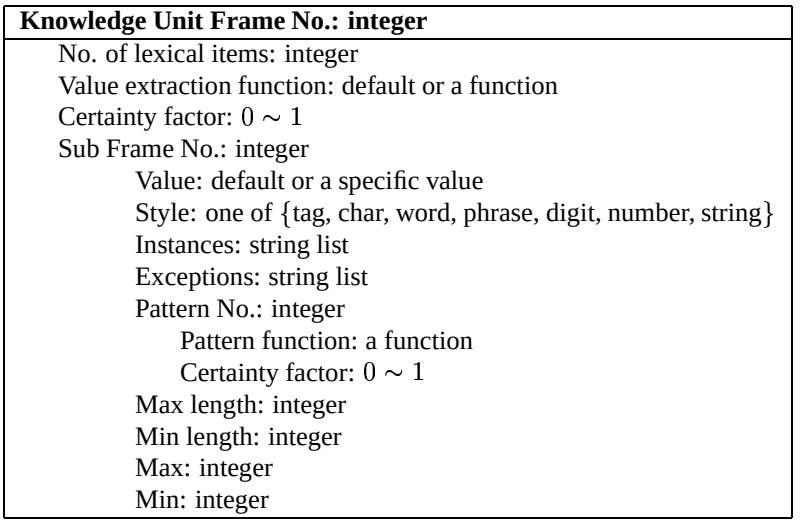


Figure 3. The syntax of the knowledge unit frames

As shown in figure 3, some important features that are useful for knowledge unit recognition and value extraction are chosen to form the slots. Each frame can have 2 or 3 slots (the second slot is optional) and a set of sub frames. The number of *lexical items* of a frame defines the number of sub frames — each lexical item is represented as a sub frame. The *value extraction function* defines how to extract the value of the knowledge unit. The *certainty factor* is used to provide a criterion for choosing between frames. The details of the value extraction function and the certainty factor will be discussed later in this section.

Each subframe has 2 to 8 slots. A subframe must include the second slot *style* and one of the third slot *instances* and the fifth slot *pattern*. All the other slots are optional. The *value* slot specifies the value returned (extracted) by this subframe. The default value is the extracted string when the slot is omitted. If the value is specified, then this subframe will return the specific value. For example, suppose a subframe can extract strings such as "flats", "flat", "SC FLAT", "Apartment", "apart". If we would like the value of the lexical item to be "flat", then we specify this slot as `value(''flat'')`. If the value slot is omitted, any of the extracted strings such as "flats", "SC Flat" can be returned as the value. The *style* slot specifies the data type of the lexical item, including *tag, character, word, phrase, digit, number* and *string*. The *tag* here means HTML tags, which can be used to define lexical items in our system. The *instances* slot contains positive keywords in a list, suggesting that the knowledge unit is present. The *exceptions* slot contains a list of negative keywords, suggesting that the knowledge unit is not present. The lexical item extraction *pattern* slot can have many patterns. Each pattern has two sub slots: a *pattern function* to identify the pattern values, for example, `any_number(X)` for extracting "123" and "5,000", and a *certainty factor* to specify the priority of the pattern. The pattern function and the certainty factor will be detailed later in this section. The *max length* and *min length* slots specify string length constraints, and the *min* and *max* slots specify range constraints for numeric lexical items.

3.2 The Value Extraction Function

In this representation, each subframe extracts a lexical item from a knowledge unit example and all lexical items are returned as a vector $[v(1), v(2), \ldots, v(n)]$, where $v(i)$ is the value extracted by subframe i. The value extraction function defines how to work out the value of the knowledge unit from the vector.

In this approach, we defined the following seven value functions:

- `default` returns a text string concatenating all items in the vector. In this case, this slot is omitted. For example, if a price was extracted by the frame as a vector of `[''$'', ''200'', '' '', ''perweek'']`, the default value would be "$200 perweek".
- `single(v(ID))` returns item No. ID in the vector. In the above example, `single(v(2))` will return "200".

