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:

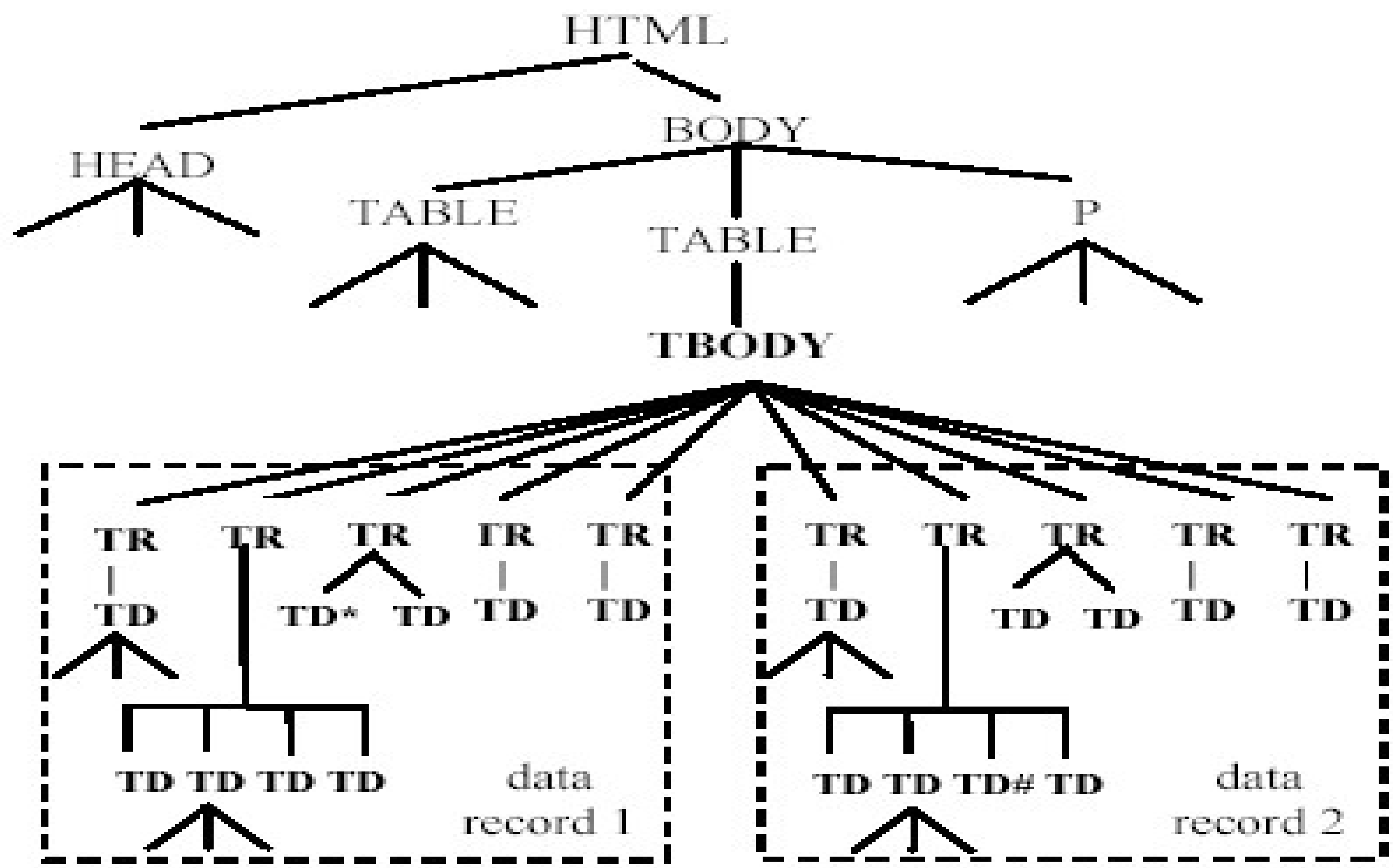
Apple Powerbook Notebook M8591LL/A (667-MHz PowerPC G4, 256 MB RAM, 30 GB hard drive)

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



HTML
HEAD
BODY
TABLE
TABLE
P
TBODY
TR TR TR TR TR
TD
TD* TD
TD
TD
TD TD TD TD
data
record 1
TR TR TR TR TR
TD
TD TD
TD
TD
TD TD TD# TD
data
record 2

