
Chapter 9: Structured Data Extraction

Supervised and unsupervised
wrapper generation

Road map

- **Introduction**
- Data Model and HTML encoding
- Wrapper induction
- Automatic Wrapper Generation: Two Problems
- String Matching and Tree Matching
- Multiple Alignments
- Building DOM Trees
- Extraction Given a List Page: Flat Data Records
- Extraction Given a List Page: Nested Data Records
- Extraction Given Multiple Pages
- Summary

Introduction

- A large amount of information on the Web is contained in regularly structured data objects.
 - often data records retrieved from databases.
 - Such Web data records are important: lists of products and services.
 - Applications: e.g.,
 - Comparative shopping, meta-search, meta-query, etc.
 - **We introduce:**
 - Wrapper induction (supervised learning)
 - automatic extraction (unsupervised learning)
-

Two types of data rich pages

■ List pages

- ❑ Each such page contains one or more lists of data records.
- ❑ Each list in a specific region in the page
- ❑ Two types of data records: flat and nested

■ Detail pages

- ❑ Each such page focuses on a single object.
- ❑ But can have a lot of related and unrelated information

CompUSA.com - Product Results - Microsoft Internet Explorer

File Edit View Favorites Tools Help





Back Forward Stop Home Search Favorites Media AutoFill Options

Address http://www.compusa.com/products/products.asp?N=200049&cm_re=A-HPF-Flat+Panel+%28LCD%29

Google Search Web AutoFill Options





Search the Web

Top Sellers

 <p>EN7410 17-inch LCD Monitor, Black/Dark Charcoal</p> <p>\$299.99</p> <p>Add To Cart (Delivery / Pick-Up) Penny Shipping</p> <p>Compare > <</p>	 <p>17-inch LCD Monitor</p> <p>\$249.99</p> <p>Add To Cart (Delivery / Pick-Up) Penny Shipping</p> <p>Compare > <</p>	 <p>AL1714cb 17-inch LCD Monitor, Black</p> <p>\$269.99</p> <p>Add To Cart (Delivery / Pick-Up) Penny Shipping</p> <p>Compare > <</p>	 <p>SyncMaster 712n 17-inch LCD Monitor, Black</p> <p>Was: \$369.99 \$299.99 SAVE \$70 after: \$70.00 mail-in rebate(s)</p> <p>Add To Cart (Delivery / Pick-Up) Penny Shipping</p> <p>Compare > <</p>
---	---	---	---

Page 1 of 6: 1 2 3 4 5 6 Next >>

Sort by: Popularity **Compare**

 <p>EN7410 17-inch LCD Monitor, Black/Dark Charcoal Product Number: 318020 Mfr. Part #: EN7410 Brand: <u>Envision</u></p>	\$299.99	Add To Cart (Delivery / Pick-Up) Penny Shipping	Compare > <
 <p>17-inch LCD Monitor Product Number: 316328 Mfr. Part #: 130611 Brand: <u>Norwood Micro</u></p>	\$249.99	Add To Cart (Delivery / Pick-Up) Penny Shipping	Compare > <
 <p>AL1714cb 17-inch LCD Monitor, Black Product Number: 317993 Mfr. Part #: ET.L1809.031 Brand: <u>Acer</u></p>	\$269.99	Add To Cart (Delivery / Pick-Up) Penny Shipping	Compare > <
 <p>SyncMaster 712n 17-inch LCD Monitor, Black Was: \$369.99</p>			

IN-STORE PICK-UP
CLICK HERE

good guys
high-end entertainment electronics
[click here](#)

CompUSA Service On-Call
Repairs & service in your home or office
[click here](#)

GANASSI RACING
CLICK HERE!

COMPUSA AUCTIONS.com

VISIT OUR BRAND SHOWCASE!

ADVERTISED SPECIALS
CLICK HERE!

start 4 Outlook Express WWW-05 tutorial 3 Microsoft PowerP... CompUSA.com - Prod... 11:57 AM

Canning Tools : Buy professional pressure canner canning jars tin strainer electric canners - C - Microsoft Internet Explorer

File Edit View Favorites Tools Help

Back Forward Stop Home Search Favorites Media Print Mail News RSS Options

Address <http://www.cooking.com/products/shprodli.asp?DeptNo=4000&ClassNo=0420&CurSort=PriceAsc> Go Links

Google Search Web AutoFill Options

[Advanced Search](#)

DEPARTMENTS





- ⋮ [Bakeware](#)
- ⋮ [Barware](#)
- ⋮ [Coffee & Tea](#)
- ⋮ [Cookbooks](#)
- ⋮ [Cook's Tools](#)
- ⋮ [Cookware](#)
- ⋮ [Cutlery](#)
- ⋮ [Furnishings](#)
- ⋮ [Gift Baskets & Sets](#)
- ⋮ [Home Keeping](#)
- ⋮ [Outdoor Living](#)
- ⋮ [Small Appliances](#)
- ⋮ [Storage & Organization](#)
- ⋮ [Tableware](#)
- ⋮ [Clearance](#)

- ⋮ [Corporate Gifts](#)
- ⋮ [Gift Certificates](#)
- ⋮ [Email Offers](#)

View by Brand: [Norpro \(3\)](#) [Ball \(3\)](#) [R.S.V.P. \(1\)](#)
[Back to Basics \(1\)](#)

View Only: [BestSellers](#) [Cooks Catalogue](#)

Sort By: [Product Type \(z-a\)](#) | [Price \(high-low\)](#) | [Customer Reviews \(high-low\)](#)

	<p>Canning Jars by Ball</p> <p>8-oz. Canning Jars, Set of 4 ★★★★★ \$4.95 INFO</p> <p>1-pt. Canning Jars, Set of 4: Blue Gingham \$5.95 BUY</p> <p style="text-align: center;">★★★★★</p>	
	<p>Canning Tools by Norpro</p> <p>12-dia. Canning Rack ★★★★★ \$5.95 BUY</p>	
	<p>Canning Tools by R.S.V.P.</p> <p>6-in. Canning Funnel ★★★★★ \$6.50 BUY</p>	
	<p>Canning Tools by Norpro</p> <p>Canning Strainer and Bag \$8.95 BUY</p>	

start 4 Outlook Exp... WWW-05 tutorial WWW-05 tutorial 3 Microsoft Po... Canning Tools : ... 11:46 AM

Ball Canning Jars, Set of 4 - Cooking.com - Microsoft Internet Explorer

File Edit View Favorites Tools Help

Back Forward Stop Home Search Favorites Media Mail Print Wordpad Links

Address http://www.cooking.com/products/shprodde.asp?SKU=142008

Google Search Web AutoFill Options

SEARCH

Keyword or Item# **GO**


[Advanced Search](#)

DEPARTMENTS

- :: [Bakeware](#)
- :: [Barware](#)
- :: [Coffee & Tea](#)
- :: [Cookbooks](#)
- :: [Cook's Tools](#)
- :: [Cookware](#)
- :: [Cutlery](#)
- :: [Furnishings](#)
- :: [Gift Baskets & Sets](#)
- :: [Home Keeping](#)
- :: [Outdoor Living](#)
- :: [Small Appliances](#)
- :: [Storage & Organization](#)
- :: [Tableware](#)
- :: [Clearance](#)
- :: [Corporate Gifts](#)

Canning Tools > [Canning Jars](#)

Canning Jars, Set of 4 (8-oz.) by Ball



add to my wish list view my wish list

Our Price: \$4.95

OUT OF STOCK

[Email me](#) when in stock

Sku# 142008

NEW GUEST RATINGS

★★★★★

Rated by **2** Reviewers


[See Reviews](#)

[Click image for larger view](#)

Also available in:
1-pt. - [Blue Gingham](#) [Red Gingham](#)


Ball has been synonymous with canning and preserving for decades, so it's no wonder they've added these decorative jars to their collection. These 8-oz. Canning Jars with Gingham Lids (colors may vary) add some fun to the process and color to

RELATED ITEMS



[Clearly Delicious](#)

Elisabeth Lambert Ortiz's treasury of preserved, pickled &...
\$19.96 **BUY**



[Blue Gingham Canning Jars, Set of 4](#)

"Ball" has been synonymous with canning and preserving for decades...
\$5.95 **BUY**

start Outlook Express WWW-05 tutorial Microsoft PowerP... Ball Canning Jars, Set... Internet 11:53 AM

Extraction results



Cabinet Organizers by Copco

9-in. [Round Turntable: White](#) ★★★★★ \$4.95 **BUY**

12-in. [Round Turntable: White](#) ★★★★★ \$7.95 **BUY**

Cabinet Organizers

14.75x9 [Cabinet Organizer \(Non-skid\): White](#) ★★★★★ \$7.95 **BUY**

Cabinet Organizers

22x6 [Cookware Lid Rack](#) ★★★★★ \$19.95 **BUY**

(a). An example page segment

image 1	Cabinet Organizers by Copco	9-in.	Round Turntable: White	*****	\$4.95
image 1	Cabinet Organizers by Copco	12-in.	Round Turntable: White	*****	\$7.95
image 2	Cabinet Organizers	14.75x9	Cabinet Organizer (Non-skid): White	*****	\$7.95
image 3	Cabinet Organizers	22x6	Cookware Lid Rack	*****	\$19.95

(b). Extraction results

Road map

- Introduction
- **Data Model and HTML encoding**
- Wrapper induction
- Automatic Wrapper Generation: Two Problems
- String Matching and Tree Matching
- Multiple Alignments
- Building DOM Trees
- Extraction Given a List Page: Flat Data Records
- Extraction Given a List Page: Nested Data Records
- Extraction Given Multiple Pages
- Summary

The data model

- Most Web data can be modeled as **nested relations**
 - typed objects allowing nested sets and tuples.
- An **instance** of a type T is simply an element of $dom(T)$.
- there is a set of **basic types**, $B = \{B_1, B_2, \dots, B_k\}$. Each B_i is an atomic type, and its domain, denoted by $dom(B_i)$, is a set of constants.
- if T_1, T_2, \dots, T_n are basic or set types, then $[T_1, T_2, \dots, T_n]$ is a **tuple type** with the domain $dom([T_1, T_2, \dots, T_n]) = \{[v_1, v_2, \dots, v_n] \mid v_i \in dom(T_i)\}$;
- if T is a tuple type, then $\{T\}$ is a **set type** with the domain $dom(\{T\})$ being the power set of $dom(T)$.

An example nested tuple type

- *name* (of type *string*),
- *image* (of type *image-file*), and
- *differentSizes* (a *set* type), consists of a set of tuples with the attributes:
 - *size* (of type *string*), and
 - *price* (of type *string*).

```
tuple product ( name:      string;
                 image:    image-file;
                 differentSizes: set ( size:  string;
                                       price: string; ))
```

- Classic flat relations are of un-nested or flat set types.
- Nested relations are of arbitrary set types.

Type tree

- A basic type B_i is a leaf tree,
- A tuple type $[T_1, T_2, \dots, T_n]$ is a tree rooted at a **tuple node** with n sub-trees, one for each T_i .
- A set type $\{T\}$ is a tree rooted at a **set node** with one sub-tree.

Note: attribute names are not included in the type tree.

We introduce a labeling of a type tree, which is defined recursively:

- If a set node is labeled ϕ , then its child is labeled $\phi.0$, a tuple node.
- If a tuple node is labeled ϕ , then its n children are labeled $\phi.1, \dots, \phi.n$.

Instance tree

- An instance (constant) of a basic type is a leaf tree.
- A tuple instance $[v_1, v_2, \dots, v_n]$ forms a tree rooted at a tuple node with n children or sub-trees representing attribute values v_1, v_2, \dots, v_n .
- A set instance $\{e_1, e_2, \dots, e_n\}$ forms a set node with n children or sub-trees representing the set elements e_1, e_2, \dots , and e_n .

Note: A tuple instance is usually called a **data record** in data extraction research.

HTML mark-up encoding of data

- There are no designated tags for each type as HTML was not designed as a data encoding language. Any HTML tag can be used for any type.
- For a tuple type, values (also called **data items**) of different attributes are usually encoded differently to distinguish them and to highlight important items.
- A tuple may be partitioned into several groups or sub-tuples. Each group covers a disjoint subset of attributes and may be encoded differently.

HTML encoding (cont ...)

- for a leaf node of a basic type labeled ϕ , an instance c is encoded with,

$$enc(\phi.c) = OPEN-TAGS\ c\ CLOSE-TAGS,$$

where *OPEN-TAGS* is a sequence of open HTML tags, and *CLOSE-TAGS* is the sequence of corresponding close HTML tags. The number of tags is greater than or equal to 0.