- `times(v(ID, F)` returns `v(ID)*F`. In the above example, if we want all prices to be represented in the unit of "per month", then we use function `times(v(2), ''30/7'')`, which converts the second item "200" to "857".
- `plus(v(ID), F)` returns `v(ID)+F`. For example, a flat *size* was extracted by a frame as the vector ["3", " ", "rooms"], which means the flat has "3 rooms" (assuming 2 bedrooms + 1 living room). If we want the number of bedrooms (2), then we can use `plus(v(1),''-1'')`.
- `multiple(ItemList)` returns all the items in *ItemList*. For example, the string "2-3 Bedrooms" for *Size* returns vector ["2", "-", "3", " ", "bedrooms"]. If we want both the value "2" and "3", we can define the value function as multiple([v(1), v(3)]).
- `concat(ItemList)` returns a text string concatenated by all the items (or constants) in *ItemList*. For example, the string "26:12:92" for *Date* is extracted by a fame as the vector ["26", ":", "12", ":", "92"]. If we want the returned value as "26/12/92", we can use function `concat(v(1), ''/'', v(3), ''/'', v(5))`.
- In function `mapto(v(ID), MaplistName)`, item no. ID is mapped to the value by using a maplist. For example, a commonly used *maplist* is *word-to-number*, which maps "one" to "1", "two" to "2", etc. If we want to convert "one bedroom" (the extracted vector ["one", " ", "bedroom"]) to "1 bedroom", we can use function `concat(mapto(v(1), word-to-number), v(2), v(3))`.

3.3 The Pattern Function

Our system supports 11 pattern functions. *any_string* matches anything; *any_tag* matches anything between $<$ and $>$; *any_number* matches all strings with any digits including digits containing `'',''` in between such as `''2,000''`; *any_word* matches all strings with any alphabetic character and characters with `''/''`, `'' ' ''` in between; *any_phrase* matches any word with only spaces in between; *any_word_in_capital* matches with any word printed in capital letters; *any_phrase_in_capital* matches any phrase printed in all capital letters; *any_delimiter* matches any punctuation and special characters; *any_digit* matches any digit; and *any_char* matches any character. In addition, a special pattern function is defined as *match_string_in_db(DatabaseName)*, which matches a string in a database. For example, `match_string_in_db(suburb)` can be used to extract a particular suburb from a suburb database.

3.4 Certainty Factors

Certainty Factor (CF) is a number between 0 and 1. The frame or pattern with a higher CF has a higher priority, that is, when there are more than one frame or pattern to be used, the one with higher priority is triggered first. The patterns which are often used and make fewer mistakes have a higher CF, while not commonly used or possibly ambiguous ones have a lower CF. The CF value is calculated during the frame learning process, which will be detailed later.

3.5 An Example

The representation of knowledge unit *Price* is given as an example, as shown in figure 4. This frame is expressive enough to extract price from "$120 PW", "800PW", "$1,200/pw", "$300 p/w", or "420 PWeek". Similarly, we use other frames to extract price from "Price: 800", or "£700 per month". In our system, price is represented in five frames.

4 Algorithm for Learning Knowledge Unit Frames

To avoid manually writing knowledge unit frames or at least reduce the amount of manual work, we developed a frame learning algorithm. The inputs of this algorithm are knowledge unit examples. If an example is positive to a particular knowledge unit, it would be a negative example to other knowledge units. The outputs of this algorithm are the frames for these knowledge units. The algorithm is shown in figure 5. Figure 6 shows an examples of learning a price frame from two price examples `$200 PW` and `$600p/week`. The algorithm is detailed in the rest of this section.

<table>
<tr><th colspan="2">Frame: price</th></tr>
<tr><td colspan="2">Knowledge Unit Frame NO. 1</td></tr>
<tr><td colspan="2">No of lexical items: 4
Value extraction function: single(v(2))
Certainty factor: 1.0</td></tr>
<tr><td>Sub Frame No.1
Style: char
Instances: [“$”, “”]</td><td>Sub Frame No.2
Style: number
Pattern No.: 1
Pattern function: any_number(X)
Certainty factor: 1.0
Max: 9999
Min: 0</td></tr>
<tr><td>Sub Frame No.3
Style: char
Instances: [“ ”, “/”, “”]</td><td>Sub Frame No. 4
Style: phrase
Instances: [“pw”, “per week”, “pwe”, “p/we”, “p/w”, “perweek”, “pweek”, “p/week”]</td></tr>
</table>

Figure 4. An Example of Knowledge Unit Frames

1. For the positive examples of each knowledge unit, do the following training process:
   (a) Initialize the first example as a tree. The name of the knowledge unit is the root, and each lexical item is a leave.
   (b) For each new example, initialise it as a tree, then try to combine this tree with the previous tree by
   - Merging nodes if the nodes have the same meaning, or
   - Grouping nodes via creating a generalized node

   (c) Stop when all positive examples are exhausted.
2. Validate the learning results on both positive and negative examples.
3. Write the learned trees to frames and save frames into the knowledge base.