The Approach

Given a page, three steps:

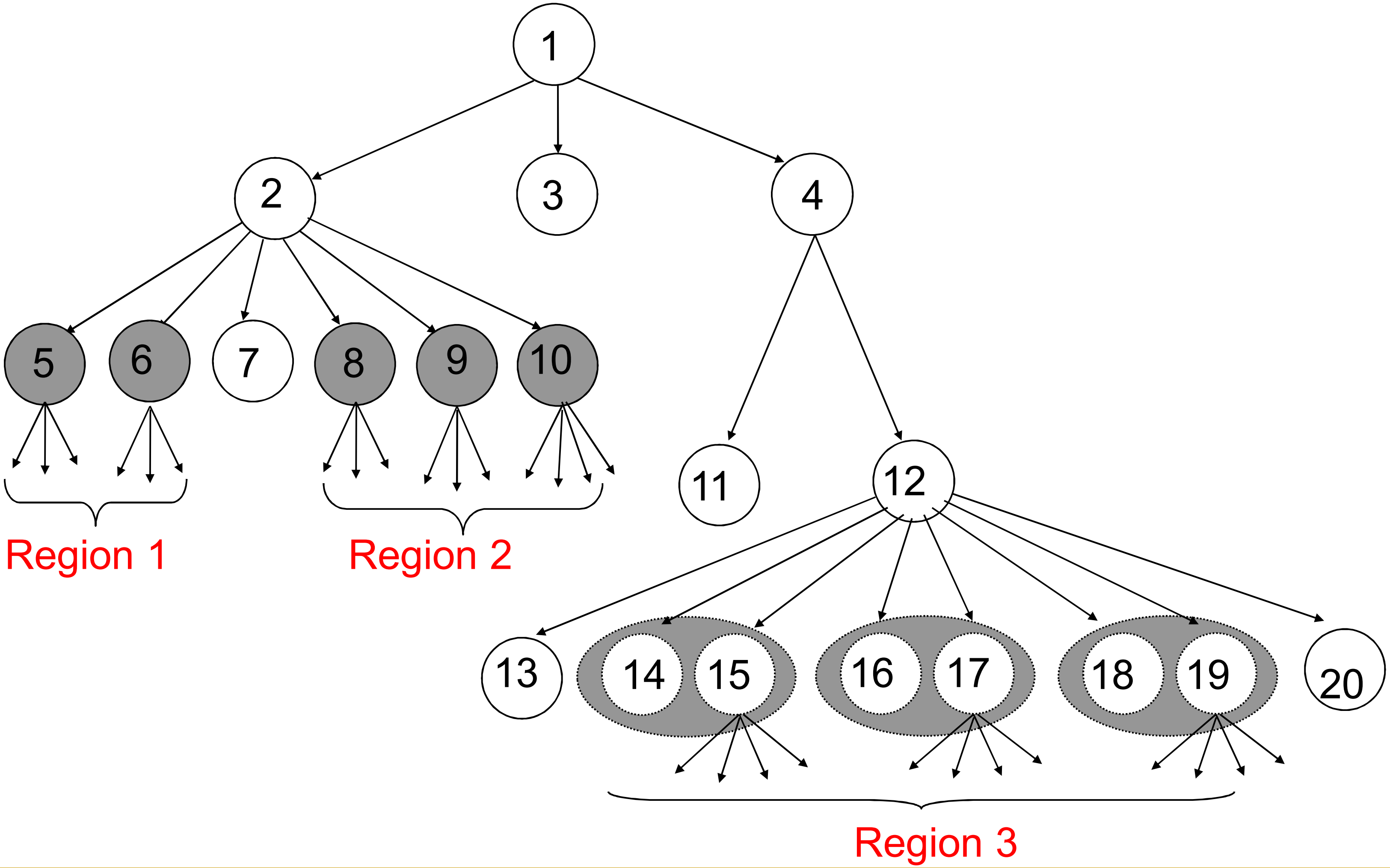
- Building the HTML Tag Tree
  - Erroneous tags, unbalanced tags, etc
- Mining Data Regions
  - Spring matching or tree matching
- Identifying Data Records

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* ≥ 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.

