On extracting link information of relationship instances from a web site

Naing, Myo Myo; Lim, Ee Peng; Goh, Dion Hoe-Lian

2003

Naing, M. M., Lim, E. P., & Goh, H. L. (2003). On Extracting Link Information on Relationship Instances from a Web Site. Proceedings of the International Conference on Web Services – Europe 2003, ICWS-Europe 2003, (September 23-24, Erfurt, Germany), Lecture Notes in Computer Science 2853, 213-226.

https://hdl.handle.net/10356/91117

https://doi.org/10.1007/b13818

Naing, M.M., Lim, E.P., and Goh, D.H. (2003). On extracting link information on relationship instances from a Web site. *Proceedings of the International Conference on Web Services – Europe 2003, ICWS-Europe 2003*, (September 23-24, Erfurt, Germany), *Lecture Notes in Computer Science 2853*, 213-226.

Title: On Extracting Link Information of Relationship Instances from a Web Site

Authors: Myo Myo Naing, Ee-Peng Lim, Dion Hoe-Lian Goh

Organization: Nanyang Technological University, Singapore

Email : mmnaing@pmail.ntu.edu.sg,
{aseplim,ashlgoh}@ntu.edu.sg

Contact Author: Myo Myo Naing
mmnaing@pmail.ntu.edu.sg

Address: Centre for Advanced Information Systems,
School of Computer Engineering,
Nanyang Technological University,
Nanyang Avenue, Singapore 639798

Telephone: +65 6790-6333

Abstract : Web pages from a web site can often be associated with concepts in an ontology, and pairs of web pages can also be associated with relationships between concepts. With such associations, web pages can be searched, browsed or even reorganized based on their concept and relationship labels. In this paper, we investigate the problem of extracting link information of relationship instances from a web site. We define the notion of link chain and formulate the link chain extraction problem. An extraction method based on sequential covering has been proposed to solve the problem. This paper presents the proposed method and the experiments to evaluate its performance. We have applied the method to extract link chain information from the Yahoo! Movie Web Site with very promising results.

Keywords: Information Extraction, Hyperlink Structure, Ontology

On Extracting Link Information of Relationship Instances from a Web Site [*]

1 Introduction

1.1 Background and Motivation

Web extraction refers to extracting data from web pages. Due to the heterogeneous nature of web information and the different application usage, many different types of web extraction problems can be defined. Nevertheless, these problems are important because there are enormous amount of web information waiting to be extracted.

In this paper, we assume that a set of ontology concepts and relationships are given to enhance the access to web pages from a web site. The web pages can be associated with the concepts while pairs of web pages can be associated with the relationships between concepts. For example, web pages from a university web site can be usually associated with concepts such as faculty, department, course, lecturer, etc.. These web pages can therefore be treated as concept instances. Relationships between concepts such as TeachCourse(lecturer,course) and OfferCourse(department,course) exist, and so are the relationship instances relating pairs of web pages. When concepts and relationships are associated with web pages, several new possibilities to access a web site become feasible [1]. For example, web pages can be queried by their concept labels, and can be navigated using the relationship labels.

In the previous web extraction research, most efforts have been devoted to the *attribute extraction* [2, 6] and *relation extraction* [3, 8, 14] tasks that assume information to be extracted come from one single web page.

This paper, on the other hand, attempts to investigate the extraction of link information for pairs of web pages that are associated with some relationship. Since the link information may span across multiple web pages, both the training and extraction processes must scan different web pages before a piece of link information can be uncovered.

For example, at the Yahoo! Movies Web Site [1], we may be interested to know the instances of the ActorOf(movie, actor) relationship which are actually the web pages of movies together with the links to their actors' web pages. To efficiently find the link information connecting from the movie web pages to their actor/actress's web pages, we need a fully automated web extraction method.

The main objective of this paper is to formally introduce the link chain extraction problem. We propose that relationship instances can be obtained by extracting the link chains between the source and target web pages of the relationship instances. We also present an extraction method that semi-automatically

[*] This work is partially supported by the SingAREN21 research grant M48020004.

[1] http://movies.yahoo.com/

extracts link chain information using very few training examples from a well-structured web site.

To measure the performance of a link chain extraction method, we further define the precision and recall measures for the extracted link chain information. A series of experiments are also conducted to evaluate our proposed method.

This link chain extraction research is closely related to several applications using web content. In our context, we are interested in automated ontology-based web annotation (OWA) [12] and would like to treat link chain extraction an essential step in automated web annotation.

