Sequence-based Web Page Template Detection

by

Wei Huang

A Thesis Presented in Partial Fulfillment
of the Requirements for the Degree
Master of Science

Approved April 2011 by the
Graduate Supervisory Committee:

Kasim Candan, Chair
Hari Sundaram
Hasan Davulcu

ARIZONA STATE UNIVERSITY

May 2011

ABSTRACT

Templates are wildly used in Web sites development. Finding the template for a given set of Web pages could be very important and useful for many applications like Web page classification and monitoring content and structure changes of Web pages. In this thesis, two novel sequence-based Web page template detection algorithms are presented. Different from tree mapping algorithms which are based on tree edit distance, sequence-based template detection algorithms operate on the Prüfer/Consolidated Prüfer sequences of trees. Since there are one-to-one correspondences between Prüfer/Consolidated Prüfer sequences and trees, sequence-based template detection algorithms identify the template by finding a common subsequence between to Prüfer/Consolidated Prüfer sequences. This subsequence should be a sequential representation of a common subtree of input trees. Experiments on real-world web pages showed that our approaches detect templates effectively and efficiently.

DEDICATION

For my parents, who offered me unconditional love and support throughout the course of this thesis.

ACKNOWLEDGEMENTS

Dr.Kasim Selçuk Candan has been the ideal thesis supervisor. His sage advice, insightful criticisms, and patient encouragement aided the writing of this thesis in innumerable ways. I also would like to thank Dr.Hari Sundaram and Dr.Hasan Davulcu for their helpful suggestions on this thesis.

Finally, I am indebted to my fellow labmates in EmitLab: Xinxin Wang, Renwei Ye, Songling Liu, Parth Nagarkar, Mithila Nagendra, Shruti Gaur and Mijung Kim for providing a stimulating and fun environment in which to learn and grow.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

Chapter 1

INTRODUCTION

Templates are widely used in web sites development. In a study, Gibson et al.[1] have found that templates represent over 40% of data available on the Web. Using web page templates bring convenience for web site designers and developers. However, the presence of templates can negatively impact information retrieval and data mining tasks. The navigation links, side bars and advertisements generated by the templates are barely related to the "main contents". Experimental evaluations have shown that the classification and clustering results can be substantially improved if templates on these pages are treated adequately [2][3][4].

Most of existing template detection algorithms are based on two basic ideas. The first idea is to divide the web page into blocks and then separate the template-generated blocks and the content blocks by some score measurement. The other idea is to find the common structure among the Document Object Model(DOM) trees of a set of web pages. In our research, we focus on detecting the template from a set of DOM trees. To identify the common subtree of DOM trees, two recent approaches based on the top-down mapping and the bottom-up mapping are proposed by Vieira et al.[2][5]. The top-down mapping algorithm is quadratic but gives optimal results. The bottom-up method is a linear time algorithm which is the theoretically fastest approach, but it may give a partial template and it may lead to false alarms. Because of limitations of the top-down and bottom-up template detection algorithms, we propose sequence-based template detection algorithms which are faster than the top-down approach and gives better results than the bottom-up method.

Motivation Application: Web page content changes regularly[6], and study showed that the content people revisit is particularly likely to change[7]. When a user visits a previously visited page, it would be helpful that changes of that page can be highlighted. However, current Web browsers do not support a historical view of Web content. Given two versions of a web page, if we can find parts repeated between these two versions, we

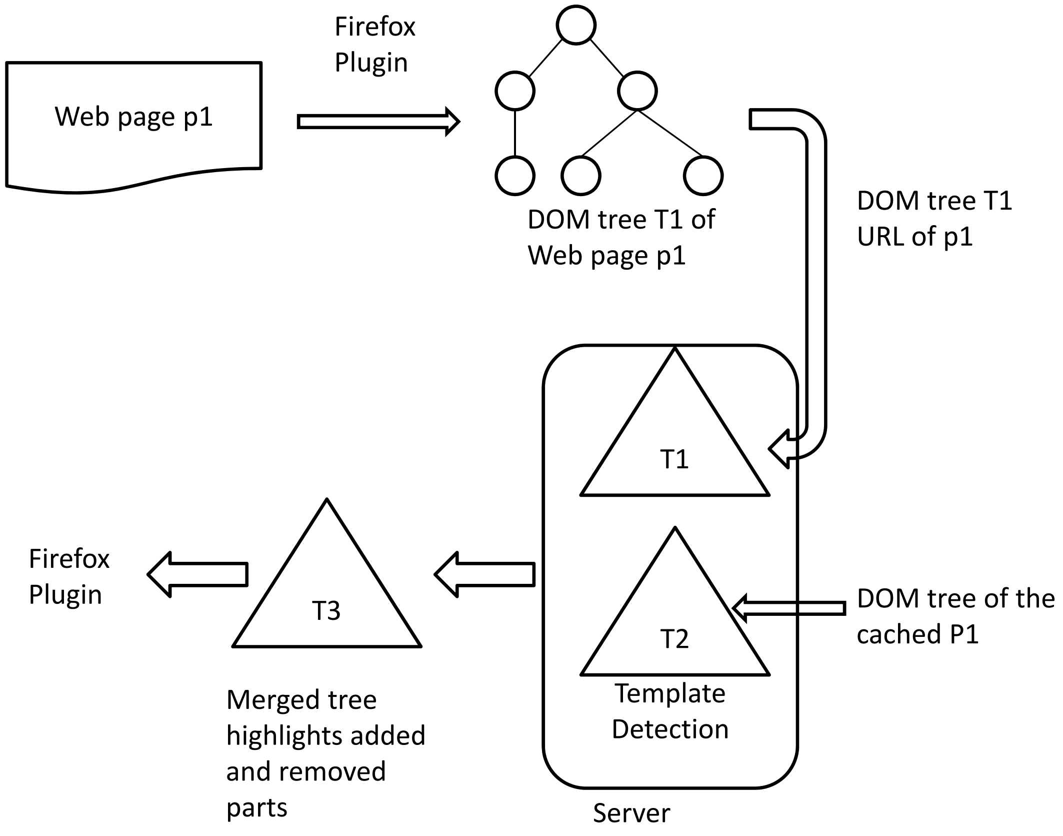
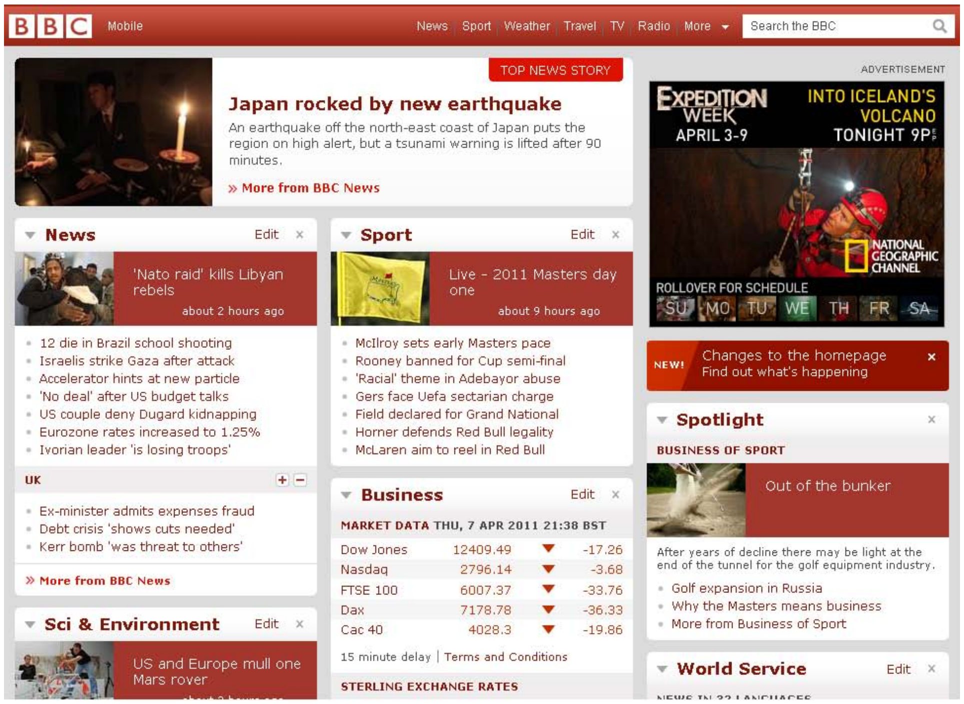

Web page p1
Firefox Plugin
DOM tree T1 of Web page p1
DOM tree T1
URL of p1
T1
T2
Template Detection
Server
DOM tree of the cached P1
T3
Merged tree highlights added and removed parts
Firefox Plugin

Figure 1.1: Illustration of the FireDiff

can easily tell what is added in the new version and what is removed from the old version. Based on the template detection algorithm, a Firefox Plugin, **FireDiff**, has been designed specifically to support awareness of how a revisited page has changed. Figure 1.1 illustrates how the system works and Figure1.2 shows a screen shot of the system. Gray shaded areas show parts that appeared in the old version but removed from current version. Yellow shaded areas indicate newly added information. Since there were not many changes between Figure 1.2a and Figure 1.2b, highlighted blocks in Figure 1.3 could help users quickly locating updated information.

Main contributions: In this thesis, Prüfer/Consolidated Prüfer sequences are applied to the problem of template detection. Before constructing Prüfer/Consolidated Prüfer sequences, each node of trees is assigned a new label based on its top-down path. Then this new label is used in constructing Prüfer/Consolidated Prüfer sequences. This


BBC Mobile | News | Sport | Weather | Travel | TV | Radio | More | Search the BBC

TOP NEWS STORY

Japan rocked by new earthquake

An earthquake off the north-east coast of Japan puts the region on high alert, but a tsunami warning is lifted after 90 minutes.