- for a tuple node labeled ϕ of n children or attributes, $[\phi.1, \dots, \phi.n]$, the attributes are first partitioned into h (≥ 1) groups $\langle \phi.1, \dots, \phi.e \rangle$, $\langle \phi.(e+1), \dots, \phi.g \rangle \dots \langle \phi.(k+1), \dots, \phi.n \rangle$ and an instance $[v_1, \dots, v_n]$ of the tuple node is encoded with

$$\begin{aligned} enc(\phi.[v_1, \dots, v_n]) = & OPEN-TAGS_1\ enc(v_1)\ \dots\ enc(v_e)\ CLOSE-TAGS_1 \\ & OPEN-TAGS_2\ enc(v_{e+1})\ \dots\ enc(v_g)\ CLOSE-TAGS_2 \\ & \dots \\ & OPEN-TAGS_h\ enc(v_{k+1})\ \dots\ enc(v_n)\ CLOSE-TAGS_h \end{aligned}$$

where *OPEN-TAGS_i* is a sequence of open HTML tags, and *CLOSE-TAGS_i* is the sequence of corresponding close tags. The number of tags is greater than or equal to 0.

- for a set node labeled ϕ , an non-empty set instance $\{e_1, e_2, \dots, e_n\}$ is encoded with

$$enc(\phi.\{e_1, \dots, e_n\}) = OPEN-TAGS\ enc(e_1)\ \dots\ enc(e_n)\ CLOSE-TAGS$$

More on HTML encoding

- By no means, this mark-up encoding covers all cases in Web pages.
 - In fact, each group of a tuple type can be further divided.
- We must also note that in an actual Web page the encoding may not be done by HTML tags alone.
 - Words and punctuation marks can be used as well.

Restaurant Name: **Good Noodles**

- 205 Willow, *Glen*, Phone 1-773-366-1987
- 25 Oak, *Forest*, Phone (800) 234-7903
- 324 Halsted St., *Chicago*, Phone 1-800-996-5023
- 700 Lake St., *Oak Park*, Phone: (708) 798-0008

Road map

- Introduction
- Data Model and HTML encoding
- **Wrapper induction**
- Automatic Wrapper Generation: Two Problems
- String Matching and Tree Matching
- Multiple Alignments
- Building DOM Trees
- Extraction Given a List Page: Flat Data Records
- Extraction Given a List Page: Nested Data Records
- Extraction Given Multiple Pages
- Summary

Wrapper induction

- Using machine learning to generate extraction rules.
 - The user marks the target items in a few training pages.
 - The system learns extraction rules from these pages.
 - The rules are applied to extract items from other pages.
- Many wrapper induction systems, e.g.,
 - WIEN (Kushmerick et al, IJCAI-97),
 - Softmealy (Hsu and Dung, 1998),
 - Stalker (Muslea et al. Agents-99),
 - BWI (Freitag and Kushmerick, AAAI-00),
 - WL² (Cohen et al. WWW-02).
- We will only focus on **Stalker**, which also has a commercial version, **Fetch**.

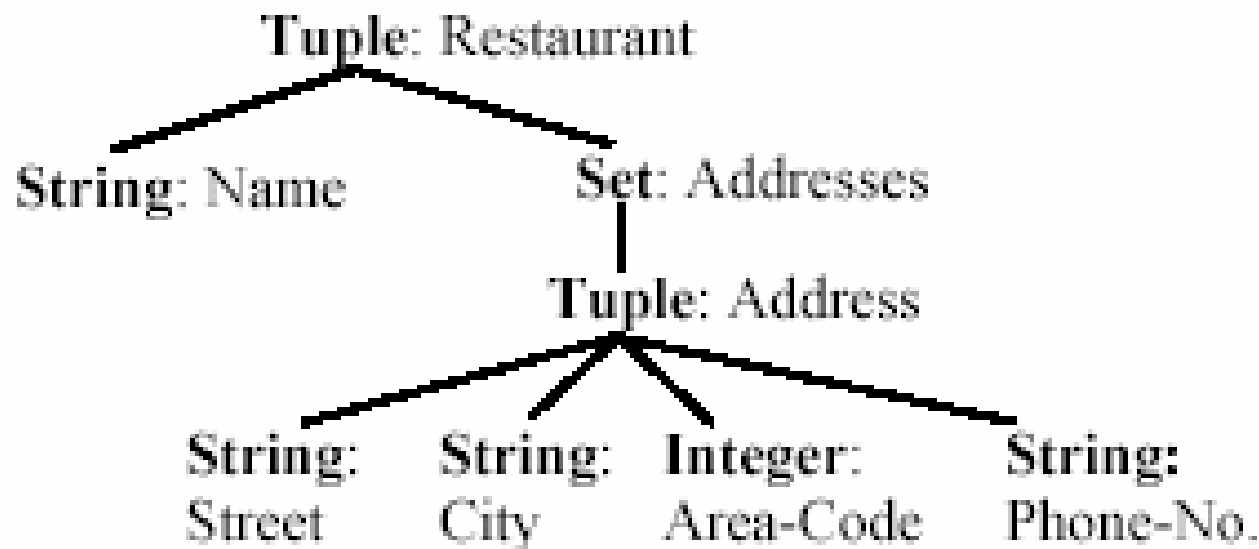
Stalker: A hierarchical wrapper induction system

- Hierarchical wrapper learning
 - Extraction is isolated at different levels of hierarchy
 - This is suitable for nested data records (embedded list)
- Each item is extracted independent of others.
- Each target item is extracted using two rules
 - A **start rule** for detecting the beginning of the target item.
 - A **end rule** for detecting the ending of the target item.

Hierarchical representation: type tree

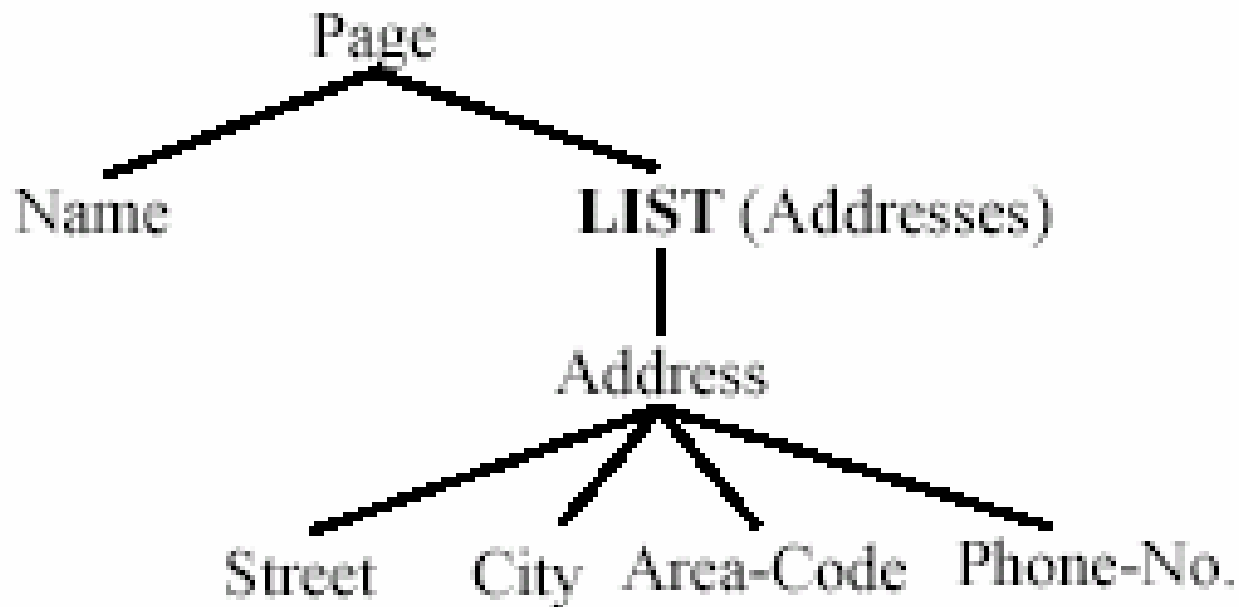
Restaurant Name: **Good Noodles**

- 205 Willow, *Glen*, Phone 1-773-366-1987
- 25 Oak, *Forest*, Phone (800) 234-7903
- 324 Halsted St., *Chicago*, Phone 1-800-996-5023
- 700 Lake St., *Oak Park*, Phone: (708) 798-0008



Data extraction based on EC tree

- The extraction is done using a tree structure called the *EC* tree (**embedded catalog tree**).
- The *EC* tree is based on the type tree above.



- To extract each target item (a node), the wrapper needs a rule that extracts the item from its parent.

Extraction using two rules

- Each extraction is done using two rules,
 - **a start rule** and **a end rule**.
- The start rule identifies the beginning of the node and the end rule identifies the end of the node.
 - This strategy is applicable to both leaf nodes (which represent data items) and list nodes.
- For a list node, **list iteration rules** are needed to break the list into individual data records (tuple instances).

Rules use landmarks

- The extraction rules are based on the idea of **landmarks**.
 - Each landmark is a sequence of *consecutive* tokens.
- Landmarks are used to locate the beginning and the end of a target item.
- Rules use landmarks

An example

- Let us try to extract the restaurant name “Good Noodles”. Rule R1 can identify the beginning :

R1: *SkipTo()* // start rule

- This rule means that the system should start from the beginning of the page and skip all the tokens until it sees the first tag. is a landmark.
- Similarly, to identify the end of the restaurant name, we use:

R2: *SkipTo()* // end rule

```
1: <p> Restaurant Name: <b>Good Noodles</b><br><br>
2: <li> 205 Willow, <i>Glen</i>, Phone 1-<i>773</i>-366-1987</li>
3: <li> 25 Oak, <i>Forest</i>, Phone (800) 234-7903 </li>
4: <li> 324 Halsted St., <i>Chicago</i>, Phone 1-<i>800</i>-996-5023 </li>
5: <li> 700 Lake St., <i>Oak Park</i>, Phone: (708) 798-0008 </li> </p>
```


Rules are not unique

- Note that a rule may not be unique. For example, we can also use the following rules to identify the beginning of the name:

R3: *SkipTo*(Name *_Punctuation_ _HtmlTag_*)

or **R4:** *SkipTo*(Name) *SkipTo*()

- **R3** means that we skip everything till the word “Name” followed by a punctuation symbol and then a HTML tag. In this case, “Name *_Punctuation_ _HtmlTag_*” together is a landmark.
 - *_Punctuation_* and *_HtmlTag_* are **wildcards**.

Extract area codes

1. Identify the entire list of addresses. We can use the start rule *SkipTo*(

), and the end rule *SkipTo*(</p>).
2. Iterate through the list (lines 2-5) to break it into 4 individual records (lines 2 - 5). To identify the beginning of each address, the wrapper can start from the first token of the parent and repeatedly apply the start rule *SkipTo*() to the content of the list. Each successive identification of the beginning of an address starts from where the previous one ends. Similarly, to identify the end of each address, it starts from the last token of its parent and repeatedly apply the end rule *SkipTo*().

Once each address record is identified or extracted, we can extract the area code in it. Due to variations in the format of area codes (some are in italic and some are not), we need to use disjunctions. In this case, the disjunctive start and the end rules are respectively **R5** and **R6**:

R5: either *SkipTo*()
or *SkipTo*(-<i>)

R6: either *SkipTo*())
or *SkipTo*(</i>)

In a disjunctive rule, the disjuncts are applied sequentially until a disjunct can identify the target node.

Learning extraction rules

- Stalker uses sequential covering to learn extraction rules for each target item.
 - In each iteration, it learns a perfect rule that covers as many positive examples as possible without covering any negative example.
 - Once a positive example is covered by a rule, it is removed.
 - The algorithm ends when all the positive examples are covered. The result is an ordered list of all learned rules.

The top level algorithm

```
Algorithm LearnRule(Examples) // Examples: training examples
1   Rule  $\leftarrow \emptyset$  // Rule: the returned rule
2   while Examples  $\neq \emptyset$  do
3     Disjunct  $\leftarrow$  LearnDisjunct(Examples)
4     remove all examples in Examples covered by Disjunct
5     add Disjunct to Rule
6   return Rule
```

Fig. 10. The main learning algorithm based on sequential covering

Example: Extract area codes

E1: 205 Willow, <i>Glen</i>, Phone 1-<i>773</i>-366-1987
E2: 25 Oak, <i>Forest</i>, Phone (800) 234-7903
E3: 324 Halsted St., <i>Chicago</i>, Phone 1-<i>800</i>-996-5023
E4: 700 Lake St., <i>Oak Park</i>, Phone: (708) 798-0008

Fig. 9. Training examples: four addresses with labeled area codes

Learn disjuncts

- BestDisjunct () prefer candidates that have:
- more correct matches
 - accepts fewer false positive
 - fewer wildcards
 - longer end-landmarks

Fig. 13. The function choose the best disjunct

Procedure LearnDisjunct(*Examples*)

```
1  let Seed ∈ Examples be the shortest example
2  Candidates ← GetInitialCandidates(Seed)
3  while Candidates ≠ ∅ do
4      D ← BestDisjunct(Candidates)
5      if D is a perfect disjunct then
6          return D
7      Candidates ← Candidates ∪ Refine(D, Seed)
8      remove D from Candidates
9  return D
```

Fig. 11. This function learns disjuncts

Example

- For the example E2 of Fig. 9, the following candidate disjuncts are generated:

D1: SkipTo(()

D2: SkipTo(*_Punctuation_*)

- D1 is selected by BestDisjunct
- D1 is a perfect disjunct.
- The first iteration of LearnRule() ends. E2 and E4 are removed

The next iteration of LearnRule

- The next iteration of LearnRule() is left with E1 and E3.
- LearnDisjunct() will select E1 as the Seed
Two candidates are then generated:
 - D3: SkipTo(**<i>**)
 - D4: SkipTo(*_HtmlTag_*)
- Both these two candidates match early in the uncovered examples, E1 and E3. Thus, they cannot uniquely locate the positive items.
- Refinement is needed.