Figure 5. Knowledge unit frame learning algorithm.

4.1 Initialising Trees

Each knowledge unit example is initialised as a tree. We have five basic data types: *phrase, word, number, special character* and *punctuation (delimiters)*. The input string is usually taken as a phrase. It is split when it has a number or a special character such as $, &, @, or space. During the splitting, a phrase is split to 4 other types of data. For example “$200 PW” is initialized to a tree with four sub nodes “$”, “200”, “ ”(space), “PW”.

4.2 Merging Nodes

Two nodes a and b will be merged if one of the four following conditions is met:(1) a and b are the same by ignoring letter cases, e.g. “FLAT” and “Flat”; (2) a and b are based on the same word root, e.g. “Flat” and “Flats”; (3) a and b are synonyms, e.g. “Flat” and “apartment”; and (4) a and b are abbreviations of the same word or phrase, or one is the abbreviation of another (e.g. “apartments”, “apart”, “aparts”). In addition, as a special case, any node a can merge with empty node ϕ.

Of all the four above considerations, the first is straight forward. While natural language processing techniques [8, 9] might be quite helpful for the other conditions, there will be a big speed and memory cost. Considering the fact that the number of words with the same root and the number of synonyms are relatively small in our test domains, we hand-coded a small dictionary to meet our goals for the second and the third conditions. In addition, we developed two abbreviation detection rules for the fourth condition.

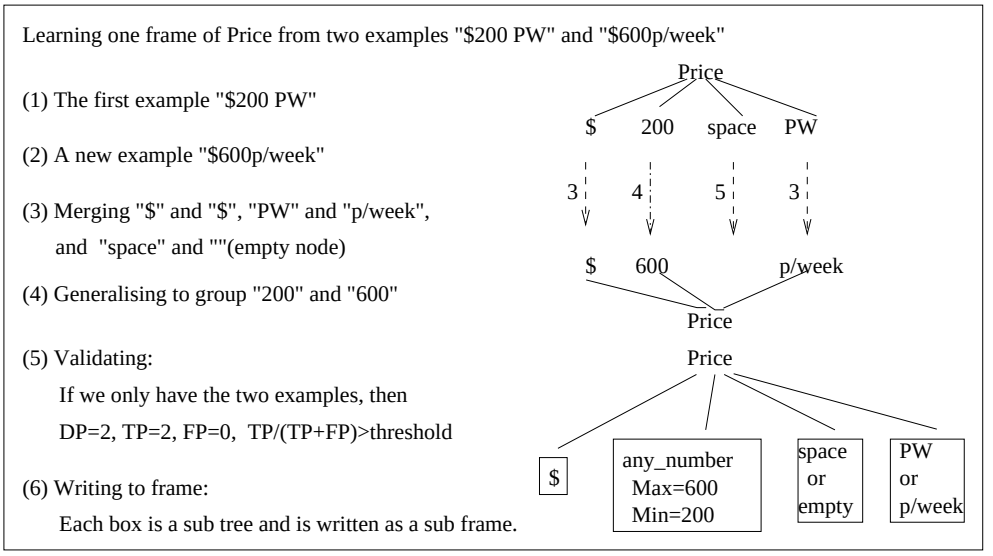


Figure 6. An Example of Learning KU Frames from Training Examples

4.2.1 Abbreviation Detection

Some heuristics are found to be useful to identify the abbreviations of words and phrases. An abbreviation of a word is often created in the following ways: in middle of a word, a number of letters are replaced by " ' " or "/", e.g. "B'wick" or "B/Wick" for "Brunswick"; at the end of the word, a number of letters are replaced by "." e.g. "Melb." for "Melbourne"; the end of the word is simply omitted, e.g. "Melb" for "Melbourne" and "Wel" for "Wellington"; at some extremes, one or two letters are used to represent the whole word, e.g. "E" for "East"; the first letter plus a number of letters selected in the middle or at the end, e.g. "BR", "BRM", "BDRMS" for "bedrooms" and "apart", "aparts", "apt" for "apartments"; for a phrase, the first letters from each word are put together, e.g. "OSP" for "Off Street Parking" and "o.n.o" for "or the nearest offer"; and all words or some words of the phrase may also be abbreviated e.g. "Melb. Uni." for "Melbourne University".