» More from BBC News

News Edit ×

'Nato raid' kills Libyan rebels
about 2 hours ago

- 12 die in Brazil school shooting
- Israelis strike Gaza after attack
- Accelerator hints at new particle
- 'No deal' after US budget talks
- US couple deny Dugard kidnapping
- Eurozone rates increased to 1.25%
- Ivorian leader 'is losing troops'

UK

- Ex-minister admits expenses fraud
- Debt crisis 'shows cuts needed'
- Kerr bomb 'was threat to others'

» More from BBC News

Sci & Environment Edit ×

US and Europe mull one Mars rover

Sport Edit ×

Live - 2011 Masters day one
about 9 hours ago

- McIlroy sets early Masters pace
- Rooney banned for Cup semi-final
- 'Racial' theme in Adebayor abuse
- Gers face Uefa sectarian charge
- Field declared for Grand National
- Horner defends Red Bull legality
- McLaren aim to reel in Red Bull

Business Edit ×

MARKET DATA THU, 7 APR 2011 21:38 BST

Dow Jones	12409.49	▼	-17.26
Nasdaq	2796.14	▼	-3.68
FTSE 100	6007.37	▼	-33.76
Dax	7178.78	▼	-36.33
Cac 40	4028.3	▼	-19.86

15 minute delay | Terms and Conditions

STERLING EXCHANGE RATES

ADVERTISEMENT

EXPEDITION WEEK APRIL 3-9 INTO ICELAND'S VOLCANO TONIGHT 9P NATIONAL GEOGRAPHIC CHANNEL ROLLOVER FOR SCHEDULE SU MO TU WE TH FR SA

NEW! Changes to the homepage ×
Find out what's happening

Spotlight ×

BUSINESS OF SPORT

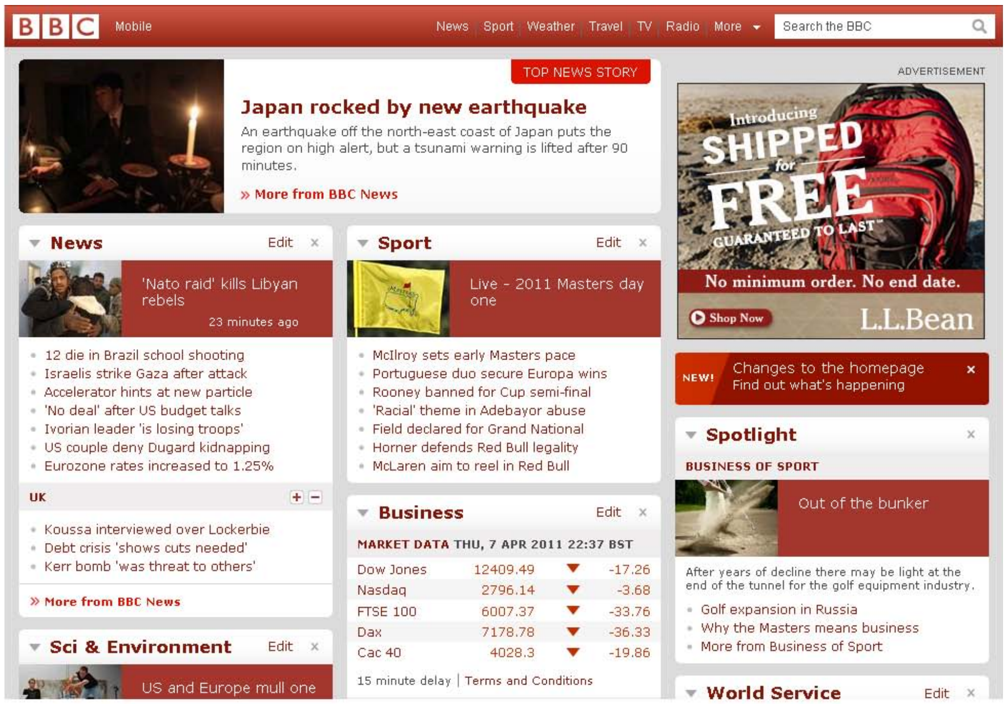
Out of the bunker

After years of decline there may be light at the end of the tunnel for the golf equipment industry.

- Golf expansion in Russia
- Why the Masters means business
- More from Business of Sport

World Service Edit ×

(a) BBC Home page at 1:38PM,4.7,2011


BBC Mobile | News | Sport | Weather | Travel | TV | Radio | More | Search the BBC

TOP NEWS STORY

Japan rocked by new earthquake

An earthquake off the north-east coast of Japan puts the region on high alert, but a tsunami warning is lifted after 90 minutes.

» More from BBC News

News Edit ×

'Nato raid' kills Libyan rebels
23 minutes ago

- 12 die in Brazil school shooting
- Israelis strike Gaza after attack
- Accelerator hints at new particle
- 'No deal' after US budget talks
- Ivorian leader 'is losing troops'
- US couple deny Dugard kidnapping
- Eurozone rates increased to 1.25%

UK

- Koussa interviewed over Lockerbie
- Debt crisis 'shows cuts needed'
- Kerr bomb 'was threat to others'

» More from BBC News

Sci & Environment Edit ×

US and Europe mull one

Sport Edit ×

Live - 2011 Masters day one

- McIlroy sets early Masters pace
- Portuguese duo secure Europa wins
- Rooney banned for Cup semi-final
- 'Racial' theme in Adebayor abuse
- Field declared for Grand National
- Horner defends Red Bull legality
- McLaren aim to reel in Red Bull

Business Edit ×

MARKET DATA THU, 7 APR 2011 22:37 BST

Dow Jones	12409.49	▼	-17.26
Nasdaq	2796.14	▼	-3.68
FTSE 100	6007.37	▼	-33.76
Dax	7178.78	▼	-36.33
Cac 40	4028.3	▼	-19.86

15 minute delay | Terms and Conditions

ADVERTISEMENT

Introducing SHIPPED for FREE GUARANTEED TO LAST™ No minimum order. No end date. Shop Now L.L.Bean

NEW! Changes to the homepage ×
Find out what's happening

Spotlight ×

BUSINESS OF SPORT

Out of the bunker

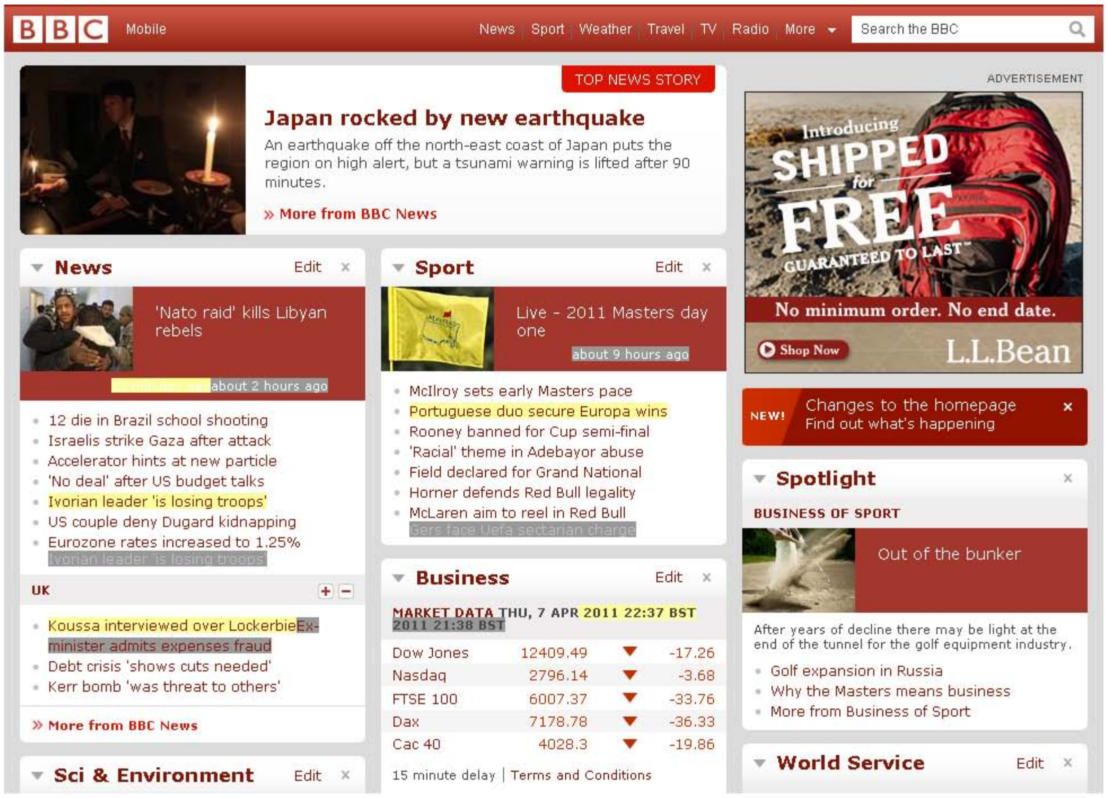
After years of decline there may be light at the end of the tunnel for the golf equipment industry.

- Golf expansion in Russia
- Why the Masters means business
- More from Business of Sport

World Service Edit ×

(b) BBC Home page at 2:37PM,4.7,2011

Figure 1.2: Screen shots of BBC Home page at different time



Japan rocked by new earthquake

An earthquake off the north-east coast of Japan puts the region on high alert, but a tsunami warning is lifted after 90 minutes.

» More from BBC News

News

'Nato raid' kills Libyan rebels

[illegible]about 2 hours ago

- 12 die in Brazil school shooting
- Israelis strike Gaza after attack
- Accelerator hints at new particle
- 'No deal' after US budget talks
- Ivorian leader 'is losing troops'
- US couple deny Dugard kidnapping
- Eurozone rates increased to 1.25%

Ivorian leader 'is losing troops'

UK

- Koussa interviewed over LockerbieEx-minister admits expenses fraud
- Debt crisis 'shows cuts needed'
- Kerr bomb 'was threat to others'

» More from BBC News

Sci & Environment

Sport

Live - 2011 Masters day one

about 9 hours ago

- McIlroy sets early Masters pace
- Portuguese duo secure Europa wins
- Rooney banned for Cup semi-final
- 'Racial' theme in Adebayor abuse
- Field declared for Grand National
- Horner defends Red Bull legality
- McLaren aim to reel in Red Bull

Gers face Uefa sectarian charge

Business

MARKET DATA THU, 7 APR 2011 22:37 BST
2011 21:38 BST

Dow Jones	12409.49	▼	-17.26
Nasdaq	2796.14	▼	-3.68
FTSE 100	6007.37	▼	-33.76
Dax	7178.78	▼	-36.33
Cac 40	4028.3	▼	-19.86

15 minute delay | Terms and Conditions

Changes to the homepage
Find out what's happening

Spotlight

BUSINESS OF SPORT

Out of the bunker

After years of decline there may be light at the end of the tunnel for the golf equipment industry.