1.2 Paper Outline

The rest of the paper is organised as follows. In Section 2, we describe the related work. We formally define the link chain extraction problem in Section 3. Our proposed method to learn extraction rules for link chain extraction is given in Section 4. The evaluation of learnt extraction rules on web pages to extract link chain information is given in Section 5. We finally conclude the paper and present future research directions in Section 6.

2 Related Work

There are a lot of extraction systems attempted to extract information from the Web [2]. Depending on the nature of the underlying sources, these systems can be generally categorized into two types: the systems for extracting information from natural language documents and those for extracting information from semi-structured documents. The general survey of different information systems has been presented in [10, 5]. As our web extraction problem involves semi-structured web pages, we survey some related extraction systems focusing on semi-structured information sources.

To extract information from a single web page, text extraction rules are often used to identify the relevant information in the page using some knowledge about the way the page is formatted or structured. In the case of web pages, this knowledge includes the way HTML tags are used to markup their semantic content.

WHISK [13] presented a rule induction approach to generate patterns for the relation instances contained in a semi-structured and free text web page. By using the users' markup training web pages, the WHISK algorithm generates the rules that utilize regular expression patterns to identify the relevant phrases and extract the delimiters of these phrases. WHISK is suitable for situations where multiple records are found in a single web page.

In the SRV project [7], information from HTML sources are extracted by using a set of training examples. The rules in SRV rely on a set of token-oriented features identifying the simple or relational properties of the tokens. These simple

[2] http://www.isi.edu/info-agents/RISE/

properties include *word*, *numeric*, and *punctuation* and the relational properties include *prev-token*, *next-token*, etc. By examining the features found in the training examples, SRV rules are able to extract a single record from a given HTML page. SRV rules are used in the WEB→KB [4] project to extract attribute instances of an ontology.

The system that is closely related to our work is the STALKER [11]. By giving a sequence of tokens around the item to be extracted and an Embedded Catalog Tree, STALKER generates the rules that can handle the hierarchical nature of the items to be extracted. Stalker rules are general enough for the documents with different formats by allowing the rules to have disjunctive properties. Rules generated by WIEN [9] are similar to Stalker but they cannot handle the nested structure or other variation of semi-structured documents.

3 Problem Statement

In this section, we formally define the hyperlink information extraction problem. Before that, we define a few important terms as follows:

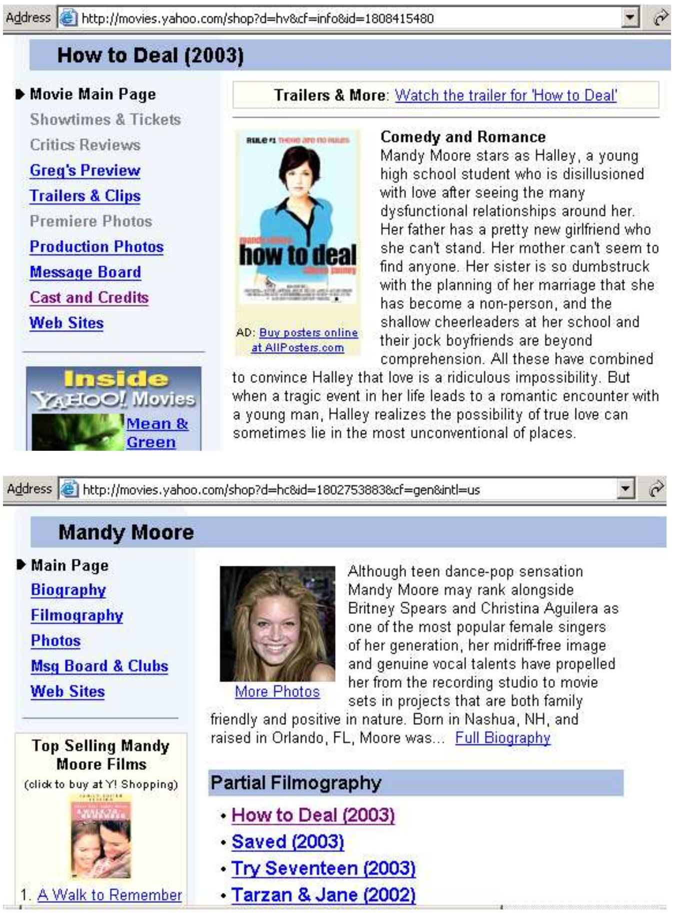
Let w be a web page. The URL of w is denoted by $w.url$. Let l be an anchor element in a web page. The target URL and anchor text of l are denoted by $l.target$ and $l.atext$ respectively.