Based on these heuristics, we define two rules for abbreviation detection: Given two words or phrases a and b,

if a is a child of b (or b is a child of a), **then** a and b are *abbreviations*

and

if a has the same first letter with b (`a[1] = b[1]`) and the rest of the letters, either `a[i] = b[i]`, or `a[i] = ''.'', ''/'', '',''` or `a[i] = ""`(omitted), for `i = 2, 3, ...`, **then** word a is a child of word b (b is a parent of a).

4.3 Grouping Nodes by Generalization

We use a generalisation-tree to represent different types of strings, as shown in figure 7. The top level is *any string*. The bottom level is specific data with a particular type such as *phrase*, *special characters*, and *numbers*. Each type of data in the bottom level can be generalized by going upward the tree. We expect the tree to be completely domain independent and site independent so that the learning algorithm is general enough to work on any Web site in different domains.

In order to make the learning effective, we limit the scope of generalization by introducing length constraints for words and phrases and range constraints for numbers. Two parameters *MaxLength* and *MinLength* are used to record the maximum and minimum length for words and phrases. Similarly, *Max* and *Min* are used to record the upper and lower boundaries of numbers.

To reduce overgeneralization, we use *Certainty Factor* (CF) [13] to evaluate the level of generalization. Bottom level nodes are specific data and the CF for each node is set to 1. The top level node can be matched with any input so the CF is set to 0. The CFs of the nodes in the middle levels are between 0 and 1, for example, *any phrase* and *any word* have very low

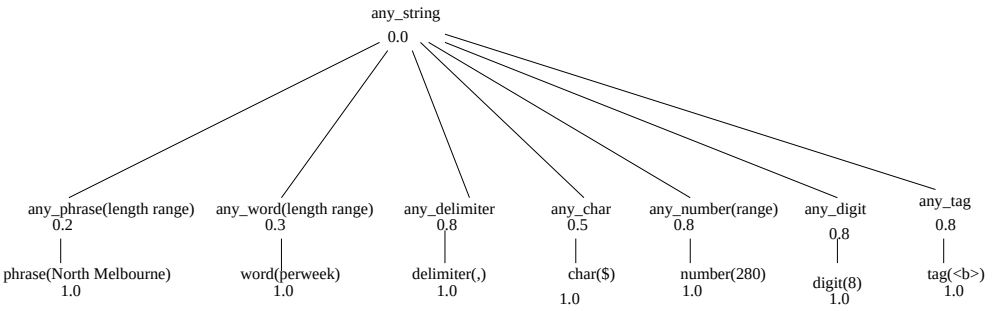


Figure 7. Tree for node generalization

CFs since they can almost match any input within the length range; *any number* and *any char* have a higher CF since they are also limited in data types. The CF values for each data type in the tree is predefined, as shown in figure 7.

CF is used to control the level of generalization. When a node is generalized, its CF becomes lower. The CF of a tree is the maximum CF of all sub nodes. If the CF of the tree is greater than a threshold ε, the generalization will be accepted and accordingly the two examples forms a single tree. Otherwise, the generalization is rejected and two trees need to be formed. Through an empirical search through experiments, we chose $\varepsilon = 0.5$. Let us examine two examples.

Example 1: Price. The second node in figure 6 in both training examples "$200 PW" and "$600 PW" can be grouped together by creating a generalised node *any_number(max=600, min=200)*. The CF of the generalized node is 0.8 (see figure 7), which is lower than either of the CFs of the 2 nodes (both of them are 1.0). The CF of the tree now is

$$CF(tree) = max\{CF(\$), CF(any\_number), CF(space), CF(PW)\} = max\{1, 0.8, 1, 1\} = 1 > \varepsilon$$

So the generalization is accepted. It is important to note that we used the maximum rather than minimum in the tree generalisation since the maximum is suitable for our purpose, that is, as long as one of segments/items in a generalised pattern (associated with a tree) is confident ($>$0.5), the pattern would be valid and accepted.