- Golf expansion in Russia
- Why the Masters means business
- More from Business of Sport

World Service

Figure 1.3: Screen shot of FireDiff

strategy guarantees that nodes with different top-down paths can not form a mapping pair. Because of this labeling strategy, Prüfer/Consolidated Prüfer sequences of trees have some nice properties that are very useful for template detection. Details are discussed in Chapter 4.

Another major contribution is that we present two sequence-based template detection algorithms in this thesis. Since sequence comparison could be faster than tree comparison, we convert the problem of finding a common subtree of two input trees to the problem of looking for a common subsequence of their sequential representations. This common subsequence is a sequential representation of a common subtree. Different from the existing bottom-up tree mapping algorithm, our sequence-based algorithm guarantee to find a common subtree which satisfies the definition of template. Experiment on real-world web pages data shows that our approaches detect templates effectively and efficiently.

Outline of the thesis: We present the related work in Chapter 2. In Chapter 3, we briefly review the top-down and bottom-up template detection algorithms and their limitations. We present our sequence-based solutions in Chapter 4. In Chapter 5, we show the experimental results in terms of efficiency and effectiveness.

Chapter 2

RELATED WORK

The problem of template detection and web page cleaning has obtained considerable attention in research. In general, template detection algorithms are studied on two levels: the *site level* and the *page level*.

Site-level template detection The intuition of the site-level template detection is that the structures or contents repeated across many web pages are regarded as parts of the template.Bar-Yossef and Rajagopalan first introduced the problem of template detection and removal in [4]. They proposed a technique based on segmentation of the DOM tree, followed by the selection of certain segments as candidate templates depending on their content.

Lin and Ho[8] introduced the concept of *block entropy* to separate the content blocks from the redundant blocks generated by the templates. It was assumed that template generated blocks should appear more frequently than the content blocks. Thus they have different entropy values. Yi et al. [9] proposed an algorithm based on the similar intuition. The Site Style Tree (SST) approach provided another way to identify the contents generated by templates. The SST is concentrating more on the visual impression single DOM tree elements are supposed to achieve. They look for identically formatted DOM sub-trees which frequently occur in the documents and therefore are declared to be produced by templates. In this approach, a SST is built to represent a summary of all the presentation styles and all the contents found in the pages of a web site. The likelihood of its nodes representing noisy nodes on the pages is evaluated based on the diversity of presentation styles and contents associated to it on the SST tree.

The use of tree-mappings for detecting templates in a given set of web pages was introduced in [2]. Reis et al. proposed the restricted top-down tree mapping algorithm RTDM to calculate a tree edit distance between two DOM trees. The tree edit distance is used as well to perform a cluster analysis in order to find clusters of different templates within the training set. Vieira et al. proposed a bottom-up mapping strategy RBM-TD[5] which identifies the template in linear time. RBM-TD is also applied to detect multiple templates for a given web instead of obtaining only one like previous template detection did.

Page-level template detection Instead of using multiple pages to detect the template, some page-level algorithms given a single page has been proposed. Most of the page-level algorithms focused on finding the informative blocks of web pages. Song et al. [10] proposed a learning method for assigning importance weights to hierarchically arranged segments in web pages. They used a vision-based page segmentation algorithm to partition a web page into semantic blocks with a hierarchical structure and then each block was assigned an importance score based on its spatial and content features. Kao et al.[11] segment a given web page using a greedy algorithm operating on features derived from the page. Debnath et al. [12] proposed a page-level algorithm that applied a classifier to DOM nodes, but only certain nodes are chosen for classification, based on a predefined set of tags.

Chakrabarti et al. [3] develop a framework for the page-level template detection. Their method first generated a training data by applying the site-level template detection method in [1] on randomly selected sites. Then a classifier was trained from the training data set to assigns importance values to each node of a page in the test phase. The decision of which nodes in that page constitute the template is made after applying an isotonic smoothing procedure, which adjusts the importance values of the nodes.

Chapter 3

BACKGROUND

Tree Mapping and Template Detection

According Document Object Model(DOM) HTML Specification[13], each web page can be represented by a DOM tree structure. The DOM tree is labeled, ordered and rooted tree structure. Figure 3.1b shows the DOM tree of the HTML code in Figure 3.1a.

For ordered trees, a *mapping* shows the one-to-one correspondence between nodes of two trees. The following is the formal definition of *mappings*.

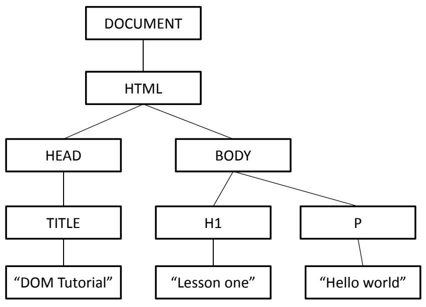
Definition 1. *Let T_x be a tree and $t_x[i]$ be the i-th node of tree T_x in a preoder walk. A mapping from a tree T_1 to a tree T_2 is a set M of ordered pairs of integers (i, j), $1 \leq i \leq n_1$, $1 \leq j \leq n_2$, satisfying the following conditions, for all (i_1, j_1),$(i_2, j_2) \in M$:*

- *$i_1 = i_2$ if, and only if,$j_1 = j_2$;*
- *$t_1[i_1]$ is to the left of $t_1[i_2]$ if, and only if, $t_2[j_1]$ is to the left of $t_2[j_2]$;*
- *$t_1[i_1]$ is an ancestor of $t_1[i_2]$ if, and only if, $t_2[j_1]$ is an ancestor of $t_2[j_2]$.*

A *mapping set* M indicates the edit operations needed to transform a tree T_1 to another tree T_2. A node $t_1[i]$ without any pair $(i, j) \in M$ associated is deleted from T_1. A

```
<html>
  <head>
    <title>DOM Tutorial</title>
  </head>
  <body>
    <h1>DOM Lesson one</h1>
    <p>Hello world!</p>
  </body>
</html>
```

(a) Example of the HTML code of a web page



(b) DOM tree of the HTML code in Figure 3.1a

Figure 3.1: DOM tree of a sample Web page

pair $(i, j) \in M$ indicates the replacement operation to substitute node $t_1[i]$ by $t_2[j]$. A node $t_2[j]$ with no pair $(i, j) \in M$ shows that it is inserted into T_2.

For each operation, a *cost* can be associated to it so the total cost to transform T_1 to T_2 can be computed. The following is the definition of *mapping cost*.

Definition 2. *Let M be a mapping between tree T_1 and tree T_2; S be the set of pairs $(i, j) \in M$ with $t_1[i]$ and $t_2[j]$ with different labels; D be the set of nodes $t_1[i]$ with no $(i, j) \in M$ associated; I be the set of nodes $t_2[j]$ with no $(i, j) \in M$ associated. The cost of mapping M is given by $|S|p + |I|q + |D|r$, where p,q,r are the costs assigned to replacement, insertion and deletion operations, respectively.*

Commonly, the cost of identical substitutions is 0 and cost of other operations is 1. The mapping with minimal cost is regarded as the optimal mapping.[2]

A *template* can be viewed as a subtree which is common to the DOM-tree representations of a web page collection. So the problem of template detection can be reduced to the problem of finding the a common subtree among a given set of trees. The latter problem can be solved by applying a tree mapping[5].

Two tree mapping based template detection algorithms were reported in recent literature. The top-down tree mapping approach was applied in [2]. Vieira, etc. proposed the bottom-up template detection algorithm in [5].

Overview of the Top-Down algorithm

Definition 3. *A mapping M between a tree T_1 and a tree T_2 is said to be top-down only if for every pair $(i_1, i_2) \in M$ there is also a pair $(i_1.parent, i_2.parent) \in M$, where i_1 and i_2 are non-root nodes of T_1 and T_2 respectively.*

The top-down distance was first introduced by Selkow. In [14], an algorithm given to compute the top-down distance between two trees in $O(n_1 n_2)$ time, where n_1 and n_2 are the numbers of nodes of two trees. A *restricted top-down mapping* was introduced in

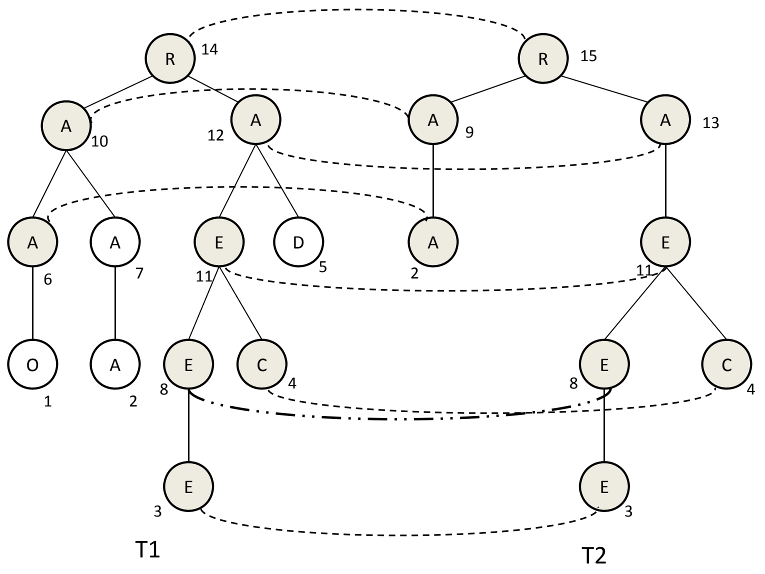


Figure 3.2: Restricted top-down mapping between T_1 and T_2

[15] to deal with problems that require the evaluation of structural similarity between web pages.

Definition 4. *A top-down mapping M between a tree T_1 and a tree T_2 is restricted if and only if for every pair $(i, j) \in M$, such that if $t_1[i] \neq t_2[j]$, there is no descendant of i or j in M.*

Based on the restricted top-down mapping, Vieira, etc proposed a template detection algorithm, RTDM-TD, in [2]. RTDM-TD first finds all identical subtrees at the same level during a post-order tree traversal. Then algorithms by Yang[16] and Reis[15] are applied to compute the restricted top-down distance between trees. The sequence of operations is obtained during the processing of calculating the restricted top-down distance. RTDM-TD keeps track of cases where no insertion, removal or update operations were applied to a given node. Similar to the top-down mapping algorithm by Chawathe[14], the RTDM-TD has a time-complexity of $O(n_1 n_2)$ in the worst case,where n_1 and n_2 are the numbers of nodes of T_1 and T_2. Figure 3.2 gives an example of restricted top-down mapping found by RTDM-TD. Gray nodes in Figure 3.2 indicate template nodes. Dash lines show the mapping pairs of the restricted top-down mapping.