Refinement

- To specialize a disjunct by adding more **terminals** to it.
- A **terminal** means a token or one of its matching wildcards.
- We hope the refined version will be able to uniquely identify the positive items in some examples without matching any negative item in any example in E .
- **Two types of refinement**
 - Landmark refinement
 - Topology refinement

Landmark refinement

- **Landmark refinement:** Increase the size of a landmark by concatenating a terminal.

- E.g.,

D5: SkipTo(- <i>)

D6: SkipTo(*_Punctuation_* <i>)

Topology refinement

- **Topology refinement:** Increase the number of landmarks by adding 1-terminal landmarks, i.e., t and its matching wildcards

D7: SkipTo(205) SkipTo(<i>)

D8: SkipTo(Willow) SkipTo(<i>)

D9: SkipTo(.) SkipTo(<i>)

D10: SkipTo(<i>) SkipTo(<i>)

D11: SkipTo(Glen) SkipTo(<i>)

D12: SkipTo(1) SkipTo(<i>)

D13: SkipTo(-) SkipTo(<i>)

D14: SkipTo(Phone) SkipTo(<i>)

D15: SkipTo(*Numeric*) SkipTo(<i>)

D16: SkipTo(*Alphabetic*) SkipTo(<i>)

D17: SkipTo(*Punctuation*) SkipTo(<i>)

D18: SkipTo(*HtmlTag*) SkipTo(<i>)

D19: SkipTo(*Capitalized*) SkipTo(<i>)

D20: SkipTo(*AlphaNum*) SkipTo(<i>)

D21: SkipTo(</i>) SkipTo(<i>)

Refining, specializing

Procedure Refine(D , $Seed$)

```
1    $D$  is a consecutive landmarks  $(l_0, l_1, \dots, l_n)$ ;  
2    $TopologyRefs \leftarrow LandmarkRefs \leftarrow \emptyset$ ;  
3   for  $i = 1$  to  $n$  do      //  $t_0$  or  $t_1$  below may be null  
4     for each sequence  $s = t_0 l_i t_1$  before the target item in  $Seed$  do  
5        $LandmarkRefs \leftarrow LandmarkRefs \cup \{(l_0, \dots, l_{i-1}, t_0 l_i, \dots, l_n)\} \cup$   
          $\{(l_0, \dots, l_{i-1}, x l_i, \dots, l_n) \mid x \text{ is a wildcard that matches } t_0\}$   
          $\cup \{(l_0, \dots, l_i t_1, l_{i+1}, \dots, l_n)\} \cup$   
          $\{(l_0, \dots, l_i x, l_{i+1}, \dots, l_n) \mid x \text{ is a wildcard that matches } t_1\}$   
6     for each token  $t$  between  $l_{i-1}$  and  $l_i$  before the target item in  $Seed$  do  
7        $TopologyRefs \leftarrow TopologyRefs \cup \{(l_0, \dots, l_i, t, l_{i+1}, \dots, l_n)\} \cup$   
          $\{(l_0, \dots, l_i, x, l_{i+1}, \dots, l_n)\} \mid x \text{ is a wildcard that matches } t\}$   
8   return  $TopologyRefs \cup LandmarkRefs$ 
```

Fig. 12. This function refines a disjunct to generate more specialized candidates

The final solution

- We can see that **D5**, **D10**, **D12**, **D13**, **D14**, **D15**, **D18** and **D21** match correctly with E1 and E3 and fail to match on E2 and E4.
- Using BestDisjunct in Fig. 13, **D5** is selected as the final solution as it has longest last landmark (- <i>).
- **D5** is then returned by **LearnDisjunct()**.
- Since all the examples are covered, LearnRule() returns the disjunctive (start) rule either **D1** or **D5**

R7: **either** *SkipTo*(()
 or *SkipTo*(- <i>)

Summary

- The algorithm learns by sequential covering
- It is based on landmarks.
- The algorithm is by no mean the only possible algorithm.
- Many variations are possible. There are entirely different algorithms.
- In our discussion, we used only the *SkipTo()* function in extraction rules.
 - *SkipUntil()* is useful too.

Identifying informative examples

- Wrapper learning needs manual labeling of training examples.
- To ensure accurate learning, a large number of training examples are needed.
- Manual labeling labor intensive and time consuming.
- Is it possible to automatically select (unlabelled) examples that are informative for the user to label.
 - Clearly, examples of the same formatting are of limited use.
 - Examples that represent exceptions are informative as they are different from already labeled examples.

Active learning

- help identify informative unlabeled examples in learning automatically.
1. Randomly select a small subset L of unlabeled examples from U .
 2. Manually label the examples in L , and $U = U - L$.
 3. Learn a wrapper W based on the labeled set L .
 4. Apply W to U to find a set of informative examples L .
 5. Stop if $L = \emptyset$, otherwise go to step 2.

Active learning: co-testing

- Co-testing exploits the fact that there are often multiple ways of extracting the same item.
- Thus, the system can learn different rules, **forward** and **backward rules**, to locate the same item.
- Let us use learning of start rules as an example. The rules learned in Section 8.2.2 are called *forward rules* because they consume tokens from the beginning of the example to the end.
- In a similar way, we can also learn *backward rules* that consume tokens from the end of the example to the beginning.

Co-testing (cont ...)

- Given an unlabeled example, both the forward rule and backward rule are applied.
- If the two rules disagree on the beginning of a target item in the example, this example is given to the user to label.
- **Intuition:** When the two rules agree, the extraction is very likely to be correct.
 - When the two rules do not agree on the example, one of them must be wrong.
 - By giving the user the example to label, we obtain an informative training example.

Wrapper maintenance

- **Wrapper verification:** If the site changes, does the wrapper know the change?
- **Wrapper repair:** If the change is correctly detected, how to automatically repair the wrapper?
- One way to deal with both problems is to learn the characteristic patterns of the target items.
- These patterns are then used to monitor the extraction to check whether the extracted items are correct.

Wrapper maintenance (cont ...)

- **Re-labeling**: If they are incorrect, the same patterns can be used to locate the correct items assuming that the page changes are minor formatting changes.
- **Re-learning**: re-learning produces a new wrapper.
- **Difficult problems**: These two tasks are extremely difficult because it often needs contextual and semantic information to detect changes and to find the new locations of the target items.
- Wrapper maintenance is still an active research area.

Road map

- Introduction
- Data Model and HTML encoding
- Wrapper induction
- **Automatic Wrapper Generation: Two Problems**
- String Matching and Tree Matching
- Multiple Alignments
- Building DOM Trees
- Extraction Given a List Page: Flat Data Records
- Extraction Given a List Page: Nested Data Records
- Extraction Given Multiple Pages
- Summary

Automatic wrapper generation

- Wrapper induction (supervised) has two main shortcomings:
 - It is unsuitable for a large number of sites due to the manual labeling effort.
 - Wrapper maintenance is very costly. The Web is a dynamic environment. Sites change constantly. Since rules learnt by wrapper induction systems mainly use formatting tags, if a site changes its formatting templates, existing extraction rules for the site become invalid.

Unsupervised learning is possible

- Due to these problems, automatic (or unsupervised) extraction has been studied.
- Automatic extraction is possible because data records (tuple instances) in a Web site are usually encoded using a very small number of fixed templates.
- It is possible to find these templates by mining repeated patterns.

Two data extraction problems

- In Sections 8.1.2 and 8.2.3, we described an abstract model of structured data on the Web (i.e., nested relations), and a HTML mark-up encoding of the data model respectively.
- The general problem of data extraction is to recover the hidden schema from the HTML mark-up encoded data.
- We study two extraction problems, which are really quite similar.

Problem 1: Extraction given a single list page

- **Input:** A single HTML string S , which contain k non-overlapping substrings s_1, s_2, \dots, s_k with each s_i encoding an instance of a set type. That is, each s_i contains a collection W_i of m_i (≥ 2) non-overlapping sub-substrings encoding m_i instances of a tuple type.
- **Output:** k tuple types $\sigma_1, \sigma_2, \dots, \sigma_k$, and k collections C_1, C_2, \dots, C_k , of instances of the tuple types such that for each collection C_i there is a HTML encoding function enc_i such that $enc_i: C_i \rightarrow W_i$ is a bijection.

Problem 2: Data extraction given multiple pages

- **Input:** A collection W of k HTML strings, which encode k instances of the same type.
- **Output:** A type σ , and a collection C of instances of type σ , such that there is a HTML encoding enc such that $enc: C \rightarrow W$ is a bijection.

Templates as regular expressions

- A **regular expression** can be naturally used to model the HTML encoded version of a nested type.
- Given an alphabet of symbols Σ and a special token "*#text*" that is not in Σ ,
 - a *regular expression* over Σ is a string over $\Sigma \cup \{\#text, *, ?, |, (,)\}$ defined as follows:

Regular expressions

- The empty string ε and all elements of $\Sigma \cup \{\#text\}$ are regular expressions.
- If A and B are regular expressions, then AB , $(A|B)$ and $(A)?$ are regular expressions, where $(A|B)$ stands for A or B and $(A)?$ stands for $(A| \varepsilon)$.
- If A is a regular expression, $(A)^*$ is a regular expression, where $(A)^*$ stands for ε or A or AA or ...

We also use $(A)^+$ as a shortcut for $A(A)^*$, which can be used to model the set type of a list of tuples. $(A)?$ indicates that A is optional. $(A|B)$ represents a disjunction.

Regular expressions and extraction

- Regular expressions are often employed to represent templates (or encoding functions).
- However, templates can also be represented as string or tree patterns as we will see later.
- Extraction:
 - Given a regular expression, a nondeterministic finite-state automaton can be constructed and employed to match its occurrences in string sequences representing Web pages.
 - In the process, data items can be extracted, which are text strings represented by *#text*.

Road map

- Introduction
- Data Model and HTML encoding
- Wrapper induction
- Automatic Wrapper Generation: Two Problems
- **String Matching and Tree Matching**
- Multiple Alignments
- Building DOM Trees
- Extraction Given a List Page: Flat Data Records
- Extraction Given a List Page: Nested Data Records
- Extraction Given Multiple Pages
- Summary

Some useful algorithms

- The key is to finding the encoding template from a collection of encoded instances of the same type.
- A natural way to do this is to detect repeated patterns from HTML encoding strings.
- **String edit distance** and **tree edit distance** are obvious techniques for the task. We describe these techniques.

String edit distance

- String edit distance: the most widely used string comparison technique.
- The **edit distance** of two strings, s_1 and s_2 , is defined as the minimum number of *point mutations* required to change s_1 into s_2 , where a point mutation is one of:
 - (1) change a letter,
 - (2) insert a letter, and
 - (3) delete a letter.

String edit distance (definition)

Assume we are given two strings s_1 and s_2 . The following recurrence relations define the edit distance, $d(s_1, s_2)$, of two strings s_1 and s_2 :

$$\begin{aligned}d(\varepsilon, \varepsilon) &= 0 && // \varepsilon \text{ represents an empty string} \\d(s, \varepsilon) &= d(\varepsilon, s) = |s| && // |s| \text{ is the length of string } s \\d(s_1+ch_1, s_2+ch_2) &= \min(d(s_1, s_2) + r(ch_1, ch_2), d(s_1+ch_1, s_2) + 1, \\& \quad d(s_1, s_2+ch_2) + 1)\end{aligned}$$

where ch_1 and ch_2 are the last characters of s_1 and s_2 respectively, and $r(ch_1, ch_2) = 0$ if $ch_1 = ch_2$; $r(ch_1, ch_2) = 1$, otherwise.

Dynamic programming

We can use a two-dimensional matrix, $m[0..|s_1|, 0..|s_2|]$ to hold the edit distances. The low right corner cell $m(|s_1|+1, |s_2|+1)$ will furnish the required value of the edit distance $d(s_1, s_2)$.

$$m[0, 0] = 0$$

$$m[i, 0] = i, \quad i = 1, 2, \dots, |s_1|$$

$$m[0, j] = j, \quad j = 1, 2, \dots, |s_2|$$

$$m[i, j] = \min(m[i-1, j-1] + r(s_1[i], s_2[j]), m[i-1, j] + 1, m[i, j-1] + 1),$$

where $i = 1, 2, \dots, |s_1|, j = 1, 2, \dots, |s_2|$, and $r(s_1[i], s_2[j]) = 0$ if $s_1[i] = s_2[j]$; $r(s_1[i], s_2[j]) = 1$, otherwise.