Example 2: Suburb. If two *Suburb* training examples are given as "Parkville" and "Carlton", the generalization to *any_word(MaxLength=9, MinLength=7)* will be rejected since

$$CF(tree) = max\{CF(any\_word)\} = 0.3 < \varepsilon$$

4.4 Validating

In the training process described above, only positive examples were used. For unseen web pages, it is quite possible for negative examples to be incorrectly considered as positive passing through the trees, which often causes problems. To avoid these problems, we use both positive and negative examples as a tuning set to validate generalized trees.

Assuming DP is the total number of positive examples for a generalised tree, TP (true positives) is the number of positive examples correctly passing through the tree, and FP (false positives) is the number of negative examples incorrectly acting as positive examples passing through the tree, *a generalized node will be rejected if*

$$FP > 0 \text{ and } \frac{TP}{DP} < \varepsilon_1$$

or

$$\frac{TP}{TP + FP} < \varepsilon_2$$

In this research, we chose $\varepsilon_1 = \varepsilon_2 = 0.5$ based on our experiments.

4.5 Writing Trees as Frames into Knowledge Base

In this step, each tree is written as a frame and saved into the knowledge base. Each node, either a merged node, generalised node, or an isolated node, which represents a sub tree, is written into a sub frame. Figure 8 shows how the learning results are saved as frames in the knowledge base. In case there are too many (more than 256) isolated values for a node (corresponding

a propositional knowledge unit such as *suburb*), the node will be written into a single sub frame in the knowledge base, these values such as suburb names will be written to a separated file/database, and the single sub frame in the knowledge base will point to the address of the file/database.

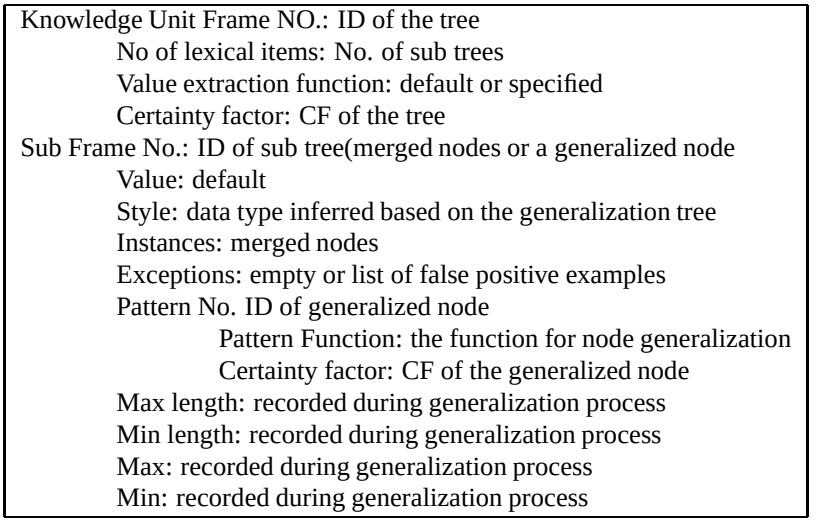


Figure 8. Writing learning results to frames in the knowledge base

Note that all parts of these frames were automatically learned except for the value extraction function, which was specified or simply omitted for default.

5 Results and Discussions

In this research, we did two groups of experiments in the domains of real estate advertisements and car advertisements. After briefly describing training examples used for the two experiments and the performance measurement, this section presents the final information extraction results.

5.1 Training Examples

As discussed in section 2.1, manually collecting sufficient training examples of domain knowledge is very time and cost consuming. In this approach, besides manually collecting some examples from a single web page of a site, we also use the knowledge units automatically extracted by our former wrapper [5] from tabular web pages in the same domain as training examples. While selecting such kinds of tabular web sites needs a bit of manual searching work, it would still reduce the total amount of manual work. In this approach, we use different strategies for forming training examples for different kinds of knowledge units.