Overview of the Bottom-Up algorithm

The *bottom-up tree distance* was introduced in [17] and an efficient bottom-up algorithm for finding all common rooted subtrees in a forest in linear time in the worst case was also given. The following is the formal definition of *bottom-up tree mapping*.

Definition 5. *A mapping M between a tree T_1 and tree T_2 is said to be bottom-up if for each pair $(t_1[i], t_2[j]) \in M$, then $(t_1[i_1], t_2[j_1]), \ldots, (t_1[i_k], t_2[i_k]) \in M$.*

To compute the *bottom-up tree mapping*, each node of two trees is assigned an integer class label, which is determined by class labels of its children and the label of itself. Given two nodes v and u, if they have the same class label, this indicates that two subtrees rooted at v and u are isomorphic. Then the largest common forest between two trees will be collected during a level-order tree traversal.

Vieira, etc.[5] proposed the RBM-TD algorithm algorithm based on Valiente's bottom-up tree mapping algorithm. The RBM-TD algorithm restricts the largest common forest to contain only the identical subtrees having the same top-down paths in both input trees. Thus, two nodes v of T_1 and u of T_2 will be considered as the template nodes if they have the same class label and their top-down paths are identical. Also, all nodes on the top-down paths of u and v will be considered as the template nodes when u and b are template nodes. Figure 3.3 gives an example of finding the template using RBM-TD. In Figure 3.3 numbers next nodes are class labels and gray nodes indicate template nodes. Nodes with two circles indicate that they are marked as template nodes since they are ancestors of template nodes.

Limitations of Top-down and Bottom-up Tree Mapping algorithms

One limitation of the top-down tree mapping algorithm is that it has a quadratic time complexity in the worst case. Though it is claimed that the RMTD-TD algorithm achieves better time complexity than quadratic time, experimental results in [5] shows that the running time of RMTD-TD was proportional to a quadratic function in practice.

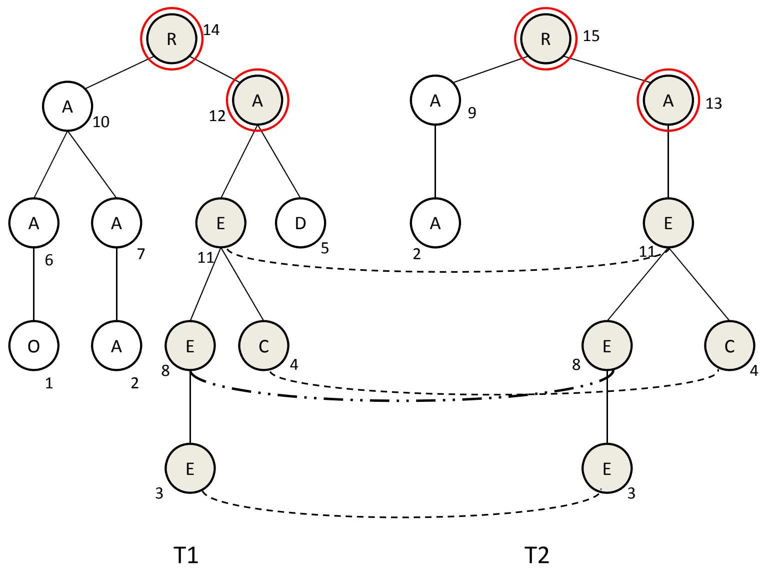
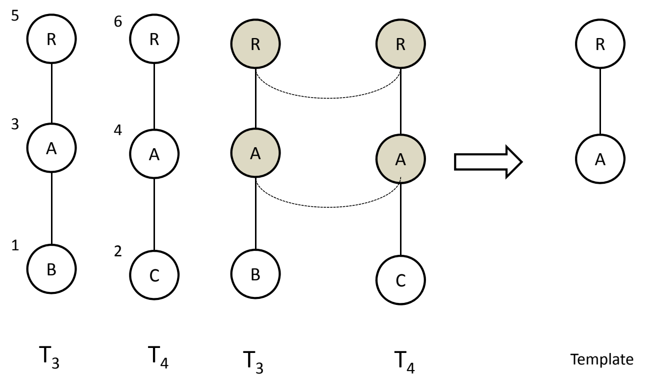


Figure 3.3: Restricted bottom-up mapping between T_1 and T_2

Although the RBM-TD is the fastest known tree mapping algorithm in literature, it has two major drawbacks. One is that some mappings are missing in the result. Figure 3.4 gives an example. Nodes of T_3 and nodes of T_4 are assigned different class labels according to the RBM-TD algorithm. Since all nodes of T_1 and T_2 are put into different equivalent classes, no mapping pairs can be found by the RBM-TD. However, there does exist one common subtree of T_3 and T_4, which is shown in Figure 3.4b.

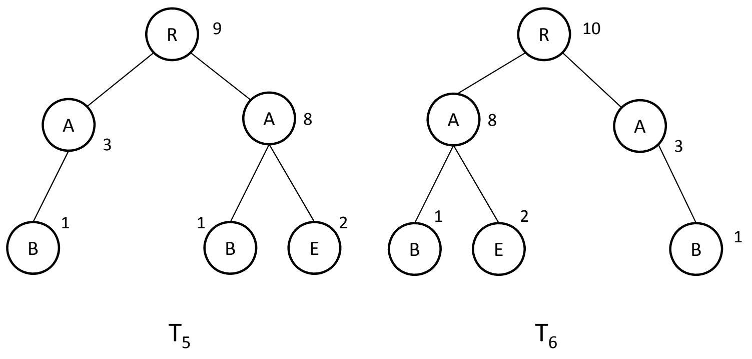
The other drawback of the RBM-TD is that it may result in false alarms. Figure 3.5 illustrates such a case. T_5 and T_6 are different since they are labeled and ordered trees. Mapping returned by the RBM-TD does not satisfy the definition of mapping for ordered trees.



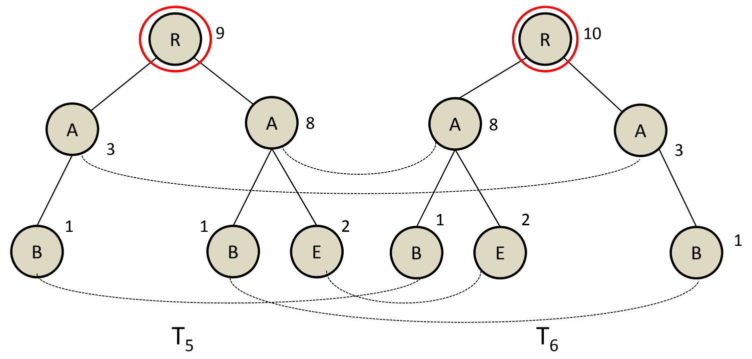
(a) Restricted bottom-up mappings between T_1 and T_2

(b) Template of T_1 and T_2

Figure 3.4: An example showing that the RBM-TD algorithm may not find all template mappings



(a) Two trees T_3 and T_4



(b) Restricted bottom-up mapping of T_3 and T_4

Figure 3.5: False mapping given by the RBM-TD

Chapter 4

SEQUENCE-BASED SOLUTION

Problem Formulation

In our work, we define the *template* of a set of web pages as following.

Definition 6. *Given a set of web pages and their DOM tree representations $T_1, T_2, \ldots T_n$, the template of $T_1, T_2, \ldots T_n$ is a common subtree T_s of $T_1, T_2, \ldots T_n$. For a tree $T \in \{T_1, T_2, \ldots T_n\}$, there should exist a restricted top-down mapping which can map all nodes of T_s to nodes of T.*

There could be multiple subtrees according to the definition of the template. The subtree with the largest size is said to be *optimal*. According to the definition of web template, template detection problem can be reduced to the problem of finding the common subtree of a set of given trees. Different from the tree mapping algorithms, we propose sequence-based template detection algorithms.

DOM to Sequence

Many approaches that transform tree structures to sequences have been reported in literature. One way to do the transformation is using a form of numbering schema that encodes each node of a tree by its positional information within the hierarchy of tree structure it belongs to. Most of the numbering schemata are based on a tree-traversal order, e.g. pre-and-post order[18] and extended pre-order[19] or textual positions of start and end tags[20][21]. In our work, two encodings are used. One is the Prüfer sequence and the other is the Consolidated Prüfer sequence. Both can construct one-to-one correspondences between a tree and the sequence.

Prüfer (1918) proposed a method that constructed a one-to-one correspondence between a labeled tree and a sequence by removing nodes from the tree one at a time. The algorithm to construct a sequence from tree T_n with n nodes labeled from 1 to n works as follows. From T_n, delete the leaf with the smallest label to form a smaller tree T_{n-1}. Let a_1 denote the label of the node that was the parent of the deleted node. Repeat this process on T_{n-1} to determine a_2 (the parent of the next node to be deleted), and continue until only two nodes joined by an edge are left. The sequence $(a_1, a_2, a_3, ..., a_{n-2})$ is called the Prüfer sequence of tree T_n. From the sequence $(a_1, a_2, a_3, ..., a_{n-2})$, the original tree T_n can be reconstructed. To construct the Prüfer sequence from tree T_n, any numbering scheme can be used to label the tree as long as it associates each node in the tree with a unique number between 1 and the total number of nodes. This guarantees a one-to-one mapping between the tree and the sequence. In our work, we adopt the same *Extended-Prüfer sequence* construction algorithm used in [22], which is slightly different to the way described above. The Prüfer sequence described above is called *Regular-Prüfer sequence* in [22]. It only contains the labels of non-leaf nodes. The *Extended-Prüfer sequence* is obtained by adding a virtual child node to each of its leaf nodes before transforming them into sequences. In this way, the *Extended-Prüfer sequence* contains all nodes of the tree. The Prüfer sequence representation defined in [22] consists of two sequences: the *Numbered Prüfer sequence*(NPS) and *Labeled Prüfer sequence*(LPS). NPS consists entirely of postorder numbers and LPS is constructed by replacing the postorder numbers in NPS with corresponding node labels. NPS and LPS convey different but complementary information. NPS gives the tree structure and LPS gives the labels for each tree node.