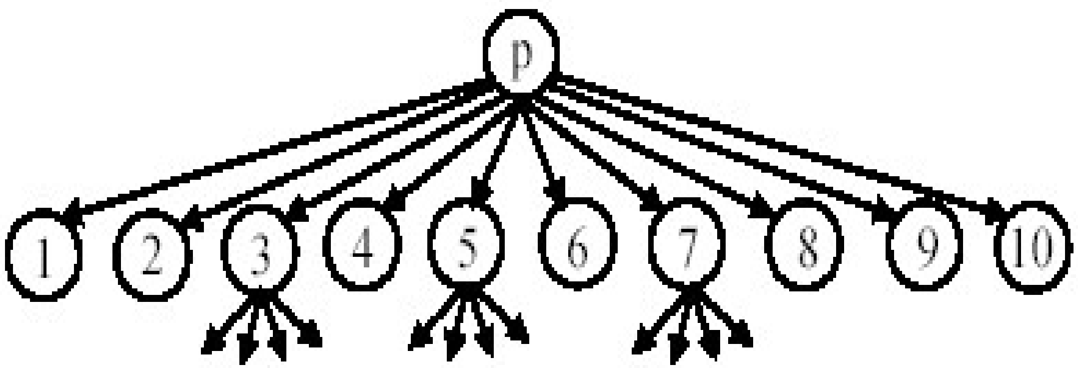


Fig. 27. Combination and comparison

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 sub-tree 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

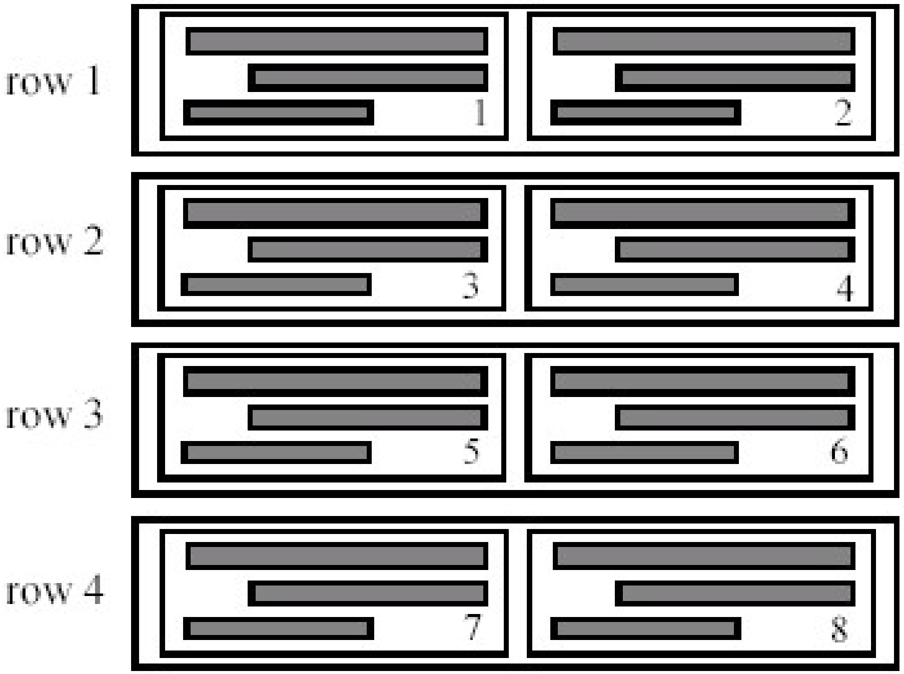
- K: Maxium number of tag nodes
- T: Similarity threshold

```
Algorithm MDR(Node, K, T)
if TreeDepth(Node) >= 3 then
    CombComp(Node.Children, K);
    DataRegions ← IdenDRs(Node, K, T);
    if (UncoveredNodes ← Node.Children − ⋃_{DR∈DataRegions} DR ) ≠ ∅ then
        for each ChildNode ∈ UncoveredNodes do
            DataRegions ← DataRegions ∪ MDR(ChildNode, K, T);
    return DataRegions
else return ∅
```

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 record.
  - 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

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)
TraverseAndMatch(*Root*, τ);
for each top level node *Node* whose children have aligned data records **do**
    PutDataInTables(*Node*);
endfor