Definition 1. (Relationship Instance)
Let $R(C_s, C_t)$ be a relationship in an ontology, a pair of web pages (w_1, w_2) from a web site is an **instance of** *$R(C_s, C_t)$ if w_1 is an instance of C_s, w_2 is an instance of C_t, and w_1 is semantically related to w_2 by the relationship R. If (w_1, w_2) is and instance of $R(C_s, C_t)$, we call w_1 and w_2 the* **source concept instance** *and* **target concept instance** *respectively.*

Definition 2. (Link Chain)
Let (w_1, w_2) be an instance of a relationship $R(C_s, C_t)$. The **link chain** *of (w_1, w_2) with respect to $R(C_s, C_t)$ is a list of* **link elements** *denoted by $((p_1, l_1), (p_2, l_2) \ldots, (p_n, l_n))$ where $p_1 = w_1$, $l_n.target = w_2.url$, l_i is an anchor element in p_i and $l_i.target = p_{i+1}.url$ $\forall 1 \leq i \leq n (n \geq 2)$, .*

The source concept instance and target concept instances may or may not be directly linked to each other. When they are indirectly linked, one or more intermediate pages will be included in the link chain to provide the list of link elements. To uniquely locate each anchor element, say l_i, in a page, we can represent each anchor element by its page offset denoted by $l_i.pos$.

Example:
Consider the web page pair (w_i, w_j) shown in Figure1. The URLs of w_i and w_j are $w_i.url$=“http://movies.yahoo.com/shop?d=hv&cf=info&id=1808415480”, and $w_j.url$=“http://movies.yahoo.com/shop?d=hc&id=1802753883&cf=gen&intl=us” respectively. The (w_i, w_j) pair is an instance of the relationship $ActorOf(Movie, Actor)$.



How to Deal (2003)

Trailers & More: Watch the trailer for 'How to Deal'

Comedy and Romance
Mandy Moore stars as Halley, a young high school student who is disillusioned with love after seeing the many dysfunctional relationships around her. Her father has a pretty new girlfriend who she can't stand. Her mother can't seem to find anyone. Her sister is so dumbstruck with the planning of her marriage that she has become a non-person, and the shallow cheerleaders at her school and their jock boyfriends are beyond comprehension. All these have combined to convince Halley that love is a ridiculous impossibility. But when a tragic event in her life leads to a romantic encounter with a young man, Halley realizes the possibility of true love can sometimes lie in the most unconventional of places.

Address
http://movies.yahoo.com/shop?d=hc&id=1802753883&cf=gen&intl=us

Mandy Moore

Although teen dance-pop sensation Mandy Moore may rank alongside Britney Spears and Christina Aguilera as one of the most popular female singers of her generation, her midriff-free image and genuine vocal talents have propelled her from the recording studio to movie sets in projects that are both family friendly and positive in nature. Born in Nashua, NH, and raised in Orlando, FL, Moore was... Full Biography

Top Selling Mandy Moore Films
(click to buy at Y! Shopping)

1. A Walk to Remember

Partial Filmography

- How to Deal (2003)
- Saved (2003)
- Try Seventeen (2003)
- Tarzan & Jane (2002)

Fig. 1. Example web page instances of ActorOf(Movie, Actor) Relationship

The link chain of (w_i, w_j) with respect to $ActorOf(Movie, Actor)$ is $((p_1, l_1), (p_2, l_2))$, where $p_1 = w_i$, l_1 is an anchor element in w_i such that
$l_1.target = p_2.url =$"http://movies.yahoo.com/shop?d=hv&id=1807599125&cf=cast"
$l_1.target = w_k.url$ for some intermediate web page w_k
l_2 is an anchor element in w_k such that $l_2.target = w_j.url$

3.1 Anchor Element Path and Patterns

To extract link chain for a given web page pair, we borrow the notion of embedded catalog(EC) description defined in the Stalker project [11] for describing the structure of *selected* anchor elements within a web page known as *anchor element path*. The original embedded catalog has been defined to represent a web page by a tree-like structure consisting of basic values as leaf nodes and composition constructs as internal nodes. Instead of using an embedded catalog to describe all components found in a web page, we use anchor element path to describe the anchor element to be extracted from a web page in order to form a part of a link chain.