In our work, each node of a given tree T is assigned a new label based on its top-down path and a post-order number. Then the Prüfer sequence A of T, is built by appending nodes to A during the post-order traversal. Each element of sequence A is a node of the tree T. Figure 4.1 gives an example of Prüfer sequence representation of a labeled tree. The dashed circles in Figure 4.1 are virtually added nodes in order to get the *Extended-Prüfer sequence*. The numbers beside the nodes are the postorder numbers. The $(label,$ post-order number$)$ pair in Figure 4.1 denotes the corresponding node in T_7.

In [23],Shirish etc., proposed a *Consolidated Prüfer Sequence*, which is more concise and space-efficient compared with the classical Prüfer sequence. Similar to the classical Prüfer sequence, the Consolidated Prüfer sequence consists of two sequences: Numbered Prüfer Sequence (NPS) and Label Sequence (LS). Like the way the classical

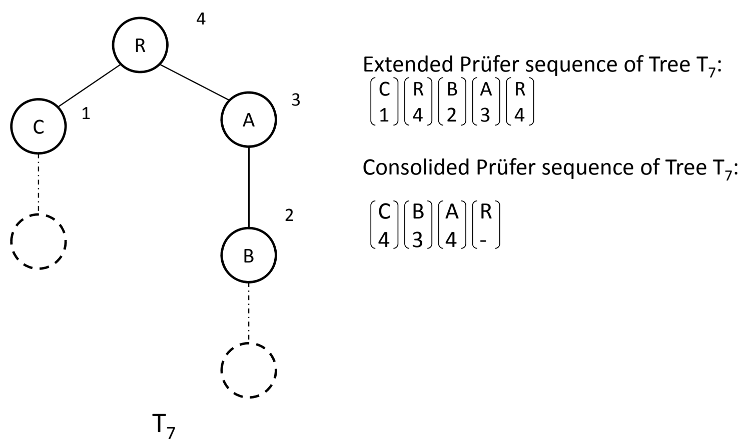

4
R
C
1
A
3
2
B
T7
Extended Prüfer sequence of Tree T7:
C 1, R 4, B 2, A 3, R 4
Consolided Prüfer sequence of Tree T7:
C 4, B 3, A 4, R -

Figure 4.1: Prüfer sequence and Consolidated Prüfer sequence of a labeled tree

Prüfer sequence being constructed, the Consolidated Prüfer sequence is constructed by doing a post-order tree traversal. The NPS of a Consolidated Prüfer sequence is constructed by removing the node with the smallest post-order traversal number and by appending its parent node number to the already constructed partial sequence. The LS is built in the similar but by appending the label of the removed nodes instead of their parent node numbers to the sequence. The Consolidated Prüfer Sequence provides a bijection between rooted, ordered and labeled trees and sequences. Since the *Extended-Prüfer sequence* is constructed by adding a virtual child node to each of its leaf nodes before transforming them into sequences, it is longer than the Consolidated Prüfer sequence.

Similar to the process of constructing the Prüfer sequence, in our work, each node of tree T is assigned a new label based on the top-down path before generating the Consolidated Prüfer sequence. Figure 4.1 shows an example of Consolidated Prüfer sequence representation of a rooted, labeled and ordered tree. Different from the Extended Prüfer sequence, a $(\text{label}, \text{post-order number})$ pair in the Consolidated Prüfer sequence corresponds to an edge in the tree T_7.

Subsequence Matching

Before we discuss how to find the template, we first introduce some basic concepts of the Longest Common Subsequence(LCS) problem. The LCS problem is well studied in literature. The dynamic programming algorithm[24] is classical to solve the LCS problem. Studies show that the general LCS problem has a lower bound theoretical time complexity of $\Omega(n^2)$. However, there are algorithms which can achieve better performance under certain situations. The theoretically fastest known algorithm is proposed by Masek and Paterson[25], which takes time $O(n^2/logn)$. For situations where the length p of a LCS is expected to be long, Myers[26] and Ukkonnen[27] developed an algorithm which takes time $O(n(n-p))$. Wu et al.[28] later improved that algorithm to $O(n(m-p))$. If the alphabet size is fixed, Hirschberg[29] and Hunt/Szymanski[30] proposed algorithms with processing time $O(pn)$ and $O(m + rlogp)$ respectively and both methods need an additional preprocessing time $O(nlogs)$.

The concept of subsequence is stated as follows.

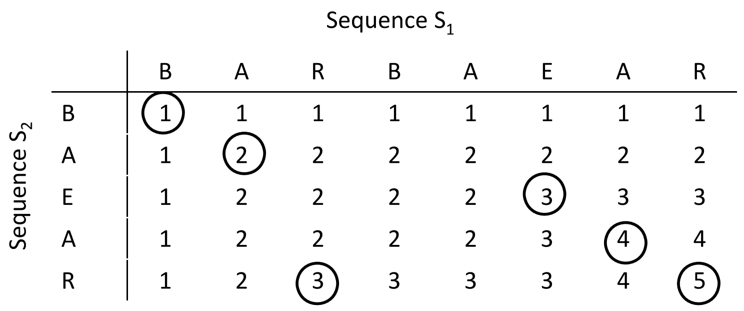
Definition 7. *A subsequence is any string that can be obtained by deleting zero or more symbols from a given string.*

The longest common subsequence is defined as follows.

Definition 8. *Let $A = a_1a_2 \ldots a_m$ and $B = b_1b_2 \ldots b_n$, $m \leq n$,be two sequences over an alphabet $\Sigma = \{\sigma_1\sigma_2 \ldots \sigma_s\}$ of size s. A sequence C is said to be the common subsequence of A and B if and only if C is a subsequence of both A and B. A common subsequence with the maximal length is defined as the longest common subsequence(LCS).*

The classical dynamic programming technique for solving LCS problem is to determine the longest common subsequence for all possible prefix combinations of input strings A and B by filling an $m \times n$ table L. The L table is filled by the following recursion.

Let $|A|$ denote the length of a string A and let A_i,$0 \leq i \leq |A|$, denote the length i prefix of A. Define
$L_{i,j} = max\{|C|$: C is a common subsequence of A_i and $B_j\}$,$0 \leq i \leq m$,$0 \leq j \leq n$, to be the length of a LCS between A_i and B_j.



Sequence S_1

Sequence S_2

	B	A	R	B	A	E	A	R
B	$\textcircled{1}$	1	1	1	1	1	1	1
A	1	$\textcircled{2}$	2	2	2	2	2	2
E	1	2	2	2	2	$\textcircled{3}$	3	3
A	1	2	2	2	2	3	$\textcircled{4}$	4
R	1	2	$\textcircled{3}$	3	3	3	4	$\textcircled{5}$

Figure 4.2: Table L of sequences S_1 and S_2

$$L_{i,j} = \begin{cases} 0 & \text{if } x = 0 \text{ or } j = 0 \\ L_{i-1,j-1} + 1 & \text{if } a_i = b_j \\ max\{L_{i-1,j}, L_{i,j-1}\} & \text{if } a_i \neq b_j \end{cases}$$

After table L is constructed, the value of $L_{m,n}$ indicates the length of an LCS of A and B. Computing the table takes time in $O(mn)$ and reconstruct an LCS takes linear time. Figure 4.2 gives an example of the table L of sequences S_1 and S_2.

Study showed that it is not necessary to fill every entry in the table L to get the LCS[31]. Thus, to improve the running time of the classical dynamic programming method, one approach is to reduce the number of entries to be computed in the table L.

An ordered pair (i, j) of positions in A and B is called a *match* if and only if $a_i = b_j$. The set of all matches is $M = \{(i, j)|1 \leq i \leq m, 1 \leq j \leq n$ and $a_i = b_j\}$. Each match belongs to a class $C_k = \{(i, j)|(i, j) \in M \wedge L_{i,j} = k\}$,$k \geq 1$ and $C_0 = \{(0, 0)\}$. A match $m \in M$ is called a *k-match*. It is proved that determining *dominant matches* is sufficient to solve the LCS problem[31]. Thus the process of computing the LCS can be sped up by only concentrating on the *dominant matches*. The *dominant match* is defined as follows.

Definition 9. *A match $(i, j) \in C_k$ is called a dominant k-match if*
$\forall(i\prime, j\prime) \in C_k : (i\prime > i \wedge j\prime \leq j)$ *or* $(i\prime \leq j \wedge j\prime > j)$.
$D_k = \{(i, j)|(i, j)$ *is a dominant k-match*$\}$.

Table 4.1: List of algorithms for LCS problem

Authors	Time Complexity	Reference
Wagner,Fischer	$O(mn)$	[24]
Hunt,Szymanski	$O(pn + n\log s)$	[30]
Hirschberg	$O(pn + n\log s)$	[29]
Apostolico,Guerra	$O(m\log n + d\log(2mn/d))$	[32]
Wu,Manber,Myers	$O(n(m-p))$	[28]
Rick	$O(ns + min\{pm, p(n-p)\})$	[31]

Circled entries of table L in Figure 4.2 give examples of dominant matches. Auxiliary data structures were introduced to compute dominant matches efficiently. The table *CLOSEST* was used in Apostolico/Guerra[32] algorithm to help finding dominant matches efficiently.