- For propositional knowledge units such as *Suburb*, *Car Make* and *Car Model*, a relatively complete instance database needs to be built. In this case, we can use as many pages as possible from a single tabular web site for a domain so that the instances have a wide coverage. Furthermore, it is also possible to collect training examples from an online web site containing this kind of knowledge units.
- For most other knowledge units with numeric values such as *Price* and *Size*, we need a range for these values presented in different formats. Since the knowledge unit values on different pages on a particular tabular web sites are often very similar, a single page for a site would be enough, however, multiple web sites are usually needed. For this kind of knowledge units, in case the tabular web pages do not have sufficient coverage, we can collect examples from sample pages of text based web sites.

No	Domain	Name of Web Site	Precision	Recall
1		NewsClassifieds	94	93
2	Real	Fairfax@Market	98	94
3	Estate	Melbourne Trading Post	90	76
4	Ads	VUT University Accommodation	98	99
5		Infoseek: Classified2000	100	99
6		Infoseek: Classified2000	100	100
7	Car	Auto Trader	92	95
8		Calling All Cars	98	98
9	Ads	Melbourne Trading Post	98	98
10		Fairfax@Market	96	98

Table 1. Results for the basic testing corpus

Ideally we can collect sufficient training examples from the corresponding tabular web pages only, however it is not practical within a limited time. In the experiments, we used 3 tabular web sites for the real estate advertisements and 4 sites for the car advertisements for the experiments from which our former tabular wrapper [5]extract knowledge units as training examples. For the second kind of knowledge units, we also used a page from each text based web site to collect training examples for learning the knowledge base.

5.2 Performance Evaluation

In this approach, we use *recall* and *precision* to measure the system performance. *Recall* refers to the number of knowledge units correctly extracted from a number of Web pages in a particular Web site as the percentage of the total number of knowledge units in these Web pages. *Precision* refers to the number of knowledge units correctly extracted from a number of web pages as a percentage of the total number of knowledge units extracted from those web pages. For example, if there are 100 knowledge units in 6 web pages in a web site and the system extracts 120 knowledge units in total among which 90 are correct, then the recall of this system will be 90% (90/100) and the precision will be 75% (90/120).

5.3 Experiment 1: Result on Basic Corpus

The first experiment is carried out on 10 web sites in the basic corpus, consisting of 5 sites in real estate advertisement domain and 5 in car advertisement domain. The system was tested on two Web pages randomly chosen and downloaded from each site. The final results are shown in Table 1.

As shown in table 1, the performance on these text based web sites varies with the flexibility and complexity of data presentation. However, the approach achieved very promising performance: the precision and recall for knowledge unit extraction are very good (close to the ideal case) on most web sites for both domains. The system with a knowledge base for each of the two domains works well on multiple Web sites. These results suggest that this approach is suitable for information extraction from flexible text based Web sites and that it can be adapt from one domain to another by changing training examples to form the domain knowledge base. In addition, tabular web sites could be used as an important source for automatically extracting knowledge units as training examples and this would reduce the amount of manual work.

5.4 Experiment 2: Results on Two More Flexible Web Sites with More Web Pages

To investigate the extensive power of this approach, we applied our approach on more web pages on two flexible web sites. In most pages of Web sites No. 1 and No. 2 shown in table 1, data are presented as paragraphs written in almost free text. The knowledge units appeared in different order and some extra information (such as agent address and contact details) was also presented. Furthermore, a single paragraph sometimes consists of advertisements for different properties, which needs to be recognised and classified into the correct groups. While the first experiment was only performed on two pages randomly chosen, this experiment used all the 10 pages for each web sites as the test bed. The results of this experiment are shown in Table 2.

As can be seen from table 2, these results show a similar pattern to those in the first experiment. The knowledge unit frames learned from the training examples were successfully applied to extracting knowledge units from flexible text based

No.	Web Sites	Precision	Recall
1	NewsClassifieds	95	95
2	Fairfax@Market	93	93

Table 2. Results from two more flexible web sites

semi-structured web pages, even for free text. This suggests that our knowledge representation and frame learning algorithm are sufficiently powerful for information formation from this kind of web pages.

5.5 Further Investigation of Knowledge Representation

To further investigate how expressive our knowledge representation is, we have examined it to five other domains: *sports scores*, *university information*, *corporate mergers*, *bush fires information*, and *port information*. All pages in these domains were downloaded from the Web and all pages are text based semi-structured. The knowledge units were successfully represented for all the five domains, which suggests our knowledge representation can be used as a general method for representing knowledge units for text-based semi-structured web pages, particularly for representing knowledge units with some range constraints such as page number, date, year, price, qualification, subject codes, credit points, the semester, etc.