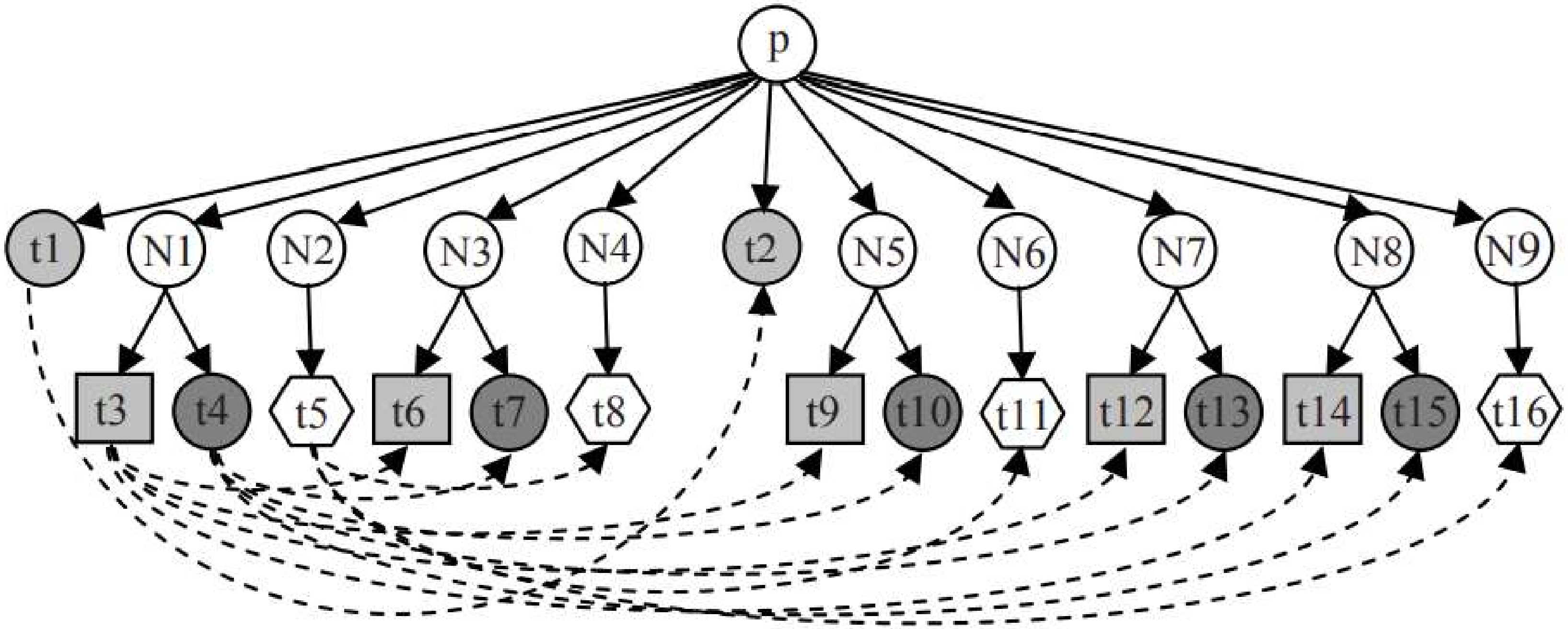
Function TraverseAndMatch (*Node*, τ)
if Depth(*Node*) ≥ 3 **then**
    for each *Child* $\in$ *Node.Children* **do**
        TraverseAndMatch(*Child*, τ); ← Recursively run the algo on children (post order)
    endfor
    Match(*Node*, τ); ← Match the root
endif

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.

```
Function Match(Node, τ)
Children ← Node.Children;
while Children ≠ ∅ do
    ChildFirst ← select and remove the first child from Children;
    for each ChildR in Children do
        if TreeMatch(ChildFirst, ChildR) > τ then
            AlignAndLink();
            Children ← Children – {ChildR}
    endfor
    if some alignments (or links) have been made with ChildFirst then
        GenNodePattern(ChildFirst)
endwhile
If consecutive child nodes in Children are aligned then
    GenRecordPattern(Node)
```

An example


p
t1
N1
N2
N3
N4
t2
N5
N6
N7
N8
N9
t3
t4
t5
t6
t7
t8
t9
t10
t11
t12
t13
t14
t15
t16

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.