Definition 3. (Anchor Element Path)
Let w be a source target instance (source web page) or an intermediate web page of a link chain $((p_1, l_1), (p_2, l_2), \ldots, (p_n, l_n))$. That is, there exists $i, 1 \leq i \leq n$ such that $p_i = w$. The **anchor element path** *of w is defined as a tuple (N, E) where N and E denote a set of nodes and a set of directed edges respectively, such that the nodes in N are connected by edges from E to form a directed chain. Each non-root internal node is either a* repetitive list *of another internal node or a leaf node. The leaf node represents the anchor element l_i in w.*

In the above definition, the root node represents the entire web page. Each internal node in the anchor element path represents some grouping of anchor elements of the same type. This arises when the link chains of different instances of the same relationship share some common web page (often the hub page of the website), and the anchor elements linking to the next web page of these link chains are grouped together as a list or table entries.

Definition 4. (Anchor Element Pattern)
The **anchor element pattern** *of a link chain $((p_1, l_1), (p_2, l_2), \ldots, (p_n, l_n))$ is defined as an ordered list of* **anchor element paths***, $(AEP_1, \ldots, AEP_n)$ such that each p_i has AEP_i as its anchor element path.*

Definition 5. (Well Structured Web Site)
A web site is known to be **well structured** *if all instance of any given relationship found at the web site have their link chains sharing the same anchor element pattern.*

In this paper, we assume that the web site from which the link chains are to be extracted is well structured. This assumption is necessary because rules for extracting link chains can only be discovered for structured web sites. For unstructured websites, the heterogeneous representation of link chain information will make it impossible for any rule learning method to derive some useful patterns.

Example:
Consider the Yahoo! Movies website, the instances of relationship *ActorOf(Movie, Actor)* share the anchor element pattern (AEP_1, AEP_2) shown in Figure 2. AEP_1 consists of only a root node and a leaf node representing the anchor element involved in the link chain starting from a movie page. AEP_2 consists of a root node, another internal node and a leaf node. While the leaf node represents the anchor element involved in the link chains that leads to an actor page, the anchor element is found in an intermediate page (known as the Casts and Credits page shown in Figure 3.). As the intermediate page groups all anchor elements linking to different actor pages as a list, the internal node in AEP_2 is therefore defined to represent this grouping.

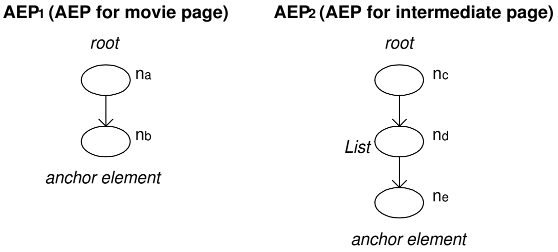


Fig. 2. Anchor Element Pattern of ActorOf(Movie, Actor) Relationship

3.2 Extraction Rules Definition

Anchor element paths and anchor element patterns are designed to model the structure of web pages containing the anchor elements involved in link chains. They alone, however, cannot be used for extracting anchor elements from web pages. We therefore have to introduce the notion of **extraction rule**.

Given an anchor element path, $AEP = (N, E)$, we associate an extraction rule with each non-root node in N.

Definition 6. (Extraction Rule)
An extraction rule is defined as a pair $< sr, tr, er >$ *where* sr, tr *and* er *represent the start, target and end rules respectively.*

Given a non-root n in AEP, the start, target and end rules of n are denoted by $n.sr$, $n.tr$ and $n.er$ respectively.

Definition 7. (Start/End Rule)
A **start/end rule** *is defined as a $Skipto(\langle landmark\rangle)$ predicate or a disjunction of conjuncts of $Skipto(\langle landmark\rangle)$ predicates. The argument $\langle landmark\rangle$ represents a sequence of tokens.*

Definition 8. (Target Rule)
A **target rule** *is defined as a $HasAnchorTextPattern(\langle regularExpression\rangle)$ predicate.*

The purpose of a start (or end) rule is to skip the content that appears before (or after) portion of content represented by a node. The target rule, only applies to a leaf node, is used to verify the content within the portion of web page represented by a leaf node. The $\langle landmark\rangle$ of a start (or end) rule is a sequence of tokens which can be string tokens or token classes. There are 7 different token classes, namely, HTMLTag, Punctuation, Alphabetic, Alphanumeric, Numeric, AllCaps, and Symbol.