An example

Example 1: We want to compute the edit distance and find the alignment of the following two strings:

S_1 : X G Y X Y X Y X
 S_2 : X Y X Y X Y T X

- The edit distance matrix and back trace path

- alignment

S_1 : X G Y X Y X Y - X
 S_2 : X - Y X Y X Y T X

	S_1	X	G	Y	X	Y	X	Y	X
S_2	0	1	2	3	4	5	6	7	8
X	1	0	1	2	3	4	5	6	7
Y	2	1	1	1	2	3	4	5	6
X	3	2	2	2	1	2	3	4	5
Y	4	3	3	2	2	1	2	3	4
X	5	4	4	3	2	2	1	2	3
Y	6	5	5	4	3	2	2	1	2
T	7	6	6	5	4	3	3	2	2
X	8	7	7	6	5	4	3	3	2

Tree Edit Distance

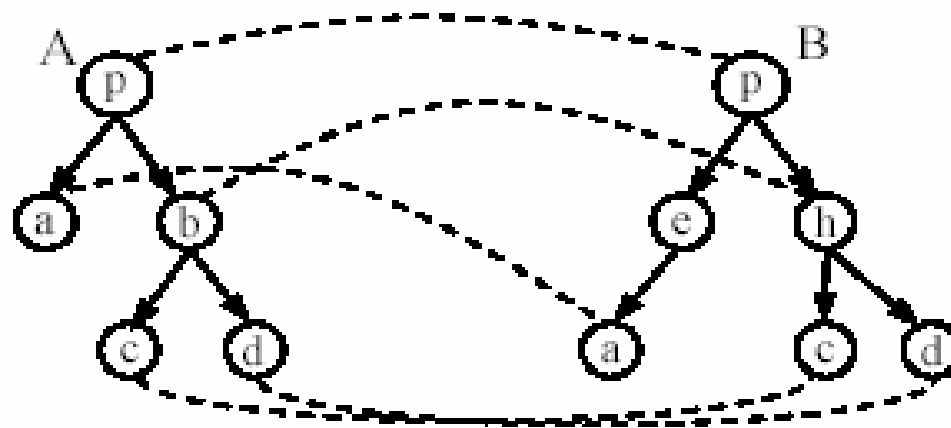
- Tree edit distance between two trees A and B (*labeled ordered rooted trees*) is the cost associated with the minimum set of operations needed to transform A into B .
- The set of operations used to define tree edit distance includes three operations:
 - node removal,
 - node insertion, and
 - node replacement.A cost is assigned to each of the operations.

Definition

Let X be a tree and let $X[i]$ be the i th node of tree X in a preorder walk of the tree. A *mapping* M between a tree A of size n_1 and a tree B of size n_2 is a set of ordered pairs (i, j) , one from each tree, satisfying the following conditions for all $(i_1, j_1), (i_2, j_2) \in M$:

- (1) $i_1 = i_2$ iff $j_1 = j_2$;
- (2) $A[i_1]$ is on the left of $A[i_2]$ iff $B[j_1]$ is on the left of $B[j_2]$;
- (3) $A[i_1]$ is an ancestor of $A[i_2]$ iff $B[j_1]$ is an ancestor of $B[j_2]$.

Intuitively, the definition requires that each node appears no more than once in a mapping and the order among siblings and the hierarchical relation among nodes are both preserved. Fig. 16 shows a mapping example.



Simple tree matching

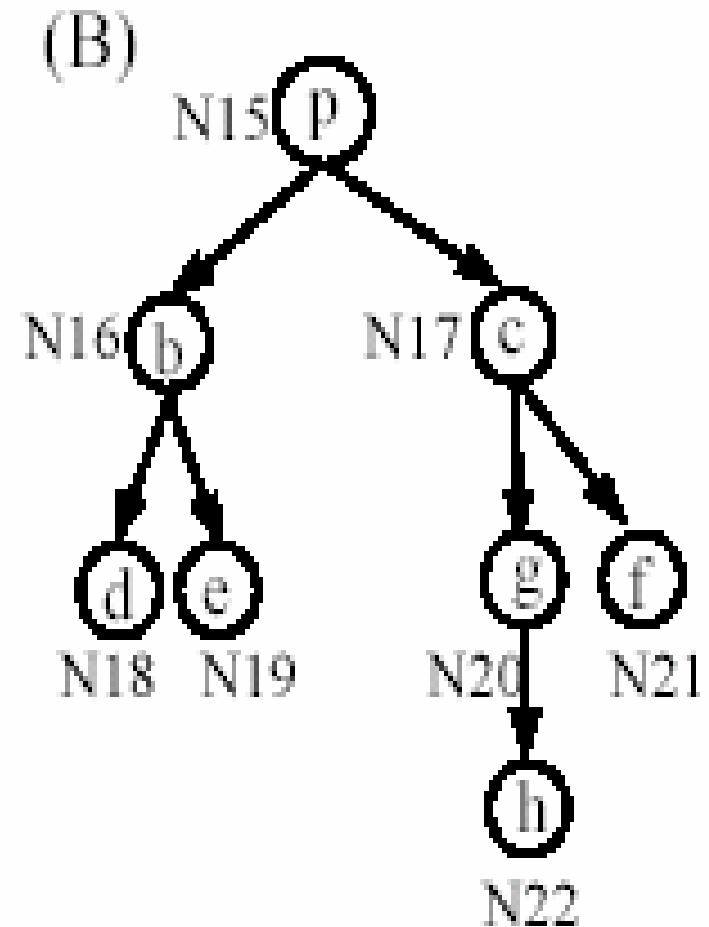
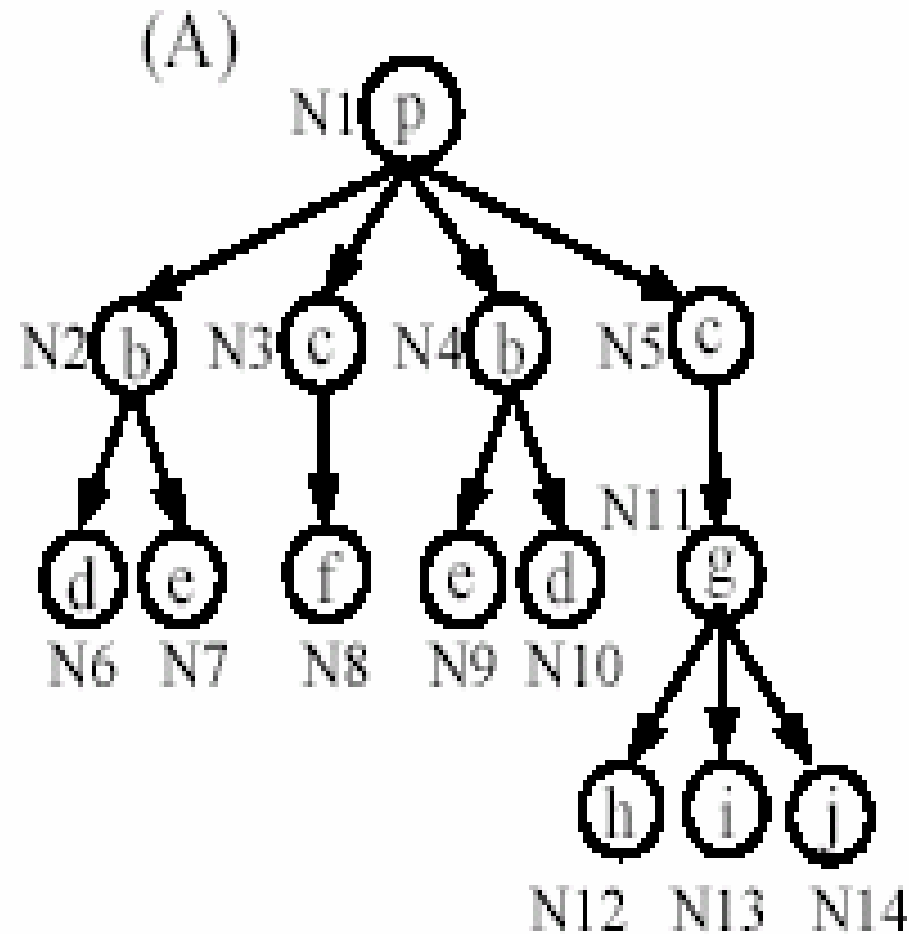
- In the general setting,
 - mapping can cross levels, e.g., node a in tree A and node a in tree B .
 - Replacements are also allowed, e.g., node b in A and node h in B .
- We describe a restricted matching algorithm, called **simple tree matching** (STM), which has been shown quite effective for Web data extraction.
 - STM is a top-down algorithm.
 - Instead of computing the edit distance of two trees, it evaluates their similarity by producing the maximum matching through dynamic programming.

Simple Tree Matching algo

Algorithm: STM(A, B)

1. **if** the roots of the two trees A and B contain distinct symbols **then**
2. **return** (0)
3. **else** $m :=$ the number of first-level sub-trees of A ;
4. $n :=$ the number of first-level sub-trees of B ;
5. Initialization: $M[i, 0] := 0$ for $i = 0, \dots, m$;
 $M[0, j] := 0$ for $j = 0, \dots, n$;
6. **for** $i = 1$ to m **do**
7. **for** $j = 1$ to n **do**
8. $M[i, j] := \max(M[i, j-1], M[i-1, j], M[i-1, j-1] + W[i, j]);$
 where $W[i, j] = \text{STM}(A_i, B_j)$
9. **end-for**
10. **end-for**
11. **return** ($M[m, n] + 1$)
12. **end-if**

An example



Road map

- Introduction
- Data Model and HTML encoding
- Wrapper induction
- Automatic Wrapper Generation: Two Problems
- String Matching and Tree Matching
- **Multiple Alignments**
- Building DOM Trees
- Extraction Given a List Page: Flat Data Records
- Extraction Given a List Page: Nested Data Records
- Extraction Given Multiple Pages
- Summary

Multiple alignment

- Pairwise alignment is not sufficient because a web page usually contain more than one data records.
- We need multiple alignment.
- We discuss two techniques
 - Center Star method
 - Partial tree alignment.

Center star method

- This is a classic technique, and quite simple. It commonly used for multiple string alignments, but can be adopted for trees.
- Let the set of strings to be aligned be S . In the method, a string s_c that minimizes,

$$\sum_{s_i \in S} d(s_c, s_i) \quad (3)$$

- is first selected as the **center string**. $d(sc, si)$ is the distance of two strings.
- The algorithm then iteratively computes the alignment of rest of the strings with sc .

The algorithm

CenterStar(S)

1. choose the center star s_c using Equation (3);
2. initialize the multiple sequence alignment M that contains only s_c ;
4. **for** each s in $S - \{s_c\}$ **do**
5. let c^* be the aligned version of s_c in M ;
6. let s' and c^{*} be the optimally aligned strings of s and c^* ;
7. add aligned strings s' and c^{*} into the multiple alignment M ;
8. add spaces to each string in M , except, s' and c^{*} , at locations where new spaces are added to c^*
9. **endfor**
10. return multiple string alignment M

An example

Example 2: We have three strings, i.e., $S = \{ABC, XBC, XAB\}$. ABC is selected as the center string s_c . Let us align the other strings with ABC.

Iteration 1: Align c^* ($= s_c$) with $s = XBC$:

c^* :	A	B	C				
s :	X	B	C				
Update M :	A	B	C	→	A	B	C
					X	B	C

Iteration 2: Align c^* with $s = XAB$:

c^* :	-	A	B	C				
s :	X	A	B	-				
Update M :	A	B	C	→	-	A	B	C
	X	B	C		-	X	B	C
					X	A	B	-

The shortcomings

- Assume there are k strings in S and all strings have length n , finding the center takes $O(k^2n^2)$ time and the iterative pair-wise alignment takes $O(kn^2)$ time. Thus, the overall time complexity is $O(k^2n^2)$.

For our data extraction task, this method has two shortcomings:

1. the algorithm runs slowly for pages containing many data records and/or data records containing many tags (i.e., long strings) because finding the center string needs $O(k^2n^2)$ time.
2. if the center string (or tree) does not have a particular data item, other data records that contain the same data item may not be aligned properly. For example, the letter X's in the last two strings (in bold) are not aligned in the final result, but they should.

Shortcomings (cont ...)

- Giving the cost of 1 for “changing a letter” in edit distance is problematic (e.g., A and X in the first and second strings in the final result) because of optional data items in data records.
- The problem can be partially dealt with by disallowing “changing a letter” (e.g., giving it a larger cost). However, this introduces another problem.
- For example, if we align only ABC and XBC, it is not clear which of the following alignment is better.