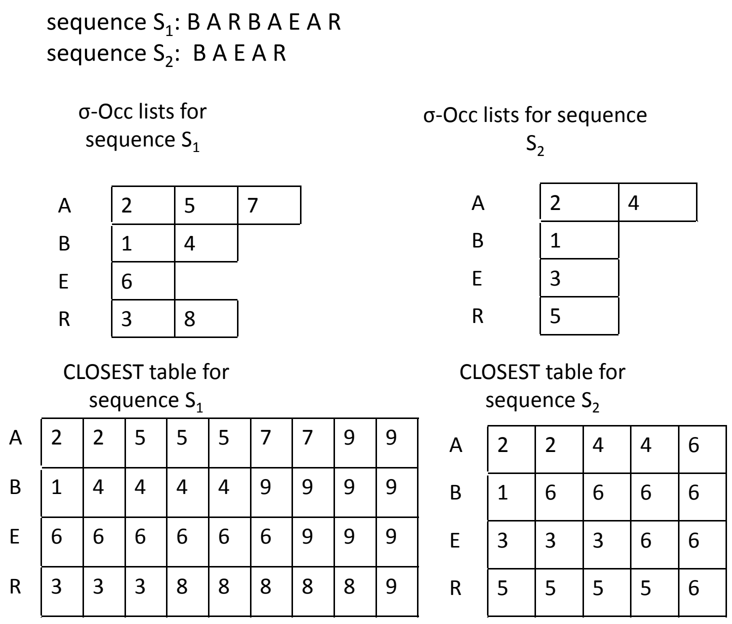
Given a sequence $B = b_1b_2\dots b_n$ over some alphabet $\Sigma = \{\sigma_1, \sigma_2, \dots \sigma_s\}$,

$$CLOSEST[\sigma_i, j] = \begin{cases} n+1 & \text{if } j = n+1 \\ min\{\{j\prime \geq j | b_{j\prime} = \sigma_i\} \cup \{n+1\}\} & \text{if } j = 1, 2, \dots, n \end{cases}$$

To compute the CLOSEST table, a list σ-OCC of all positions of B which correspond to occurrences of σ, for all $\sigma \in \Sigma$, in increasing order. Figure 4.3 gives an example of σ-OCC lists and the CLOSEST table. σ-OCC lists can be obtained by scanning the sequence once and the CLOSEST table can be computed in $O(ns)$. The time complexity of computing the CLOSEST table can be reduced to $\Theta(n)$ using a compact representation[31]. Rick[31] improved Apostolico/Guerra algorithm to $O(ns + min\{pm, p(n-p)\})$ in time complexity. If the compact representation of the CLOSEST table is used, the time complexity of Rick's approach is $O(n + min\{pm\log s, p(n-p)\log s\})$.

Since Rick's algorithm is fast when an LCS is expected to be long and it has a much smaller degeneration in intermediate situations, we adopt Rick's algorithm in our work as the subsequence matching algorithm.

Let sequences A and B denote the Prüfer sequences of trees T_1 and T_2. In the rest of this thesis, a match is a $(A[i], B[j])$ pair such that $A[i].label = B[j].label$. Also, the label of each node is newly assigned according to its top-down path from the root. The


sequence S_1: B A R B A E A R
sequence S_2: B A E A R

σ-Occ lists for sequence S_1

A	2	5	7
B	1	4	
E	6		
R	3	8	

σ-Occ lists for sequence S_2

A	2	4
B	1	
E	3	
R	5	

CLOSEST table for sequence S_1

A	2	2	5	5	5	7	7	9	9
B	1	4	4	4	4	9	9	9	9
E	6	6	6	6	6	6	9	9	9
R	3	3	3	8	8	8	8	8	9

CLOSEST table for sequence S_2

A	2	2	4	4	6
B	1	6	6	6	6
E	3	3	3	6	6
R	5	5	5	5	6

Figure 4.3: CLOSET table and σ-OCC lists for S_1 and S_2

subsequence matching phase gives all dominant matches of A and B for later template retrieval.

Template Match Identification

Let A and B be sequences derived from two trees T_1 and T_2. LCS L of A and B gives of a set of matches. Each match $(A[i], B[j])$ can be seen as a mapping pair between nodes of $A[i]$ and $B[j]$. Since the subsequence matching algorithm generates the LCS based on the label information and ignores the structural information, *structure matching* is needed to prune false positives matches.

Because the sequence A is obtained by appending nodes of a tree to it during a post-order traversal. If $A[i]$ and $A[j]$ represent the same node in tree T, we say $A[i] = A[j]$.

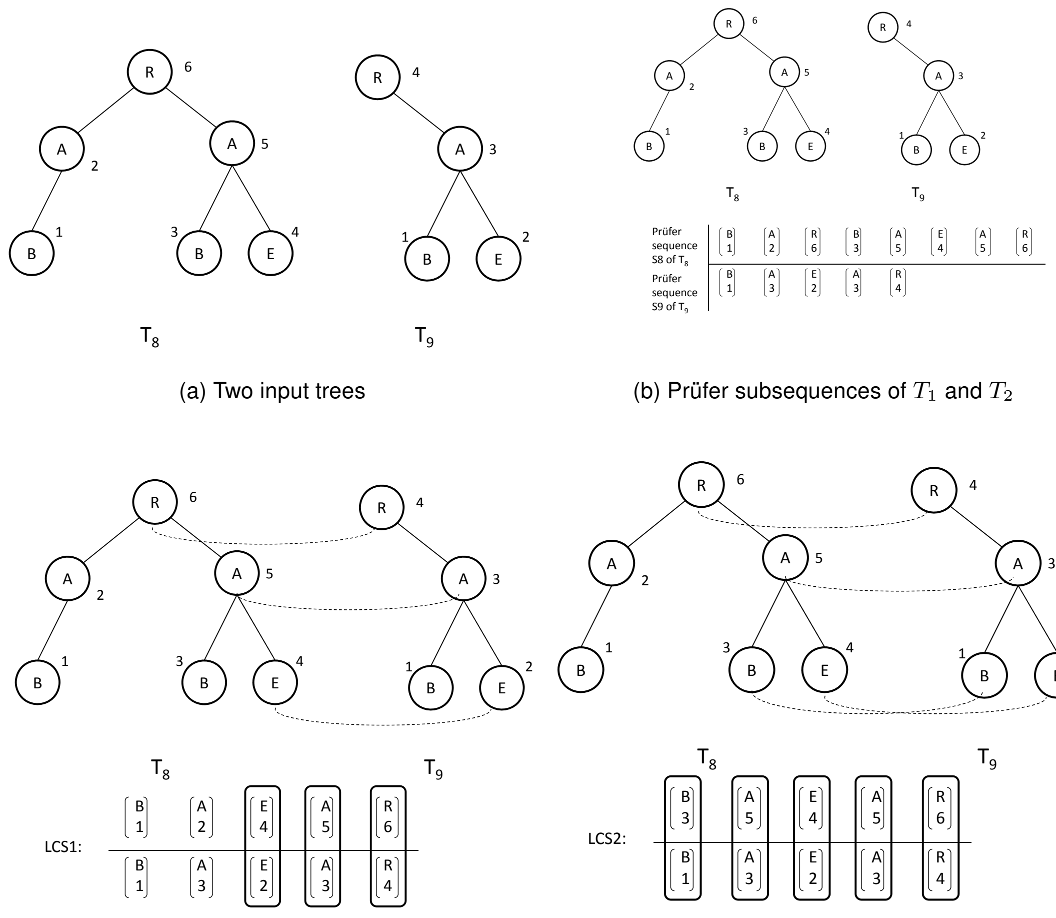
Definition 10. *Template match: Let A and B be sequences derived from two trees T_1 and T_2. Consider a LCS of A and B, $L = \{(A[i_1], B[j_1]), (A[i_2], B[j_2]), \dots, A[i_m], B[j_m]\}$. $(A[i_k], B[j_k])$ is a template match if and only if either of the following conditions hold:*

1. *Both $A[i_k]$ and $B[j_k]$ are root nodes.*
2. *$(A[i_k].parent, B[j_k].parent)$ is a template match and currently there do not exist template matches $(A[i_k], B[j_p])$ and $(A[i_q], B[j_k])$, $B[j_p] \neq B[j_k]$ and $A[i_q] \neq A[i_k]$.*

Based on the definition of template match, Algorithm 1 shows a naive way to find template matches from a given LCS. Figure 4.4 shows an example of template matches found from LCSs. Figure 4.4b shows Prüfer sequences of two input trees in Figure 4.4a. Two LCSs, $LCS1$ and $LCS2$, can be found by comparing the sequences. Template matches found from $LCS1$ and $LCS2$ and their corresponding mapping pairs are shown in Figure 4.4c and Figure 4.4d. One major drawback of this naive method is that we need to enumerate all possible LCSs to obtain the optimal mapping. This is very costly since number of all LCSs could be exponential[23].

Algorithm 1 TemplateMatchIdentification-Naive(LCS)

```
1: $LCS\_A \leftarrow$ entries of $A$ in $LCS$
2: $LCS\_B \leftarrow$ entries of $B$ in $LCS$
3: $i \leftarrow$ size of $LCS$, $j \leftarrow$ size of $LCS$
4: $i++$,$j++$
5: while $i > 0$ AND $j > 0$ do
6:     if $LCS\_A[i]$ and $LCS\_B[j]$ are root nodes then
7:         mark $LCS\_A[i]$ and $LCS\_B[j]$ as template nodes
8:         $mapping[LCS\_A[i]] \leftarrow LCS\_B[j]$, $rmapping[LCS\_B[j]] \leftarrow LCS\_A[i]$
9:         $i--$,$j--$
10:    end if
11:    if $mapping[LCS\_A[i]] = LCS\_B[j]$ AND $rmapping[LCS\_B[j] = LCS\_A[i]$ then
12:        $i--$,$j--$
13:        continue;
14:    end if
15:    if $mapping[LCS\_A[i]] \neq LCS\_B[j]$ AND $rmapping[LCS\_B[j] \neq LCS\_A[i]$ then
16:        $i--$,$j--$
17:        continue;
18:    end if
19:    if $mapping[LCS\_A[i]] = null$ AND $rmapping[LCS\_B[j] = null$ then
20:        if $(A[i].parent, B[j].parent)$ is a template match then
21:            $mapping[LCS\_A[i]] \leftarrow LCS\_B[j]$, $rmapping[LCS\_B[j]] \leftarrow LCS\_A[i]$
22:            $i--$,$j--$
23:            continue;
24:        end if
25:    end if
26: end while
```



(a) Two input trees

(b) Prüfer subsequences of T_1 and T_2

(c) Mapping found from LCS1

(d) Mapping found from LCS2

Figure 4.4: Finding template match from possible LCSs

Optimization

To avoid generating all LCSs, we present two template match identification algorithms operating on the dominant matches obtained in subsequence matching phase.