A $Skipto()$ predicate skips everything in the web page content until it has consumed a sequence of string tokens matching its landmark. The $Skipto()$ predicate returns True if a match is found, and False otherwise.

The $HasAnchorTextPattern()$ predicate matches the anchor text of a given anchor element with its regular expression ($\langle regularExpression\rangle$) argument. If the anchor text satisfies the regular expression argument, the predicate returns True. Otherwise, a False value is returned.

3.3 Extraction Rule Evaluation on a Web Page

Given the AEP of a web page, we would like to be able to evaluate the extraction rules of the AEP nodes and extract anchor elements from the web page. The extraction rule evaluation algorithm is given in Algorithm 1.

In the EvaluateRule algorithm, *page* and AEP_{page} are the given web page and its AEP respectively. The $anchor - elements$ variable stores the list of anchor elements from the web page. The algorithm evaluates the non-root nodes in a top-down manner. The variable *content* is used to store the portion of web content extracted using the start and end rules of the parent node.

4 Extraction Rule Learning Problem

4.1 Formal Definition

Having define the extraction rules for AEPs, we now formally state the extraction rule learning problem as follows:

Definition 9. Extraction Rule Learning Problem
Given a set of link chains $\{lc_1, \ldots, lc_m\}$ of instances belong to a relationship $R(C_s, C_t)$ where $lc_i = ((p_{1i}, l_{1i}), \ldots, (p_{ni}, l_{ni}))$, the anchor element pattern of web pages in the link chains ($AEP_1, \ldots, AEP_n$), the **extraction rule learning problem** *is to derive the extraction rule for each non-root node in $AEP_1, \ldots, AEP_n$.*

Algorithm 1 EvaluateRule($page$,AEP_{page})

```
Let anchor-elements be empty
Let content be page
for each non-root node $n$ in $AEP_{page}$ from the topmost level to the leaf level do
  if $n$ is a leaf node then
    Extract an anchor element $ae$ by applying $n.sr$ and $n.er$ on content
    if $ae$ satisfies $n.tr$ then
      Add $ae$ to anchor-elements
    end if
  else
    Extract a new content by applying $n.sr$ and $n.er$ on content
  end if
end for
Return anchor-elements
```

As shown the above problem definition, we assume that the AEPs of web pages are given. A set of link chains is also used as input to generate rules. Since the extraction rules consist of start, end and target rules, we will describe how the given link chains are used to derive training data for learning different types of rules. We will also present our proposed rule learning method.

4.2 Training Data

Let a web page w be a sequence of tokens S. A token is a number, a word or a HTML tag. When a web page w follows the structure of a particular AEP, we say that w as an instance of AEP. The content of a root node in AEP is the entire sequence S, and the content of a child node in AEP is a subsequence of the content of its parent node.

We divide the training data for rule learning into two components, the training data for generating start and end rules denoted by TR, and that for generating target rules denoted by EX. Given a non-root node n in AEP, the two kinds of training data are denoted by $n.TR$ and $n.EX$. Hence, from the input link chains, it is necessary for us to derive $n.TR$ and $n.EX$ for every non-root node n in AEP.