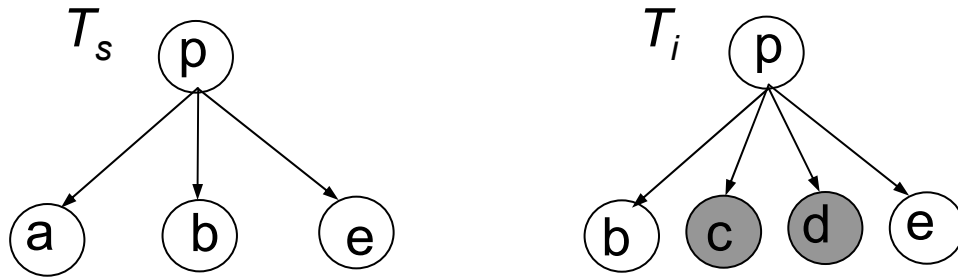
(1) A – B C
– X B C

(2) – A B C
X – B C

The partial tree alignment method

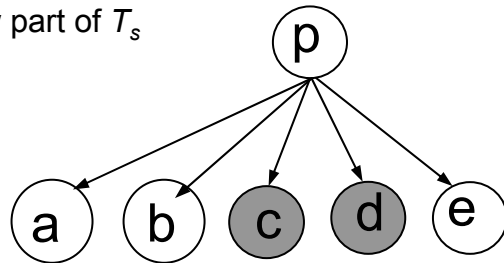
- **Choose a seed tree:** A seed tree, denoted by T_s , is picked with the maximum number of data items.
 - The seed tree is similar to center string, but without the $O(k^2n^2)$ pair-wise tree matching to choose it.
 - **Tree matching:**
 - For each unmatched tree T_i ($i \neq s$),
 - match T_s and T_i .
 - Each pair of matched nodes are linked (aligned).
 - For each unmatched node n_j in T_i do
 - expand T_s by inserting n_j into T_s if a position for insertion can be uniquely determined in T_s .
- The expanded seed tree T_s is then used in subsequent matching.

Partial tree alignment of two trees

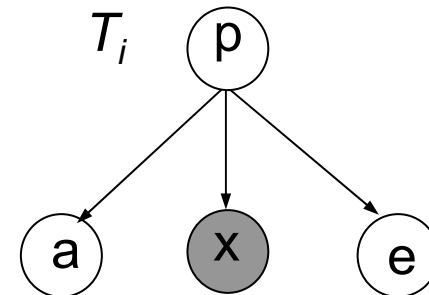
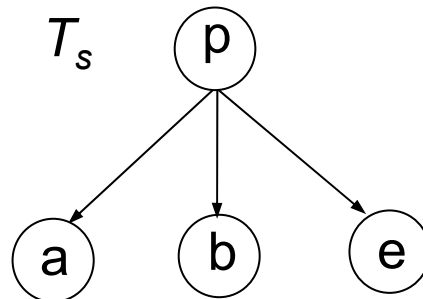


Insertion is possible

New part of T_s



Insertion is not possible



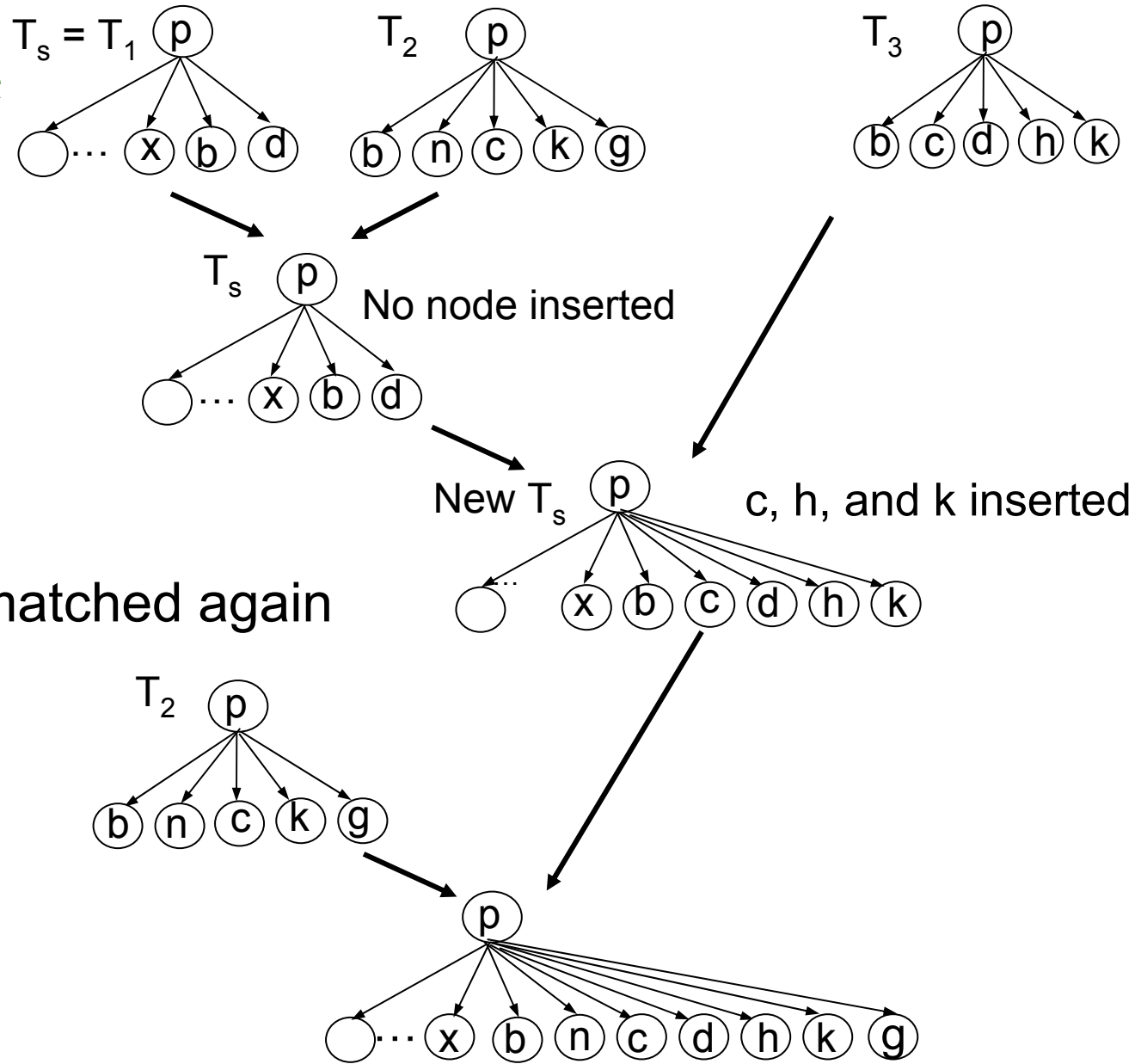
Partial alignment of two trees

Algorithm PartialTreeAlignment(S)

1. Sort trees in S in descending order of the number of unaligned data items;
2. $T_s \leftarrow$ the first tree (which is the largest) and delete it from S ;
3. $R \leftarrow \emptyset$;
4. **while** ($S \neq \emptyset$) **do**
5. $T_i \leftarrow$ select and delete next tree from S ; // follow the sorted order
6. STM(T_s, T_i); // tree matching
7. AlignTrees(T_s, T_i); // based on the result from line 6
8. **if** T_i is not completely aligned with T_s **then**
9. **if** InsertIntoSeed(T_s, T_i) **then** // True: some insertions are done
10. $S = S \cup R$;
11. $R \leftarrow \emptyset$
12. **endif**;
13. **if** there are still unaligned items in T_i that are not inserted into T_s **then**
14. $R \leftarrow R \cup \{T_i\}$
15. **endif**;
16. **endif**;
17. **endwhile**;
18. Output data fields from each T_i to a data table based on the alignment results.

Fig. 21. The partial tree alignment algorithm

A complete example



Output Data Table

	...	x	b	n	c	d	h	k	g
T_1	...	1	1			1			
T_2			1	1	1			1	1
T_3			1		1	1	1	1	

Road map

- Introduction
- Data Model and HTML encoding
- Wrapper induction
- Automatic Wrapper Generation: Two Problems
- String Matching and Tree Matching
- Multiple Alignments
- **Building DOM Trees**
- Extraction Given a List Page: Flat Data Records
- Extraction Given a List Page: Nested Data Records
- Extraction Given Multiple Pages
- Summary

Building DOM trees

- We now start to talk about actual data extraction.
- The usual first step is to build a DOM tree (tag tree) of a HTML page.
 - Most HTML tags work in pairs. Within each corresponding tag-pair, there can be other pairs of tags, resulting in a nested structure.
 - Building a DOM tree from a page using its HTML code is thus natural.
- In the tree, each pair of tags is a **node**, and the nested tags within it are the **children** of the node.

Two steps to build a tree

- **HTML code cleaning:**

- Some tags do not require closing tags (e.g., , <hr> and <p>) although they have closing tags.
- Additional closing tags need to be inserted to ensure all tags are balanced.
- Ill-formatted tags need to be fixed. One popular program is called **Tidy**, which can be downloaded from <http://tidy.sourceforge.net/>.

- **Tree building:** simply follow the nested blocks of the HTML tags in the page to build the DOM tree. It is straightforward.

Building tree using tags & visual cues

- Correcting errors in HTML can be hard.
- There are also dynamically generated pages with scripts.
- Visual information comes to the rescue.
- As long as a browser can render a page correct, a tree can be built correctly.
 - Each HTML element is rendered as a rectangle.
 - Containments of rectangles representing nesting.

An example

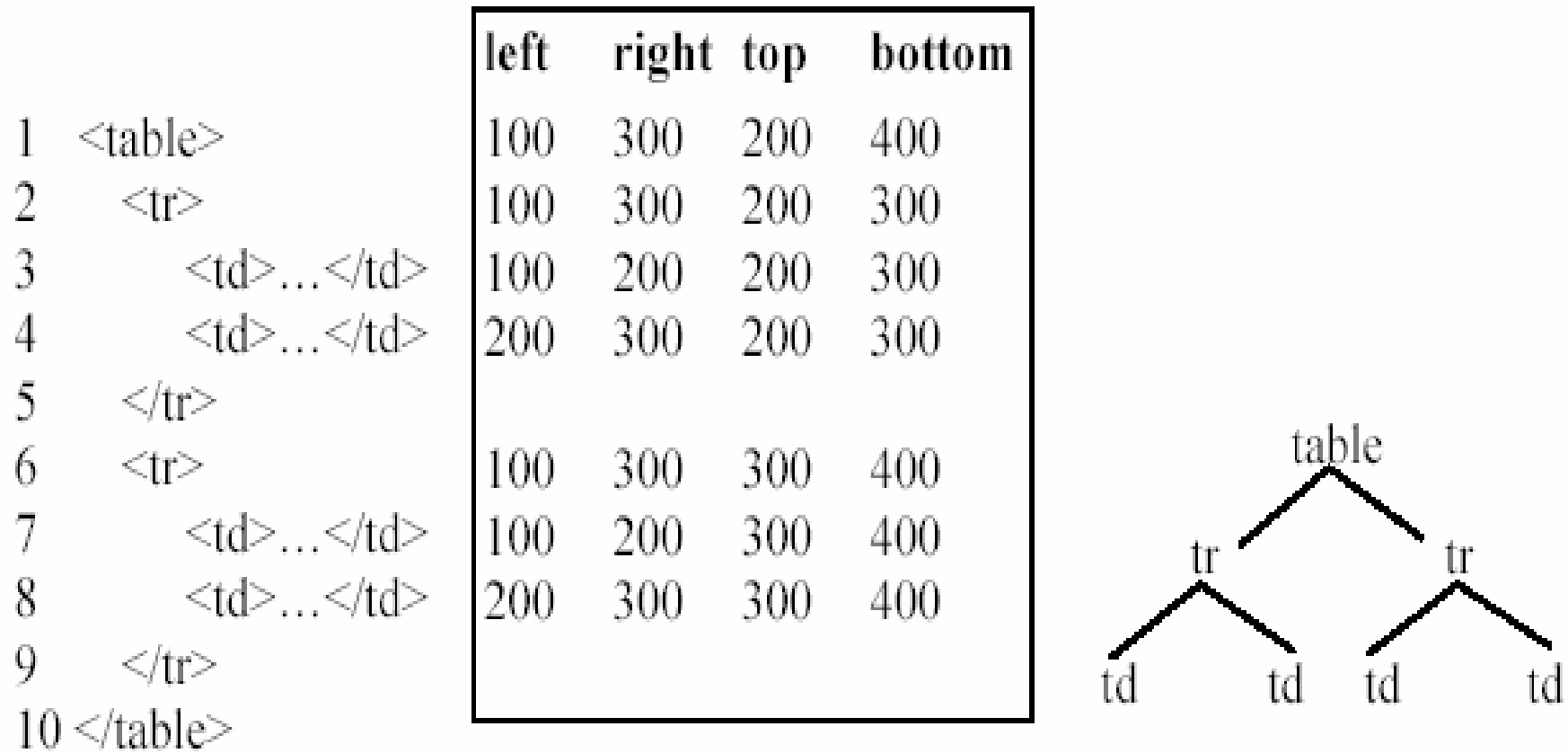


Fig. 23. A HTML code segment, boundary coordinates and the resulting tree

Road map

- Introduction
- Data Model and HTML encoding
- Wrapper induction
- Automatic Wrapper Generation: Two Problems
- String Matching and Tree Matching
- Multiple Alignments
- Building DOM Trees
- **Extraction Given a List Page: Flat Data Records**
- Extraction Given a List Page: Nested Data Records
- Extraction Given Multiple Pages
- Summary

Extraction Given a List Page: Flat Data Records

- Given a single list page with multiple data records,
 - Automatically segment data records
 - Extract data from data records.
- Since the data records are flat (no nested lists), string similarity or tree matching can be used to find similar structures.
 - Computation is a problem
 - A data record can start anywhere and end anywhere

Two important observations

- **Observation 1:** A group of data records that contains descriptions of a set of similar objects are typically presented in a contiguous region of a page and are formatted using similar HTML tags. Such a region is called a **data region**.
- **Observation 2:** A set of data records are formed by some child sub-trees of the same parent node.