Prüfer sequence-based Template Match Identification

Theorem 1. *Let p denote the length of an LCS of sequence A and B. A and B are Prüfer sequences of trees T_1 and T_2. Then D_p ,the set of dominant p-matches has only one match $(A[i], B[j])$ and nodes $A[i]$ and $B[j]$ are roots of tree T_1 and T_2.*

Proof We prove Theorem 1 by contraction. If $|D_p| \neq 1$ and there exists another dominant match $(A[i\prime], B[j\prime]) \in D_p$, where $A[i\prime]$ and $B[j\prime]$ are not root nodes. Then there must exist two root nodes $A[i]$ and $B[j]$ where $i > i\prime$ and $j > j\prime$ because the root node is always the last element of a Prüfer sequence. According to the definition of a dominant match, $(A[i], B[j])$ should be a dominant $(p+1)$-match. Then the length of the LCS of sequence A and B is $(p+1)$. Here is the contradiction.

According to Theorem 1, our template identification algorithm starts from D_p and then backtrack to D_1. This is similar to a top-down tree mapping procedure, since Prüfer sequences are built in a post-order fashion.

Since DOM trees are labeled, rooted and ordered trees, template matches should also preserve relative orders between nodes. If a match $(A[i], B[j])$ is marked as a template match, it means that template matches between nodes to the right of $A[i]$ and $B[j]$ are already identified. To preserve relative orders between nodes, next template match $(A[i\prime], B[j\prime])$ should satisfy $i\prime < i \wedge j\prime < j$. Since A and B are obtained from post-order traversals, nodes of T_1 and T_2 are visited from top to bottom and right to left if A and B are visited from right to left. Thus each edge of a tree will be visited at most once. This guarantees that template matches identified preserve relative orders between trees.

Theorem 2. *Given nodes $A[i]$ and $A[j] (j > i)$. $A[j]$ is the first element having the same label as $A[i]$ on the right side of $A[i]$. $A[i] = A[j]$ if and only if there does not exist an $A[l] (1 < l < j)$ such that $A[l].label = A[i].parent.label$.*

Proof ONLY IF: A is constructed by appending the parent of the node currently being visited in a post-order traversal. If there exists an $A[l] (i < l < j)$ such that $A[l].label = A[i].parent.label$, then it means that node $A[i]$ pointing to in the tree structure is already visited when $A[j]$ is appended to the sequence. Since each node is visited only once during the sequence construction, $A[i] \neq A[j]$. Thus, there does not exist an $A[l] (i < l < j)$ such that $A[l].label = A[i].parent.label$.

IF: We prove the necessary condition by contradiction. If $A[i] \neq A[j]$, $A[i]$ must be a node which is left to $A[j]$ since $i < j$. Node $A[i]$ must be visited and the parent of $A[i]$ must be appended to the sequence right after $A[i]$ since sequence A is constructed by appending the parent node to the sequence in a post-order tree traversal. Thus, there exists an $A[l](1 < l < j)$ such that $A[l].label = A[i].parent.label$. Here is the contradiction.

According to Theorem 2, if $A[j]$ is the first node on the right side of $A[i]$ such that $A[i] = A[j]$, then $\{A[i+1]A[i+2] \dots A[j-1]\}$ is the sequential representation of the subtree rooted at $A[j-1]$.$A[j-1]$ is a child node of $A[j]$. In other words, $i+1$ is the *left boundary* of the sequential representation of the subtree rooted at $A[j-1]$ in A.

Now, for $A[i]$,$(1 \leq i \leq n)$, we introduce a *branch boundary* attribute denoted by $A[i].bd$, to indicate the range of one of its branch in the sequence.

$$A[i].bd = max(A[i].parent.bd, s+1)$$

$s = max(\{j \mid A[j].label = A[i].label \wedge 1 \leq j < i\})$. If there does not exist an $A[j]$ such that $A[j].label = A[i].label$ $(1 \leq j \leq i)$, then $s = 0$. If $A[i]$ is root node, $A[i].parent.bd = 1$.

For each σ in the alphabet Σ, positions it appearing in the sequence is stored in the σ-Occ list in increasing order during the subsequence matching phase. The boundary attribute of a given element $A[i]$ can be computed efficiently using binary search. The algorithm to compute the boundary of an element is presented in Algorithm 2.

Algorithm 2 computeBoundaries($(A[i], B[j])$)

```
1: $occ\_list\_A \leftarrow \sigma$-Occ list of $A[i].label$ for $A$
2: $occ\_list\_B \leftarrow \sigma$-Occ list of $B[i].label$ for $B$
3: $left \leftarrow$ the largest value which is smaller than $i$ in $occ\_list\_A$
4: $up \leftarrow$ the largest value which is smaller than $j$ in $occ\_list\_B$
5: **if** $left < B[j].parent.bd$ **then**
6:     $left \leftarrow B[j].parent.bd$
7: **end if**
8: **if** $up < A[i].parent.bd$ **then**
9:     $up \leftarrow A[i].parent.bd$
10: **end if**
11: $A[i].bd \leftarrow up$
12: $B[j].bd \leftarrow left$
```

If $A[i].parent.bd < s+1$, then $A[j] = A[i]$ according to Theorem 2. Thus, sequence $A[A[i].bd]A[A[i].bd+1]] \dots A[i-1]$ is the sequential representation of the subtree rooted at $A[i-1]$.

If $(A[i], B[j])$ is the current template match and neither of nodes is leaf node, this means that currently we are finding mapping pairs between subtrees rooted at $A[i-1]$ and $B[j-1]$. Let $up = A[i].bd$, $left = B[j].bd$. Thus, match $(A[i\prime], B[j\prime])$ is not considered as a template match if $i\prime < up$ or $j\prime < left$ since $A[i\prime]$ or $B[j\prime]$ is not a descendant of $A[i]$ or $B[j]$ respectively.

Let $(A[i], B[j])$ be the current template match. An unmarked match $(A[s], B[t])$ is a template match if it satisfy the following conditions:

1. $(A[s].parent, B[t].parent)$ is a template match
2. $A[s].parent.bd \leq s < i$ and $B[t].parent.bd \leq t < j$

Given a match $(A[i], B[j]) \in C_k$, it means that k is the upper bound of the size of the common forest of forests denoted by $A[1]A[2]\ldots A[i]$ and $B[1]B[2]\ldots B[j]$. To find template matches as many as possible while maintaining a low cost, a match $(A[i\prime], B[j\prime]) \in C_{k\prime}$ is marked as a template match if it satisfies the above conditions and $k\prime$ is the largest possible value.

If current template match is $(A[i], B[j])$ and $(A[i], B[j]) \in C_k$, $(A[s], B[t]) \in C_{k\prime}$ is picked as next template match if it satisfy the following conditions:

1. $(A[s].parent, B[t].parent)$ is a template match
2. $s < i$ and $t < j$
3. $A[s].parent.bd \leq s$ and $B[t].parent.bd \leq t$
4. $k\prime < k$ and the value of $k\prime$ is closest to k

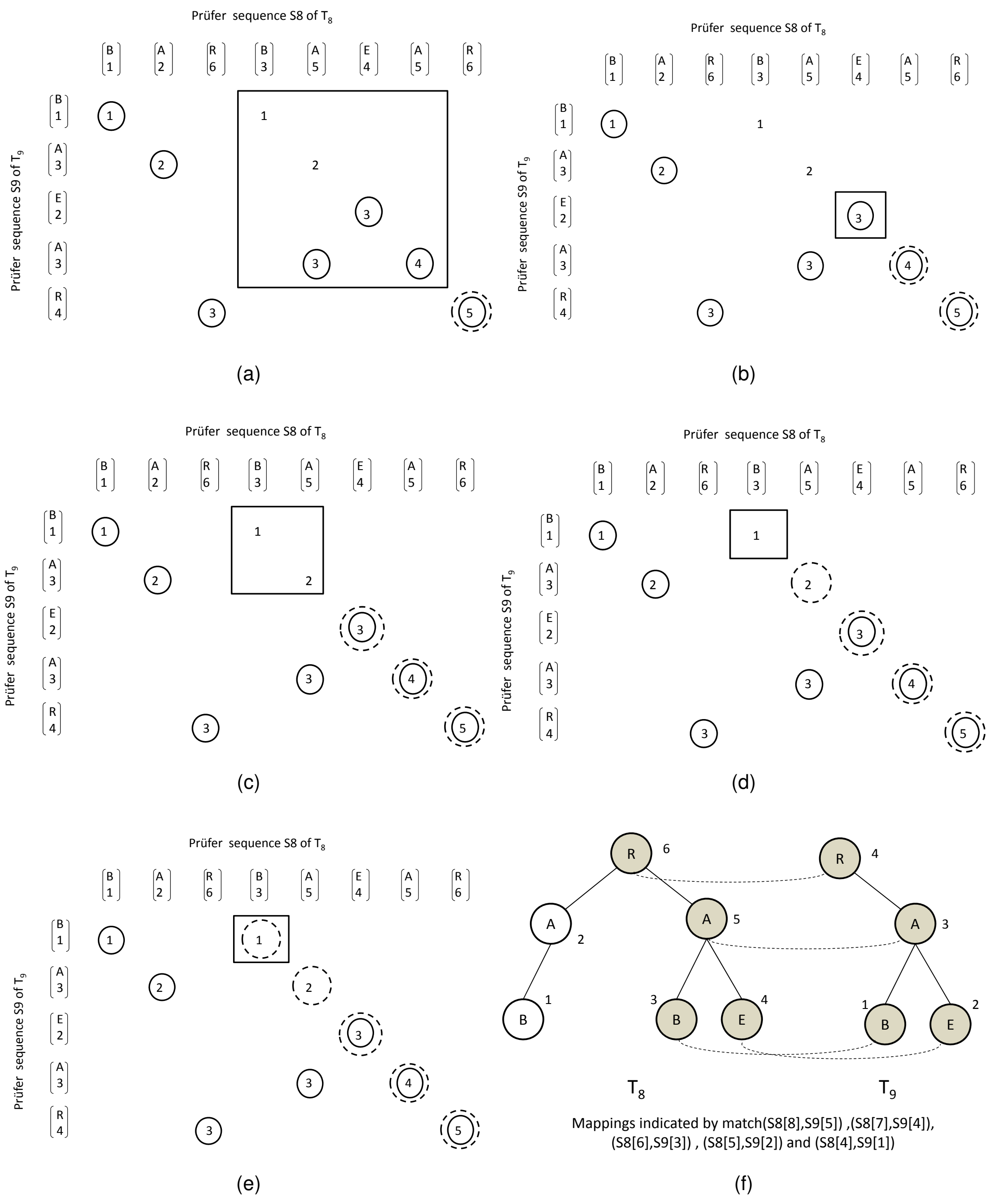
Condition 1,2, and 3 ensure that identified template matches can form a mapping which is consistent with the definition of restricted top-down tree mapping. Condition 3 and 4 is used to include as many template matches as possible. If a dominant match satisfies all other conditions except Condition 3, we can check if there is any non-dominant match which satisfies all four conditions. In this thesis, we call Condition 3 and 4 the current *boundary conditions*.