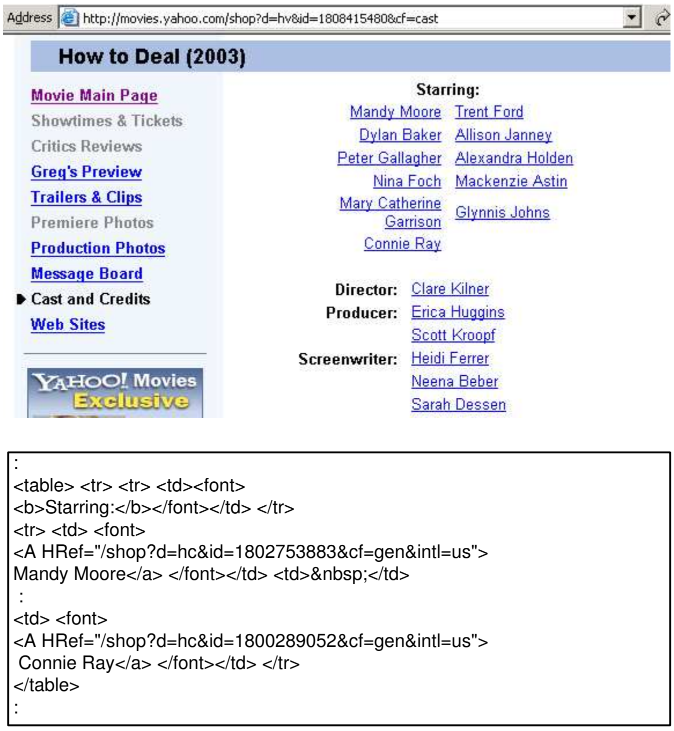
Given a non-root node n from a AEP, $n.TR$ consists of a set of $(content, prefix^{+}, prefix^{-}, suffix^{+}, suffix^{-})$ tuples, one for each web page having AEP as the anchor element path. The *content* component represents the web page content extracted by the start and end rules of its parent node of n. The $prefix^{+}$ and $prefix^{-}$ components are the positive and negative training sequences for $n.sr$ respectively. The $suffix^{+}$ and $suffix^{-}$ components are the positive and negative training sequences for $n.er$ respectively. Suppose the content to be extracted from a training web page for n is a subsequence $(content[k], content[k+1], \cdots, content[m-1])$ and *content* has a length of p A positive training sequence in $prefix^{+}$ is $(content[1], \cdots, content[k-1])$. A positive training sequence in $suffix^{+}$ is $(content[m], \cdots, content[p])$. Once $prefix^{+}$ is derived, equal number of negative training sequences are derived from *content* and included in $prefix^{-}$. The same applies to $suffix^{-}$.

If n is a leaf node, $n.EX$ is also required. It consists of a set of anchor text of the anchor elements from the training web pages for AEP and is to be used as training data to generate the target rule.

4.3 Extraction Rule Learning Algorithm

With the training data, we learn the start and end rules using the **GetSERule()** algorithm, and the target rules using the **GetTargetRule()** algorithm shown in Algorithms 2 and 4.

We demonstrate how to generate these rules using the anchor element path AEP_2 in Figure 2 and a web page satisfying AEP_2. Figure 3 illustrates the part in the web page containing the *list* of anchor elements leading to the different actor/actress pages.



```
:
<table> <tr> <tr> <td><font>
<b>Starring:</b></font></td> </tr>
<tr> <td> <font>
<A HRef="/shop?d=hc&id=1802753883&cf=gen&intl=us">
Mandy Moore</a> </font></td> <td>&nbsp;</td>
 :
<td> <font>
<A HRef="/shop?d=hc&id=1800289052&cf=gen&intl=us">
 Connie Ray</a> </font></td> </tr>
</table>
:
```

Fig. 3. Example intermediate web page and it's partial source

The start and end rule to be learnt for the non-root internal node in AEP_2 are:

$n_d.sr =$ `Skipto( <b>Starring:</b></font></td> )Skipto( <font> )`
$n_d.er =$ `Skipto( </table> ) Skipto( </font></td> )`

The start rule, $n1.sr$, skips everything from the beginning of the web page until reaching the landmark "`(<b>Starring:</b></font></td>)`" and again skips the remaining content until the landmark "`(<font>)`". The end rule $n1.er$ starts from the end of the web page, skips everything until reaching the landmark "`(</table>)` " and again skips the remaining content until reaching the landmark "`(</font></td>)`".

The following start, end and target rules are to be learnt for the leaf node of the AEP_2:

$n_e.sr =$ `Skipto( <font> )`
$n_e.er =$ `Skipto( </font></td> )`
$n_e.tr =$ `HasAnchorTextPattern(`*`namepattern`*`)`

The rules $n_e.sr$ and $n_e.er$ are quite straightforward. The target rule $n_e.tr$ verifies if the anchor text within the anchor element extracted using $n_e.sr$ and $n_e.er$ matches the *namepattern* pattern which is defined as follows:

namepattern = `[A-Z][a-z]{1,15}[-]?\s?[A-z]?\.?\x?[[A-Z][a-z]{1,15}]*`

The above pattern looks for name string that begins with a uppercase character. Several other patterns in the form of regular expressions, e.g. date pattern, can also be defined and used in target rules.

The **GetSERule()** algorithm (see Algorithm 2) is a sequential covering algorithm adapted from [11]. It learns the start or end rule of the given *examples* in $n.TR$ for a non-root node n of the AEP. To generate a start rule, *examples* consists of a set of ($content$, $prefix^+$, $prefix^-$. To generate an end rule, *example* consists of a set of ($content$, $suffix^+$, $suffix^-$) tuples. As long as there exists an uncovered positive examples (prefixes or suffixes), it tries to generate the perfect rule. Once all the positive examples are covered, the best start or end rule is returned.