An example

1.



Customer Rating:



[Apple iBook Notebook M8600LL/A \(600-MHz PowerPC G3, 128 MB RAM, 20 GB hard drive\)](#)

Buy new: **\$1,194.00**

Usually ships in 1 to 2 days

Best use: (what's this?)	Business: ●●●●○	Portability: ●●●●○	Desktop Replacement: ●●●●○	Entertainment: ●●●●○
--	-----------------	--------------------	----------------------------	----------------------

600 MHz PowerPC G3, 128 MB SRAM, 20 GB Hard Disk, 24x CD-ROM, AirPort ready, and Mac OS X, Mac OS X, Mac OS 9.2, Quick Time, iPhoto, iTunes 2, iMovie 2, AppleWorks, Microsoft IE

2.



Customer Rating:



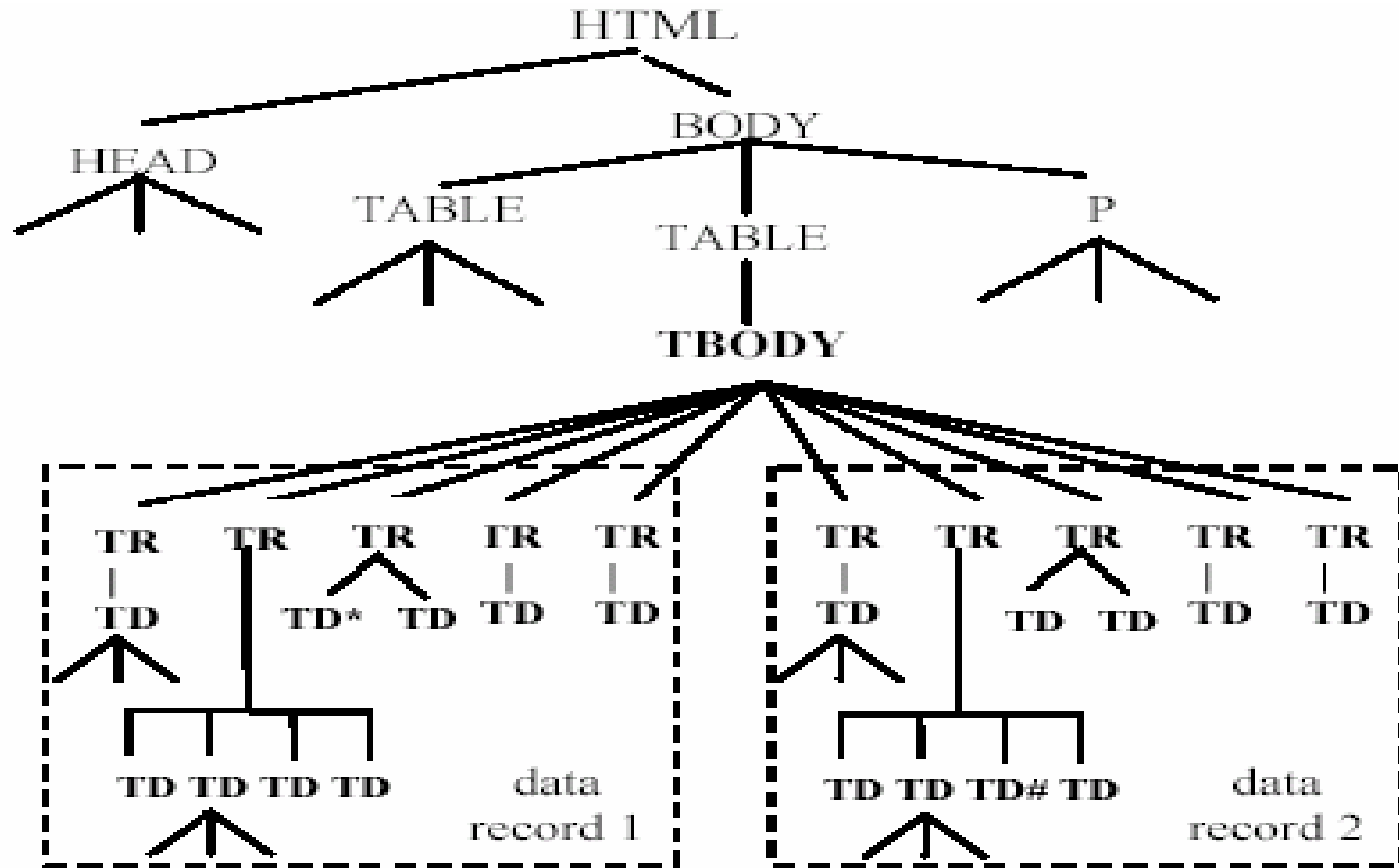
[Apple Powerbook Notebook M8591LL/A \(667-MHz PowerPC G4, 256 MB RAM, 30 GB hard drive\)](#)

Buy new: **\$2,399.99**

Best use: (what's this?)	Portability: ●●●●○	Desktop Replacement: ●●●●○	Entertainment: ●●●●○
--	--------------------	----------------------------	----------------------

667 MHz PowerPC G4, 256 MB SDRAM, 30 GB Ultra ATA Hard Disk, 24x (read), 8x (write) CD-RW, 8x; included via combo drive DVD-ROM, and Mac OS X, QuickTime, iMovie 2, iTunes(6), Microsoft Internet Explorer, Microsoft Outlook Express, ...

The DOM tree



The Approach

Given a page, three steps:

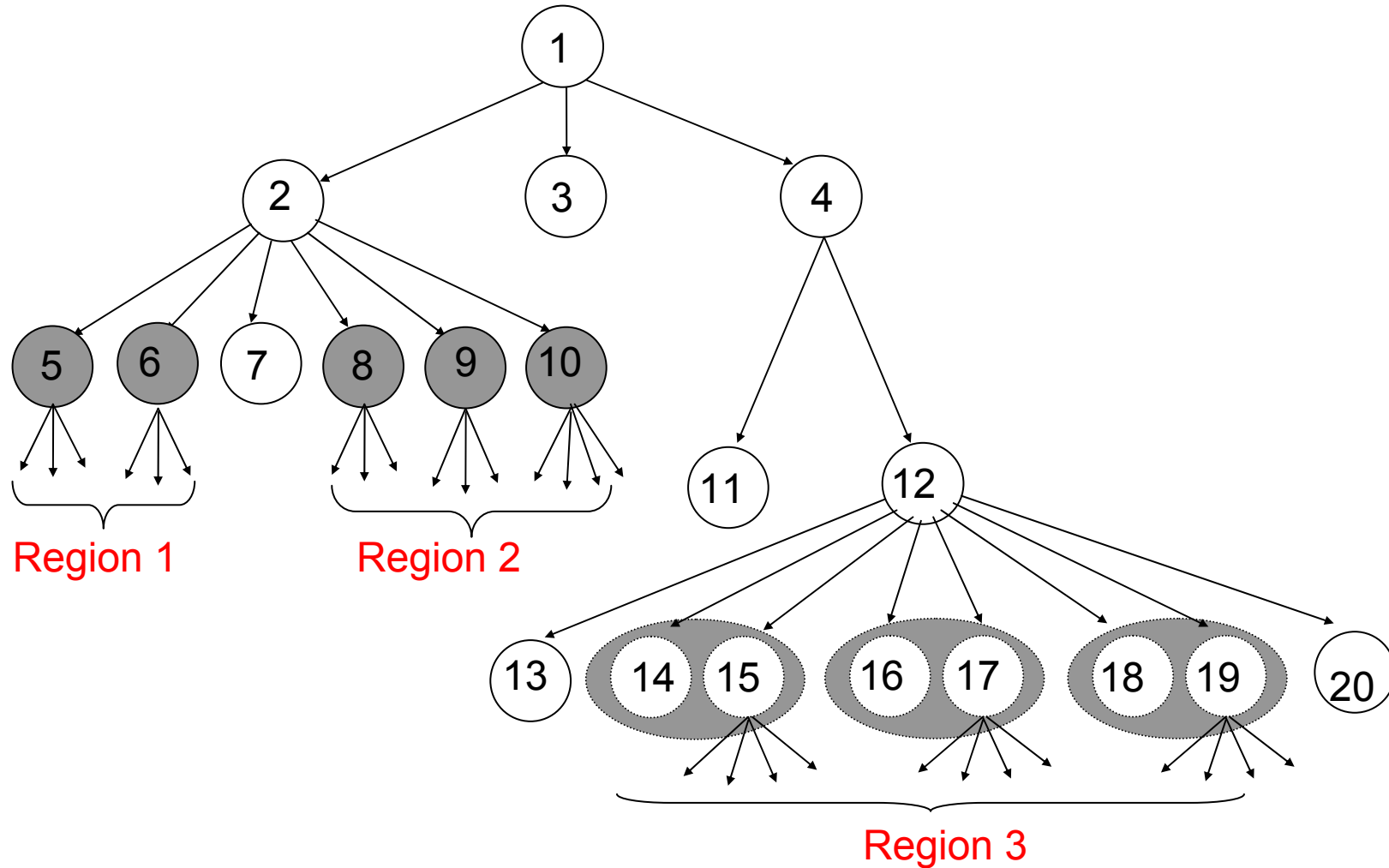
- Building the HTML Tag Tree
 - Erroneous tags, unbalanced tags, etc
- Mining Data Regions
 - Spring matching or tree matching
- Identifying Data Records

Rendering (or visual) information is very useful
in the whole process

Mining a set of similar structures

- **Definition:** A *generalized node* (a *node combination*) of length r consists of r ($r \geq 1$) nodes in the tag tree with the following two properties:
 - the nodes all have the same parent.
 - the nodes are adjacent.
- **Definition:** A *data region* is a collection of two or more generalized nodes with the following properties:
 - the generalized nodes all have the same parent.
 - the generalized nodes all have the same length.
 - the generalized nodes are all adjacent.
 - the similarity between adjacent generalized nodes is greater than a **fixed threshold**.

Mining Data Regions



Mining data regions

- We need to find where each generalized node starts and where it ends.
 - perform string or tree matching
- Computation is not a problem anymore
 - Due to the two observations, we only need to perform comparisons among the children nodes of a parent node.
 - Some comparisons done for earlier nodes are the same as for later nodes (see the example below).

Comparison

We use Fig. 27 to illustrate the comparison process. Fig. 27 has 10 nodes below a parent node p . We start from each node and perform string (or tree) comparison of all possible combinations of component nodes. Let the maximum number of components that a generalized node can have be 3 in this example.

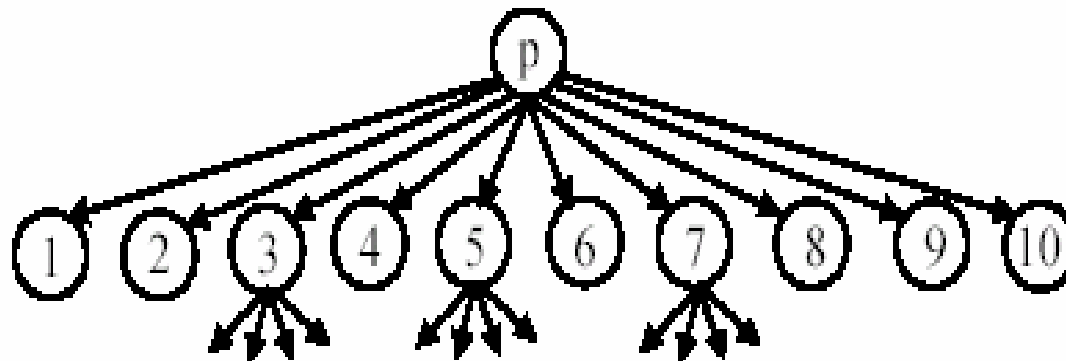


Fig. 27. Combination and comparison

Comparison (cont ...)

Start from node 1: We compute the following string comparisons.