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.
  - 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)

The RoadRunner System

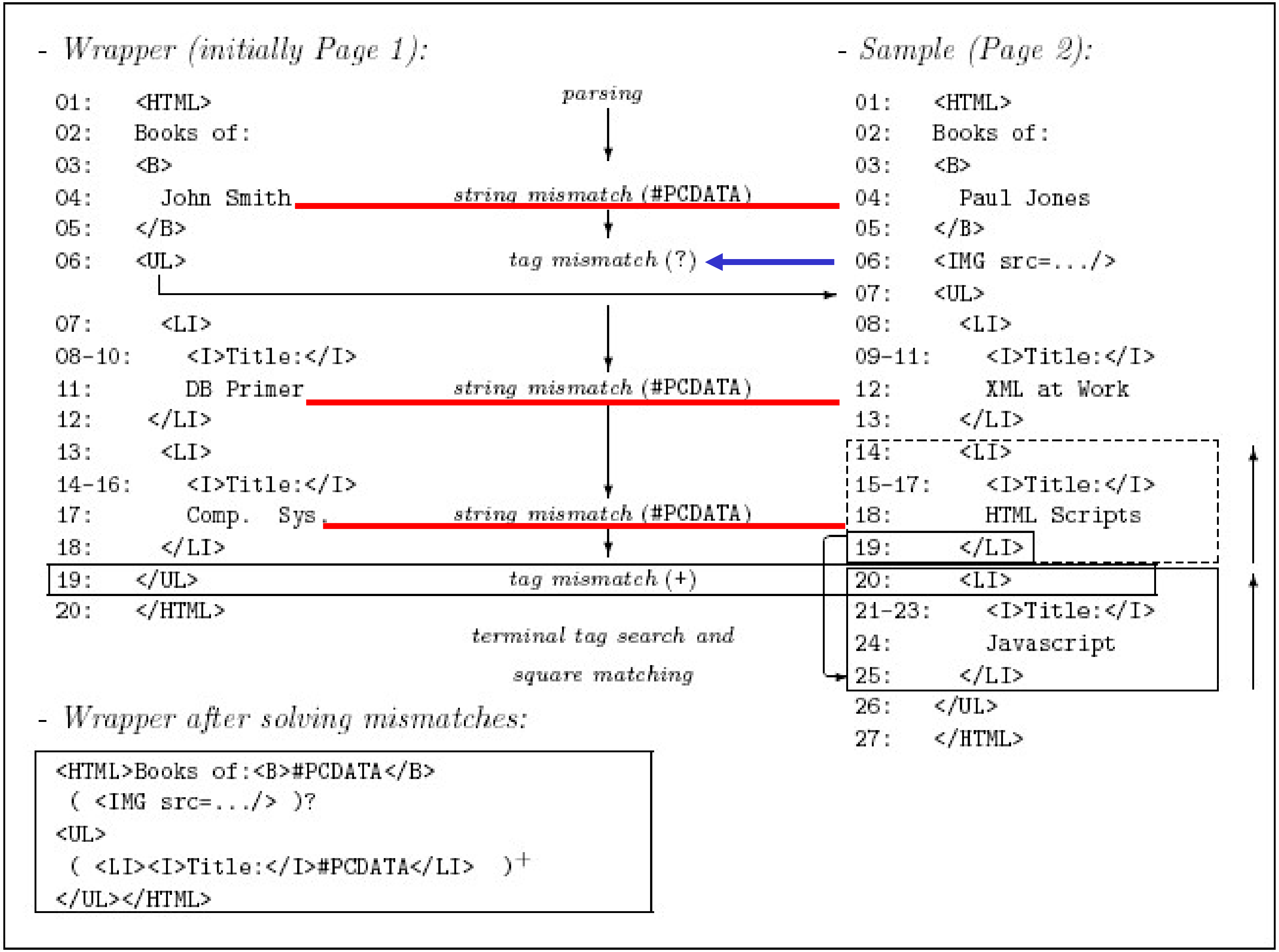
- Given a set of positive examples (multiple sample pages). Each contains one or more data records.
- From these pages, generate a wrapper as a union-free regular expression (i.e., no disjunction).
- Support nested data records.

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>
parsing
string mismatch (#PCDATA)
tag mismatch (?)
string mismatch (#PCDATA)
string mismatch (#PCDATA)
tag mismatch (+)
terminal tag search and
square matching
- 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>
- 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).

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**.