Algorithm 2 GetSERule(*examples*)

```
SERule ← empty
while examples ≠ empty do
    rule ← LearnRule(examples)
    examples ←(examples − examples covered by rule)
    SERule ← SERule + rule
end while
Return SERule
```

The **GetSERule()** algorithm calls **LearnRule()** function to learn a perfect rule. This latter first generates the initial set of candidates by using the shortest prefix (or suffix) in the training prefixes (or suffixes). It repetitively selects and

Algorithm 3 LearnRule(*examples*)

```
$Seed$ ← $example_i$ with shortest length in $examples$
$Candidates$ ← GetInitialCandidates($Seed$)
repeat
    BestRefiner ← GetBestRefiner(Candidates, $examples$)
    BestSolution ← GetBestSolution(Candidates ∪ BestSolution, $examples$)
    Candidates ← Refine(BestRefiner, $Seed$)
until IsPerfect(BestSolution) or BestRefiner is $empty$
return PostProcess(BestSolution)
```

Algorithm 4 GetTargetRule(EX)

```
Let $P$ be a array of predefined pattern strings
for each $P[i]$ in $P$ do
    $match\text{-}count[i]$ ←0
end for
$max = match\text{-}count[1]$
for each $A_{textj}$ in EX do
    for each $P[i]$ in $P$ do
        if $A_{textj}$ match $P[i]$ then
            $match\text{-}count[i]$ ← $match\text{-}count[i]$+1
            if $match\text{-}count[i] > max$ then
                $max$ ← $match\text{-}count[i]$
                $position$ ← $i$
            end if
        end if
    end for
end for
$a - pattern = P[position]$
$TargetRule$ ← $HasAnchorTextPattern$(a-pattern)
return $TargetRule$
```

refines the best candidate until the candidates are empty or the rule is perfect. The GetBestRefiner() and GetBestSolution() functions are based on some predefined heuristics to refine the rules and they return the best solution by applying different set of criteria.

The **GetTargetRule()** generates a target rule by examining the training anchor texts in $n.EX$. Each training anchor text is compared with the pre-defined patterns. The pre-defined pattern with maximum match count is determined and used as the argument of the $HasAnchorTextPattern$ predicate which is in turns returned it as the target rule.

5 Link Chain Construction

In this section, we describe how the link chains of a particular relationship are to be constructed by using ConstructLinkChain() algorithm shown in Algorithm 5. Inputs to the CreateLinkChain() algorithm are the AEP Pattern and a set of source web pages.

The algorithm firstly extracts anchor elements from the source pages and assign them to *anchor-element*$_1$. From these anchor elements, it obtains the next set of pages P_2 by applying the $GetPages()$ function. The extraction of anchor elements from pages in P_2 will be performed. This process repeats until all the required anchor elements are extracted. The algorithm then constructs the link chains using different combinations of link elements.

Algorithm 5 ConstructLinkChain(P_1, $AEPPAttern$)

```
Let $linkchain \leftarrow$ empty
Let $AEPPattern$ be $(AEP_1, \cdots, AEP_m)$
for each $p_1$ in $P_1$ do
  $anchor\text{-}elements_1 \leftarrow$ EvaluateRule($p_1, AEP_1$)
  for (i = 2 to m) do
    $P_i \leftarrow$ GetPages($anchor\text{-}elements_{i-1}$)
    $anchor\text{-}elements_i \leftarrow$ EvaluateRule($P_i, AEP_i$)
  end for
  for each combination of $(< p_1, a_1 >, \cdots, < p_m, a_m >)$ where $p_j \in P_j$, $a_j \in anchor\text{-}element_j$, $1 \leq j \leq m$ do
    Add $(< p_1, a_1 >, \cdots, < p_m, a_m >)$ to $linkchains$
  end for
end for
Return $linkchains$
```

6 Performance Evaluation

We evaluate our proposed algorithms on Yahoo! movie web site. The movie home pages are taken as the source pages for four kinds of relationships: Actor-Of, Directed-by, Produced-by and Written-by. The target pages are the home pages of Actors, Directors, Producers and Writers. Giving at most 4 training link chains and anchor element pattern of the given link chains, the rules of nodes from the anchor element paths are generated for extracting the series of anchor elements from the given pages.