- (1, 2), (2, 3), (3, 4), (4, 5), (5, 6), (6, 7), (7, 8), (8, 9), (9, 10)
- (1-2, 3-4), (3-4, 5-6), (5-6, 7-8), (7-8, 9-10)
- (1-2-3, 4-5-6), (4-5-6, 7-8-9)

(1, 2) means that the tag string of node 1 is compared with the tag string of node 2. The tag string of a node includes all the tags of the subtree of the node. (1-2, 3-4) means that the combined tag string of nodes 1 and 2 is compared with the combined tag string of nodes 3 and 4.

Start from node 2: We only compute:

- (2-3, 4-5), (4-5, 6-7), (6-7, 8-9)
- (2-3-4, 5-6-7), (5-6-7, 8-9-10)

The MDR algorithm

Algorithm $\text{MDR}(Node, K, T)$

```
1  if  $\text{TreeDepth}(Node) \geq 3$  then
2     $\text{CombComp}(Node.Children, K)$ ;
3     $DataRegions \leftarrow \text{IdenDRs}(Node, K, T)$ ;
4    if  $(\text{UncoveredNodes} \leftarrow Node.Children - \bigcup_{DR \in DataRegions} DR) \neq \emptyset$  then
5      for each  $ChildNode \in \text{UncoveredNodes}$  do
6         $DataRegions \leftarrow DataRegions \cup \text{MDR}(ChildNode, K, T)$ ;
7    return  $DataRegions$ 
8  else return  $\emptyset$ 
```

Fig. 28. The overall algorithm

Find data records from generalized nodes

- A generalized node may not represent a data record.
- In the example on the right, each row is found as a generalized node.
- This step needs to identify each of the 8 data records.
 - Not hard
 - We simply run the MDR algorithm given each generalized node as input
- There are some complications (read the notes)



2. Extract Data from Data Records

- Once a list of data records is identified, we can align and extract data items from them.
- Approaches (align multiple data records):
 - Multiple string alignment
 - Many ambiguities due to pervasive use of table related tags.
 - Multiple tree alignment (partial tree alignment)
 - Together with visual information is effective

Generating extraction patterns and data extraction

- Once data records in each data region are discovered, we align them to produce an extraction pattern that can be used to extract data from the current page and also other pages that use the same encoding template.
- **Partial tree alignment algorithm** is just for the purpose.
- Visual information can help in various ways (read the notes)

Road map

- Introduction
- Data Model and HTML encoding
- Wrapper induction
- Automatic Wrapper Generation: Two Problems
- String Matching and Tree Matching
- Multiple Alignments
- Building DOM Trees
- Extraction Given a List Page: Flat Data Records
- **Extraction Given a List Page: Nested Data Records**
- Extraction Given Multiple Pages
- Summary

Extraction Given a List Page: Nested Data Records

- **We now deal with the most general case**
 - Nested data records
- **Problem with the previous method**
 - not suitable for nested data records, i.e., data records containing nested lists.
 - Since the number of elements in the list of each data record can be different, using a fixed threshold to determine the similarity of data records will not work.

Solution idea

- The problem, however, can be dealt with as follows.
 - Instead of traversing the DOM tree top down, we can traverse it post-order.
 - This ensures that nested lists at lower levels are found first based on repeated patterns before going to higher levels.
 - When a nested list is found, its records are **collapsed** to produce a single template.
 - This template replaces the list of nested data records.
- When comparisons are made at a higher level, the algorithm only sees the template. Thus it is treated as a flat data record.

The NET algorithm

Algorithm NET(*Root*, τ)

```
1  TraverseAndMatch(Root,  $\tau$ );
2  for each top level node Node whose children have aligned data records do
3    PutDataInTables(Node);
4  endfor
```

TraverseAndMatch (*Node*, τ)

```
1  if Depth(Node)  $\geq 3$  then
2    for each Child  $\in$  Node.Children do
3      TraverseAndMatch(Child,  $\tau$ );
4    endfor
5    Match(Node,  $\tau$ );
6  endif
```

Fig. 31. The overall NET algorithm

The MATCH algorithm

- It performs tree matching on child sub-trees of *Node* and template generation. τ is the threshold for a match of two trees to be considered sufficiently similar.

Match(*Node*, τ)

1 *Children* \leftarrow *Node.Children*;

2 **while** *Children* $\neq \emptyset$ **do**

3 *ChildFirst* \leftarrow select and remove the first child from *Children*;

4 **for each** *ChildR* in *Children* \leftarrow *Children* $- \{ChildFirst\}$ **do**

5 **if** **TreeMatch**(*ChildFirst*, *ChildR*) $> \tau$ **then**

6 AlignAndLink();

7 *Children* \leftarrow *Children* $- \{ChildR\}$

8 **endfor**

9 **if** some alignments (or links) have been made with *ChildFirst* **then**

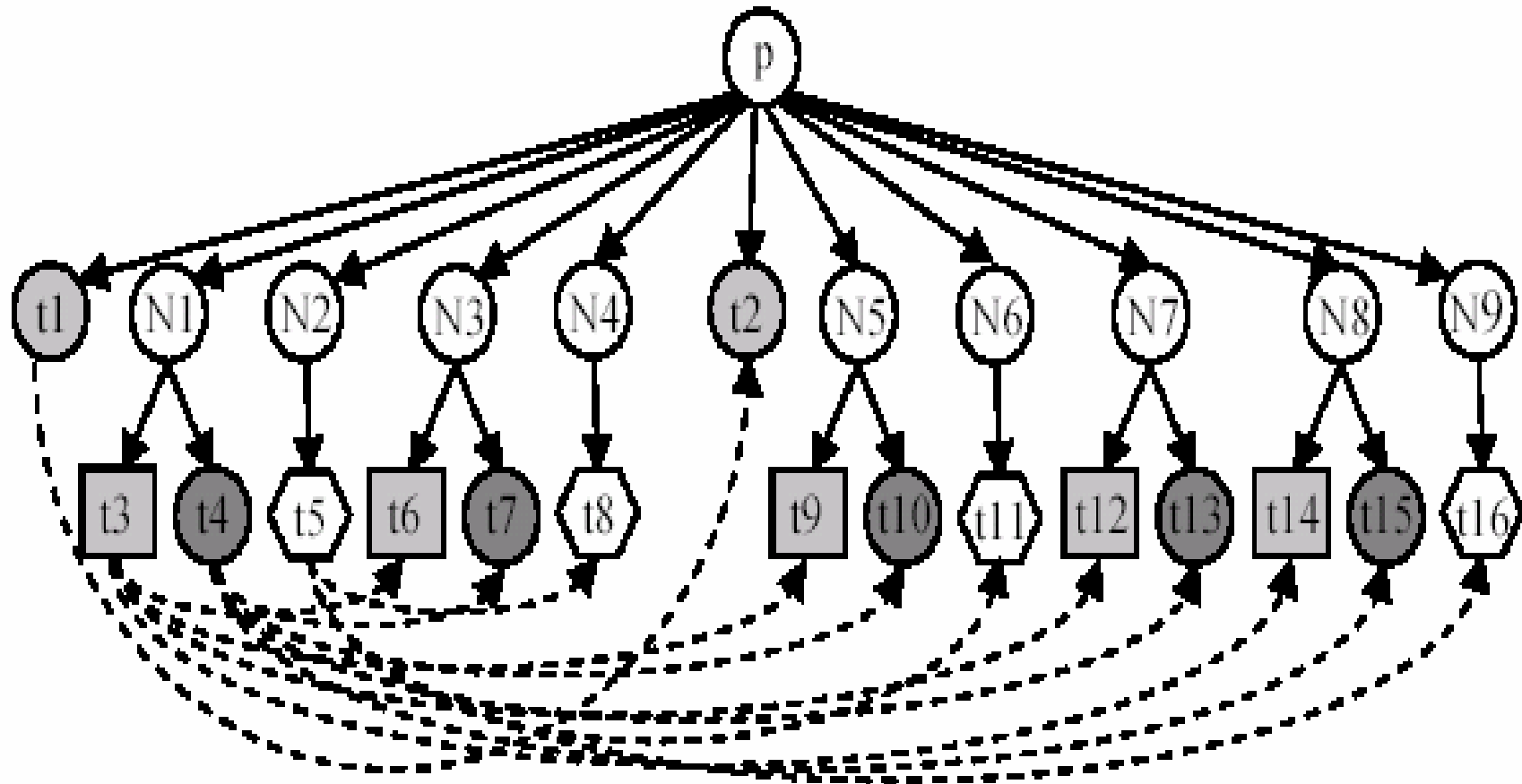
10 GenNodeTemplate(*ChildFirst*)

11 **endwhile**

12 **If** consecutive child nodes in *Children* are aligned **then**

13 GenRecordTemplate(*Node*)

An example



GenNodeTemplate

- It generates a **node template** for all the nodes (including their sub-trees) that match *ChildFirst*.
 - It first gets the set of matched nodes *ChildRs*
 - then calls `PartialTreeAlignment` to produce a template which is the final seed tree.
- **Note:** `AlignAndLink` aligns and links all matched data items in *ChildFirst* and *ChildR*.

GenRecordPattern

- This function produces a regular expression pattern for each data record.
- This is a grammar induction problem.
- Grammar induction in our context is to infer a regular expression given a finite set of positive and negative example strings.
 - However, we only have a single positive example. Fortunately, structured data in Web pages are usually highly regular which enables heuristic methods to generate “simple” regular expressions.
 - We need to make some assumptions

Assumptions

- Three assumptions
 - The nodes in the first data record at each level must be complete.
 - The first node of every data record at each level must be present.
 - Nodes within a flat data record (no nesting) do not match one another.
- On the Web, these are not strong assumptions. In fact, they work well in practice.

Generating NFA

GenRecordPattern(Node)

- 1 *String* \leftarrow Assign a distinctive symbol to each set of matched children of *Node*;
- 2 create an NFA $N = (Q, \Sigma, \delta, q_0, F)$, where $Q = \{q_0\}$, q_0 is the start state, Σ is the symbol set containing all symbols appeared in *String*, δ is the transition function, $F = \emptyset$ is the set of accept states;
- 3 $q_c \leftarrow q_0$; // q_c is the current state
- 4 **for** each symbol s in *String* in sequence **do**
- 5 **if** \exists a transition $\delta(q_c, s) = q_n$ **then**
- 6 $q_c \leftarrow q_n$ // transit to the next state;
- 7 **else if** $\exists \delta(q_i, s) = q_j$, where $q_i, q_j \in Q$ **then** // s appeared before
- 8 **if** $\exists \delta(q_f, \varepsilon) = q_i$, where $\delta(q_i, s) = q_j$ and $f \geq c$ **then**
- 9 TransitTo(q_c, q_j);
- 10 **else** create a transition $\delta(q_c, \varepsilon) = q_i$, where $\delta(q_i, s) = q_j$
- 11 $q_c \leftarrow q_j$
- 12 **else** create a new state q_{c+1} and a transition $\delta(q_c, s) = q_{c+1}$;
- 13 $Q = Q \cup \{q_{c+1}\}$;
- 14 $q_c \leftarrow q_{c+1}$
- 15 **if** s is the last symbol in *String* **then**
- 16 Assign the state with the largest subscript the accept state q_r , $F = \{q_r\}$;
- 17 TransitTo(q_c, q_r);
- 18 **endfor**
- 19 generate a regular expression pattern based on the NFA N ;
- 20 Substitute all the node templates into the regular expression pattern.

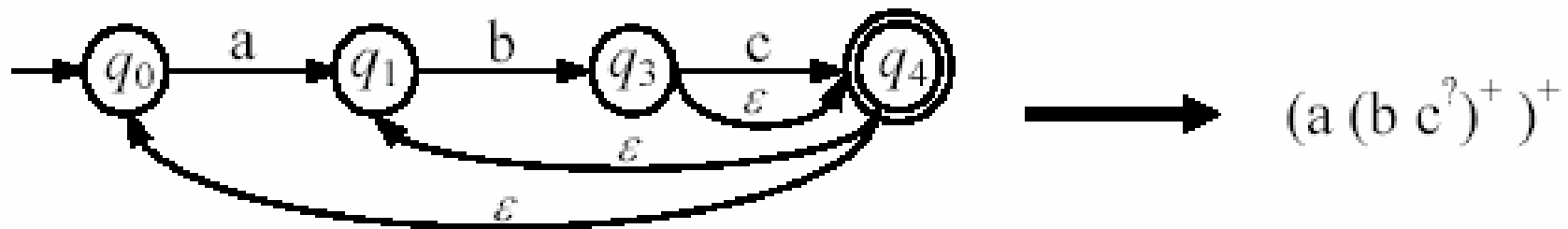
An example

- Line 1 simply produces a string for generating a regular expression.

For our example, we obtain the following:

	t1	N1	N2	N3	N4	t2	N5	N6	N7	N8	N9
<i>String:</i>	a	b	c	b	c	a	b	c	b	b	c

- The final NFA and the regular expression



Example (cont ...)