6 Conclusions and Future Work

The goal of this paper is to investigate a learning/adaptive approach to automatically building knowledge bases for information extraction from flexible text based web pages. The goal was achieved by developing a frame based knowledge representation and a frame learning algorithm. This approach was tested on ten web sites of real estate advertisements and car advertisements, and nearly all the knowledge units were successfully extracted with very few false alarms. These results suggest that both the knowledge unit frame representation and the frame learning algorithm work well, domain knowledge base can be learned from training examples, and the domain specific knowledge base can be used for information extraction from flexible text-based semi-structured Web pages on multiple Web sites. Some training examples were automatically collected from a number of tabular web pages in the same domain, which greatly reduced the manual work. The investigation of the knowledge representation on five other domains suggests that this approach can be easily applied to other domain by simply changing the training examples.

Our System has the following characteristics:

- Rather than using grammar patterns and linguistics rules which are commonly used in information extraction from natural language, a very small amount of domain knowledge is used to guide information extraction, which greatly reduced the complexity of knowledge representation and learning algorithm and improved the efficiency of information extraction.
- Domain knowledge is clearly separated from other parts of the system, so this system could be adapted to other domains by changing the training examples to automatically learn a domain knowledge base.
- Unlike most systems developed in this area, which usually designed for a particular web site, our system can be used for information extraction from multiple Web sites in a specific domain.

This approach has a number of limitations, which need to work out in the future:

- Although the knowledge base can be almost automatically learned by the frame learning algorithm, a bit of manual work is need to collect sufficient training examples and to check whether there are big mistakes in the learned knowledge base.
- The knowledge unit frame representation is not particularly effective in describing knowledge units with infinite numbers of values and without clear keywords and format, such as *Paper Title*, *Book Name*, *Journal Name*. The identification of these knowledge units is currently based on their context. Better representations of these kinds of knowledge units need to be discovered in the future.

References

[1] B. Adelberg. Nodose- a tool for semi-automatically extracting structured and semistructured data from text documents. *SIGMOD RECORD*, 27:283–294, 1998.

[2] N. Ashish and C. Knoblock. Wrapper generation for semi-structured internet sources. In *Workshop on management of semi-structured data, in conjunction with PODS/SIGMOD*, Tucson, Arizona, 1997.

[3] N. Ashish and C. A. Knoblock. Semi-automatic wrapper generation for internet information sources. In *Proceedings of Cooperative Information Systems*, 1997.

[4] R. B. Doorenbos, O. Etzioni, and D. S. Weld. A scalable comparison-shopping agent. In *Agent 97*, 1997.

[5] X. Gao and L. Sterling. Autowrapper: Automatic wrapper generation for multiple online services. In Gilbert H. Young, editor, *World Wide Web: Technologies and Applications for the New Millennium*, chapter 8, pages 61–70. C.S.R.E.A. Press, 2000.

[6] J. Hammer, H. Garcia-Molina, J. Cho, R. Aranha, and A. Crespo. Extracting semistructured information from the web. In *Workshop on management of semistructured data*, 1997.

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

[8] G. A. Miller. Wordnet: a lexical database for english. *Communications of the ACM*, 38(11):39–41, 1995.

[9] G. A. Miller, R. Beckwith, C. Fellbaum, D. Gross, and K. Miller. 5 papers on wordnet. http://www.cogsci.princeton.edu/ wn/, 1993.

[10] E. Riloff. Automatically constructing a dictionary for information extraction tasks. In *Proceedings of the Eleventh National Conference on Artificial Intelligence*, pages 811–816. AAAI Press/ MIT Press, 1993.

[11] A. Sahuguet and F. Azavant. Wysiwyg web wrapper factory (w4f). http://db.cis.upenn.edu/Publications/, 1999.

[12] L. Sterling. Surveying and reviewing methods of representing scenarios. Technical report, Department of Computer Science, The University of Melbourne, Melbourne, 1998.

[13] L. Yal-inalp and L. S. Sterling. Diagnosing jaundice expert system. *Computer Math. with Applications*, 20:125–140, 1990.