We then apply the learned rules on (1000) movie homepages from the movie web site, and observe whether the extracted link chains actually lead to the correct target pages.

Given a relationship, let the number of link chains existed in the test data set be N_{lr}, the number of extracted link chains be N_{le} and the number of correctly extracted link chains be N_{lc} . The precision P and recall R is defined as follows:

$$P = \frac{N_{lc}}{N_{le}} \tag{1}$$

$$R = \frac{N_{le}}{N_{lr}} \tag{2}$$

The number of example link chains, precision and recall for each relationship are described in Table 1.

Table 1. Link Chain Extraction Results of Yahoo! Movie Web Site

Relationship	# training link chains	# extracted link chains	Precision	Recall
Actor-Of	4	7918	100%	100%
Directed-by	3	970	100%	100%
Produced-by	2	1312	100%	100%
Written-by	1	48	100%	100%

The results was very encouraging as we obtained 100% precision and recall for all relationships for even very small number of training link chains. This results is achieved mainly because the Yahoo! Movie web site is a well structured web site. The extraction rules generated by our algorithms are able to accurately select the anchor elements to be extracted.

7 Conclusion and Future Work

In this paper, we propose to extract link chains, an important piece of information linking pairs of web pages with some relationships. The link chain information can be very useful in several web applications including ontology-based web annotation [12]. By including link chain information in an annotation of relationship instance, users can use it to guide the browsing process for a web site. The experimental results show that our method of extracting link chain information achieves high precision and recall for the web pages in a well structured web site.

We plan to conduct our method on loosely structure web site of other domains. Currently, our method is only suitable for link chains sharing the same anchor element patterns. As part of our future work, we will continue our research in the following directions:

- Enhance our method to extract relationship instances that involve different anchor element patterns.
- Investigate methods for deriving more information from the extracted link chains to facilitate ontological web annotations.

References

1. T. B.-Lee, J. Hendler, and O. Lassila. The Semantic Web, May 2001. URL:http://www.scientificamerican.com/2001/0501issue/0501berners-lee.html.
2. S. Baluja, V. Mittal, and R. Sukthankar. Applying machine learning for high performance named-entity extraction. *Computational Intelligence*, 16, Nov. 2000.
3. Sergey Brin. Extracting patterns and relations from the world wide web. In *WebDB Workshop at 6th International Conference on Extending Database Technology*, 1998.
4. M. Craven, D. DiPasquo, D. Freitag, A. McCallum, T. M. Mitchell, K. Nigam, and S. Slattery. Learning to Construct Knowledge Bases from the World Wide Web. *Artificial Intelligence*, 118(1-2):69–113, 2000.
5. Line Eikvil. Information Extraction from World Wide Web-A Survey. Technical Report 945, Norweigan Computing Center, 1999.
6. F.Ciravegna. Adaptive Information Extraction from Text by Rule Induction and Generalisation. In *Prodeedings of the 17th International Conference on Artificial Intelligence*, Seattle, USA, August 2001.
7. D. Freitag. Information extraction from HTML: Application of a general machine learning approach. In *Proc. of the 15th Conf. on Artificial Intelligence (AAAI-98)*, pages 517–523, 1998.
8. Benjamin Habegger. Multi-pattern wrappers for relation extraction from the Web. In *Proceedings of the Europeen Conference on Artificial Intelligence*, 2002.
9. N. Kushmerick. Wrapper induction: Efficiency and expressiveness. *Artificial Intelligence*, 118(1-2):15–68, 2000.
10. Ion Muslea. Extraction patterns for information extraction tasks: A survey. In *In AAAI-99 Workshop on Machine Learning for Information Extraction*, 1999.
11. Ion Muslea, Steven Minton, and Craig A. Knoblock. Hierarchical wrapper induction for semistructured information sources. *Autonomous Agents and Multi-Agent Systems*, 4(1/2):93–114, 2001.
12. M. M. Naing, E.-P. Lim, and D. H.-L. Goh. Ontology-based Web Annotation Framework for HyperLink Structures. In *Proceedings of the International Workshop on Data Semantics in Web Information Systems*, Singapore, December 2002.
13. S. Soderland. Learning Information Extraction Rules for Semi-structured and Free Text. *Journal of Machine Learning*, 34(1-3):233–272, 1999.
14. N. Sundaresan and J. Yi. Mining the Web for Relations. In *Proceedings of the WWW9 Conference*, pages 699–711, 2000.