- We finally obtain the following

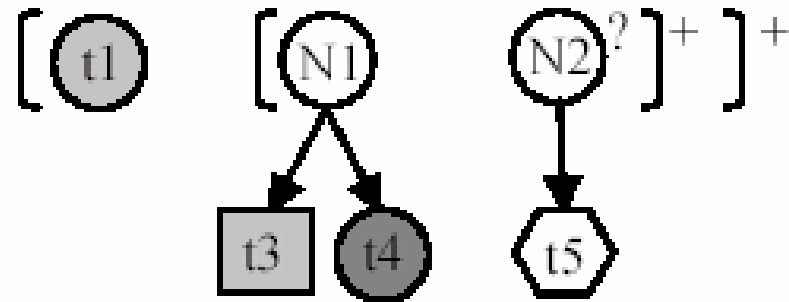


Fig. 36. The regular expression pattern produced from Fig. 33.

Data extraction

- The function PutDataInTables (line 3 of NET) outputs data items in a table, which is simple after the data record templates are found.
- An example

t1	t3	t4	t5
t1	t6	t7	t8
t2	t9	t10	t11
t2	t12	t13	
t2	t14	t15	t16

Fig. 37. The output data table for the example in Fig. 33.

An more complete example

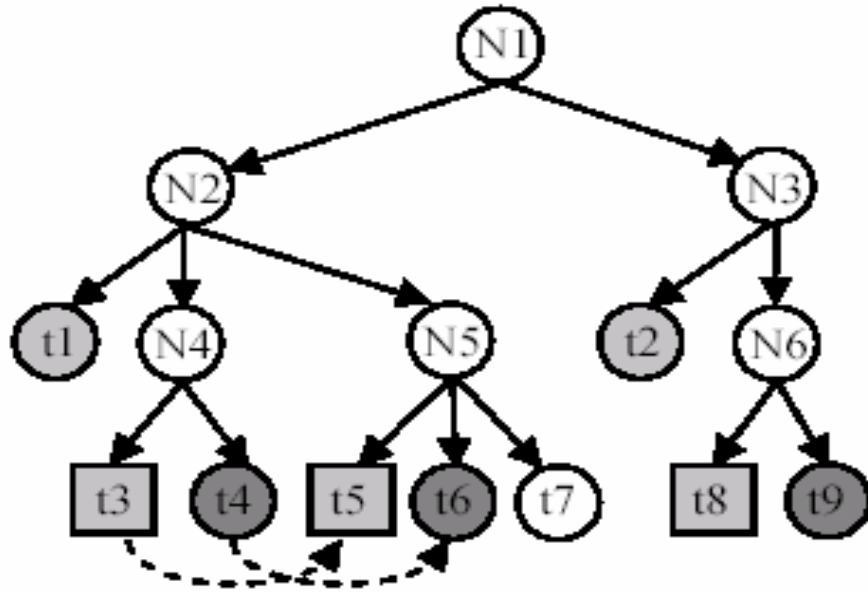


Fig. 38. Aligned data nodes are linked

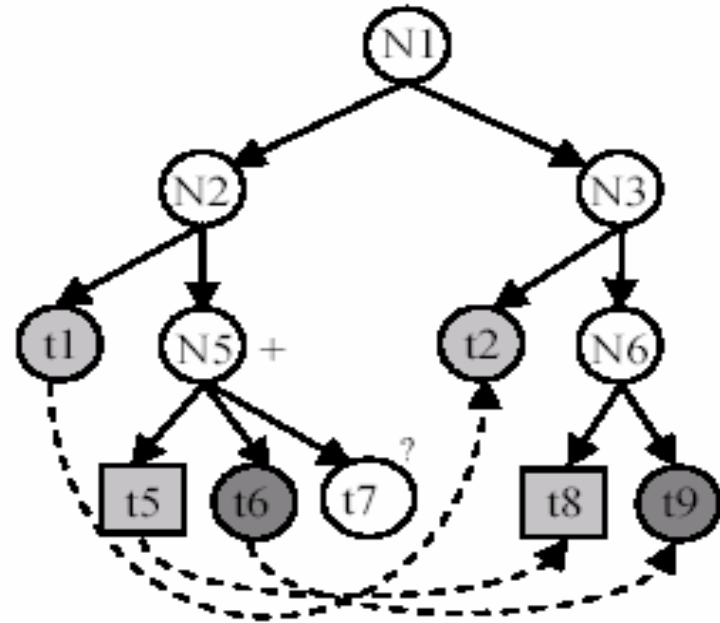


Fig. 39. Alignment after collapsing

t1	t3	t4	
t1	t5	t6	t7
t2	t8	t9	

Fig. 40. The output data table for the example in Fig. 38

Road map

- Introduction
- Data Model and HTML encoding
- Wrapper induction
- Automatic Wrapper Generation: Two Problems
- String Matching and Tree Matching
- Multiple Alignments
- Building DOM Trees
- Extraction Given a List Page: Flat Data Records
- Extraction Given a List Page: Nested Data Records
- **Extraction Given Multiple Pages**
- Summary

Extraction Given Multiple Pages

- We now discuss the **second extraction problem** described in Section 8.3.1.
 - Given multiple pages with the same encoding template, the system finds patterns from them to extract data from other similar pages.
 - The collection of input pages can be a set of list pages or detail pages.
- Below, we first see how the techniques described so far can be applied in this setting, and then
 - describe a technique specifically designed for this setting.

Using previous techniques

- **Given a set of list pages**

- The techniques described in previous sections are for a single list page.

- They can clearly be used for multiple list pages.

- If multiple list pages are available, they may help improve the extraction.

- For example, templates from all input pages may be found separately and merged to produce a single refined pattern.

- This can deal with the situation where a single page may not contain the complete information.

Given a set of detail pages

- In some applications, one needs to extract data from detail pages as they contain more information on the object. Information in list pages are quite brief.
- For extraction, we can treat each detail page as a data record, and extract using the algorithm described in Section 8.7 and/or Section 8.8.
 - For instance, to apply the NET algorithm, we simply create a rooted tree as the input to NET as follows:
 - create an artificial root node, and
 - make the DOM tree of each page as a child sub-tree of the artificial root node.

Difficulty with many detail pages

- Although a detail page focuses on a single object, the page may contain a large amount of “noise”, at the top, on the left and right and at the bottom.
- Finding a set of detail pages automatically is non-trivial.
 - List pages can be found automatically due to repeated patterns in each page.
 - Some domain heuristics may be used to find detail pages.
 - We can find list pages and go to detail pages from there

An example page (a lot of noise)

WEEKLY AD STORE LOCATOR HELP RESEARCH CENTER CONTACT US

Thousands of Possibilities | GET YOURS

BEST BUY

SEARCH Entire Site FOR GO

COMPUTERS MUSIC, MOVIES, GAMES & TOYS ELECTRONICS CAMERAS & CAMCORDERS HOME & APPLIANCES PHONES & COMMUNICATIONS OFFICE PRODUCTS

Best Buy > Digital Cameras > Point & Shoot Digital Cameras > Product Info

Nikon Coolpix 5.1-Megapixel Digital Camera
Model: S1

Easy-to-use scene modes, Red-Eye Fix and a large LCD screen make it simple to take digital pictures you'll be proud to share.

VIEW MORE PHOTOS
Reg. Price: \$329.99
On Sale Now:
[See price in cart](#)

ADD TO CART

✓ On Sale
✓ \$50 Epson Printer Mail-In Rebate
Special Best Buy
✓ [imaging lab card offer!](#)
✓ Free Shipping

Shipping: Usually ships in 1 business day
[Estimate arrival time.](#)

Store Pickup: Available at most stores [Select preferred store availability](#)

ADD TO WISH LIST

>> **Learn more about Nikon's In-Camera Red-Eye Fix!** (Flash demo) [Protect Your Investment](#)

- 5.1-megapixel CCD captures high-resolution images up to 2592 x 1944 pixels
- 3x optical/4x digital/12x total zoom; nonextending Zoom-Nikkor ED lens
- 2.5" TFT-LCD monitor with brightness adjustment

More Options

- [Protect your investment with a Service Plan.](#)
- [Do you have all the accessories you need?](#)
- [Compare with products in this price range.](#)

Product Specs Accessories Product Research

Product Features

- Shooting modes include auto, 4 scene-assist, scene, Best Shot Selector, blur warning, date imprint and self-timer

Welcome. Please [create an account](#) or [Sign in.](#)

Your Cart
Contains 0 items
Subtotal: **\$0.00**
[View Cart](#) | [Checkout](#)

Your Account
[Best Buy Credit](#)
[Order Status](#)
[Wish List](#)
[Gift Cards](#)

NO INTEREST FINANCING
[Apply and Buy Today](#)

SAVE BIG
visit our online **Outlet** store

The RoadRunner System

- Given a set of positive examples (multiple sample pages). Each contains one or more data records.
- From these pages, generate a wrapper as a union-free regular expression (i.e., no disjunction).
- Support nested data records.

The approach

- To start, a sample page is taken as the wrapper.
- The wrapper is then refined by solving mismatches between the wrapper and each sample page, which generalizes the wrapper.
 - A mismatch occurs when some token in the sample does not match the grammar of the wrapper.

Different types of mismatches and wrapper generalization

- Text string mismatches: indicate data fields (or items).
 - Tag mismatches: indicate
 - optional elements, or
 - Iterators, list of repeated patterns
 - Mismatch occurs at the beginning of a repeated pattern and the end of the list.
 - Find the last token of the mismatch position and identify some candidate repeated patterns from the wrapper and sample by searching forward.
 - Compare the candidates with upward portion of the sample to confirm.
-

- Wrapper (initially Page 1):

```

01: <HTML>
02: Books of:
03: <B>
04:   John Smith
05: </B>
06: <UL>
07:   <LI>
08-10:   <I>Title:</I>
11:     DB Primer
12:   </LI>
13:   <LI>
14-16:   <I>Title:</I>
17:     Comp. Sys.
18:   </LI>
19: </UL>
20: </HTML>

```

- Sample (Page 2):

```

01: <HTML>
02: Books of:
03: <B>
04:   Paul Jones
05: </B>
06: <IMG src=.../>
07: <UL>
08:   <LI>
09-11:   <I>Title:</I>
12:     XML at Work
13:   </LI>
14:   <LI>
15-17:   <I>Title:</I>
18:     HTML Scripts
19:   </LI>
20:   <LI>
21-23:   <I>Title:</I>
24:     Javascript
25:   </LI>
26: </UL>
27: </HTML>

```

parsing

string mismatch (#PCDATA)

tag mismatch (?)

string mismatch (#PCDATA)

string mismatch (#PCDATA)

tag mismatch (+)

terminal tag search and square matching

- Wrapper after solving mismatches:

```

<HTML>Books of:<B>#PCDATA</B>
 ( <IMG src=.../> )?
<UL>
 ( <LI><I>Title:</I>#PCDATA</LI> )+
</UL></HTML>

```

Computation issues

- The match algorithm is exponential in the input string length as it has to explore all different alternatives.
- Heuristic pruning strategies are used to lower the complexity.
 - Limit the space to explore
 - Limit backtracking
 - Pattern (iterator or optional) cannot be delimited on either side by an optional pattern (the expressiveness is reduced).

Many other issues in data extraction

- Extraction from other pages.
 - Disjunction or optional
 - A set type or a tuple type
 - Labeling and Integration
- (Read the notes)

Road map

- Introduction
- Data Model and HTML encoding
- Wrapper induction
- Automatic Wrapper Generation: Two Problems
- String Matching and Tree Matching
- Multiple Alignments
- Building DOM Trees
- Extraction Given a List Page: Flat Data Records
- Extraction Given a List Page: Nested Data Records
- Extraction Given Multiple Pages
- **Summary**

Summary

Wrapper induction

■ Advantages:

- ❑ Only the target data are extracted as the user can label only data items that he/she is interested in.
- ❑ Due to manual labeling, there is no integration issue for data extracted from multiple sites as the problem is solved by the user.

■ Disadvantages:

- ❑ It is not scalable to a large number of sites due to significant manual efforts. Even finding the pages to label is non-trivial.
- ❑ Wrapper maintenance (verification and repair) is very costly if the sites change frequently.

Summary (cont ...)

Automatic extraction

■ Advantages:

- It is scalable to a huge number of sites due to the automatic process.
- There is little maintenance cost.

■ Disadvantages:

- It may extract a large amount of unwanted data because the system does not know what is interesting to the user. Domain heuristics or manual filtering may be needed to remove unwanted data.
- Extracted data from multiple sites need integration, i.e., their schemas need to be matched.

Summary (cont...)

- In terms of extraction accuracy, it is reasonable to assume that wrapper induction is more accurate than automatic extraction. However, there is no reported comparison.
- Applications
 - Wrapper induction should be used in applications in which the number of sites to be extracted and the number of templates in these sites are not large.
 - Automatic extraction is more suitable for large scale extraction tasks which do not require accurate labeling or integration.
- **Still an active research area.**