Algorithm 3 gives the procedure of finding template matches. The input array $dm[1, 2, \ldots, p]$ contains p lists of dominant matches and p is the length of the LCS. Each list $dm[i]$,$1 \leq i \leq p$, consists of all dominant matches in D_i. According to the definition of template and Theorem 1, the algorithm starts by marking the *root-root* match in D_p as the template match and then backtracks to D_1 to identify template matches. At Line 9, a match $(A[i\prime], B[j\prime])$ is removed from the head of $dm[k]$. If $(A[i\prime], B[j\prime])$ is already a template match, then we just update boundaries up,$left$,$right$ and $bottom$ (Line 11-15). If $(A[i\prime].parent, B[j\prime.parent])$ is a template match, we check if $(A[i\prime], B[j\prime])$ satisfies the current boundary conditions. If it is true, mark $(A[i\prime], B[j\prime])$ as template match and update current boundary conditions(Line 18-24). Otherwise, we find non-dominant matches which are within the boundaries (Line 26-28). If $(A[i\prime].parent, B[j\prime.parent])$ is not a template match, but at least one of $A[i\prime].parent$ and $B[j\prime.parent]$ is marked, there could be non-dominant matches qualifying for template match. Line 32 to Line 43 shows how to find an alternative non-dominant matches. Without loss of generality, let us assume $A[i\prime].parent$ is not marked and $B[j\prime.parent]$ is marked. Since $B[j\prime.parent]$ is mapped to some node $A[k]$, current boundary conditions are defined by $B[j\prime.parent]$,$A[k]$ and current template match. After we have the boundary conditions, we check to see if there is a non-dominant match $(A[l], B[j\prime])$ within the boundaries. Finding one l which satisfies the boundary conditions would be enough since according the Theorem 2, all $A[l]$ within the boundaries represent the same node in the tree. Finding l is similar to Algorithm 2. l can be obtained efficient by running binary search on the σ-OCC list of $A[i\prime].label$.

Figure 4.5 gives an example showing a step-by-step template match identification based on Prüfer sequence. The input trees are T_8 and T_9 given in Figure 4.4a. In figure 4.5, entries circled by solid lines are the dominant matches obtained in the subsequence matching phase. In Figure 4.5a, the root-root match $(S8[8], S9[5])$ is initialized as template match which is denoted by the dash circle. Since root node appears twice in $S8$, boundaries $left = 4$,$up = 1$,$right = 7$ and $bottom = 4$. The rectangle indicates the current boundary conditions. In Figure 4.5b, $(S8[7], S9[4])$ is within the boundaries and neither $S8[7]$ nor $S9[4]$ is mapped to other nodes. Thus, $(S8[7], S9[4])$ is marked as the template node and boundary conditions are updated. Similarly, $(S8[6], S9[3])$ in Figure 4.5c is marked. Since $(S9[2].parent, S8[2].parent)$ is a template match, current boundaries $left = S8[2].parent.bd = 4$,$up = S9[2].parent.bd = 1$,$right = 5$ and $bottom = 2$.$(S8[2], S9[2])$ is not within the boundaries. Thus it is not marked as template and alternative non-dominant match $(S8[5], S9[2])$ is selected. Similarly, for $(S8[1], S9[1])$, current boundaries $left$ and up are defined by $S8[5].bd = 4$ and $S9[2].bd = 1$. Thus $(S8[1], S9[1])$ is discarded and $(S8[4], S9[1])$ is marked template match in Figure 4.5e. Figure 4.5f shows all node mapping pairs indicated by template matches.

Let d denote the number of dominant matches. During the template match identification phase, at most $3d$ matches are visited and d matches are marked as template match. When a match is marked as a template match, its boundaries are computed. The time complexity of boundary-computation operation is dependent on how many times a $\sigma \in \Sigma$ appears in a sequence, the upper bound is $\log n + \log m$, where n and m are the size of Prüfer sequences. In the preprocess phase, it takes linear time to construct Prüfer sequences. In the subsequence matching phase, the LCS used in our work has a time complexity of $O(n + min\{pm \log s, p(n-p) \log s\})$, where p is the size of a LCS and s is the size of the alphabet. Thus the time complexity of Prüfer sequence-based tree template detection algorithm has a upper bound $O(m + 2n + min\{pm \log s, p(n-p) \log s\} + d + 2d(\log n + \log m))$. In [31], it has been proved that $d \leq p(m + n - 2p + 1)$. Thus, this algorithm is fast when $n \approx m$ and the p is expected to long or short. In the situation of template detection, since the given web pages are supposed to be generated from the same template, their DOM trees are similar and thus their Prüfer sequences are similar. This makes the Prüfer sequence-based algorithm suitable for template detection.

One drawback of the Prüfer sequence-based tree template detection algorithm is that sizes of sequences may be larger than sizes of trees. According to the construction rules of the Prüfer sequence, a node in the tree may appear multiple times in the



Mappings indicated by match(S8[8],S9[5]) ,(S8[7],S9[4]), (S8[6],S9[3]) , (S8[5],S9[2]) and (S8[4],S9[1])

Figure 4.5: Step by step template match identification based on Prüfer sequence

sequence depending on how many children it has. If a tree has a large fanout, then the size of its Prüfer sequence could be very large. This will has a negative impact on the performance of the Prüfer sequence-based tree template detection algorithm.

Consolidated Prüfer sequence-based Template Match Identification

The Consolidated Prüfer sequence-based algorithm is based on the similar idea as the previous Prüfer sequence-based algorithm. The size of a Consolidated Prüfer sequence is exactly the same as the size of the corresponding tree. So the Consolidated Prüfer sequence has shorter sequence length compared with the classical Prüfer sequence. The Consolidated Prüfer sequence-based algorithm can take less time to compute dominant matches and retrieve the template.

Theorem 3. *Let p denote to the length of an LCS of sequence A and B. A and B are Prüfer sequences of trees T_1 and T_2. Then D_p ,the set of dominant p-matches has only one match $(A[i], B[j])$ and nodes $A[i]$ and $B[j]$ are roots of tree T_1 and T_2.*

Theorem 3 is obvious since each node of a given tree appears exactly once in its corresponding Consolidated Prüfer sequence and root node is always the last node to be visited. So, similar to Prüfer sequence-based algorithm, we start from D_p and backtrack to D_1 to identify template matches.

For each $A[i] \in A$ and $B[j] \in B$, we also add a boundary attribute to it. This boundary attribute is defined the same as in the previous section.

If current template match is $(A[i], B[j])$ and $(A[i], B[j]) \in C_k$, $(A[s], B[t]) \in C_{k\prime}$ is picked as next template match if it satisfy the following conditions:

1. $(A[s].parent, B[t].parent)$ is a template match
2. $s < i$ and $t < j$
3. $A[s].parent.bd \leq s$ and $B[t].parent.bd \leq t$
4. $k\prime < k$ and the value of $k\prime$ is closest to k

Algorithm 3 identifyTemplateMatches-PS($dm[1, 2, \ldots, p]$)

```
1: $(A[i], B[j]) \leftarrow$ remove the first match from $dm[p]$
2: mark $(A[i], B[j])$ as a template match
3: $mapping[A[i]] \leftarrow B[j]$,$rmapping[B[j]] \leftarrow A[i]$
4: $right \leftarrow j$,$bottom \leftarrow i$
5: $computeBoundaries((A[i], B[j]))$
6: for $k \leftarrow p - 1$ to 1 do
7:     $found \leftarrow false$
8:     while $found = false$ AND $dm[k] \neq \emptyset$ do
9:         $(A[i\prime], B[j\prime]) \leftarrow$ remove the first match from $dm[k]$
10:        if $i\prime \leq right$ AND $j\prime \leq bottom$ then
11:            if $(A[i\prime], B[j\prime])$ is already a template match then
12:                $right \leftarrow j\prime$,$bottom \leftarrow i\prime$
13:                $computeBoundaries((A[i\prime], B[j\prime]))$
14:                $found \leftarrow true$
15:                continue
16:            end if
17:            if $(A[i\prime].parent, B[j\prime].parent)$ is a template match then
18:                if $A[i\prime].parent.bd \leq i\prime \leq bottom$ AND $B[j\prime].parent \leq j\prime \leq right$ then
19:                    mark $(A[i\prime], B[j\prime])$ as a template match;
20:                    $right \leftarrow j\prime$,$bottom \leftarrow i\prime$
21:                    $mapping[A[i\prime]] \leftarrow B[j\prime]$,$rmapping[B[j\prime]] \leftarrow A[i\prime]$
22:                    $computeBoundaries((A[i\prime], B[j\prime]))$
23:                    $found \leftarrow true$
24:                    continue
25:                end if
26:                if either $i\prime$ or $j\prime$ is not in the boundaries then
27:                    $getAlternative \leftarrow true$
28:                end if
29:                if at least one of $A[i\prime].parent$ and $B[j\prime].parent$ is mapped to some node then
30:                    $getAlternative \leftarrow true$
31:                end if
32:                if $(A[i\prime], B[j\prime])$ is dominant match AND $getAlternative = true$ then
33:                    if $i\prime$ is not in the boundaries then
34:                        $parent\_A \leftarrow rmapping[B[j\prime].parent]$
35:                        find $l$,$l \leftarrow min\{k \mid A[k].label = A[i\prime].label \wedge k \in [parent\_A.bd, bottom]\}$
36:                        append non-dominant match $(A[l], B[j\prime])$ to $dm[k]$
37:                    end if
38:                    if $j\prime$ is not in the boundaries then
39:                        $parent\_B \leftarrow mapping[A[i\prime].parent]$
40:                        find $l$,$l \leftarrow min\{k \mid B[k].label = B[j\prime].label \wedge k \in [parent\_B.bd, right]\}$
41:                        append non-dominant match $(A[i\prime], B[l])$ to $dm[k]$
42:                    end if
43:                end if
44:            end if
45:        end if
46:    end while
47: end for
```

Condition 1, and 2 ensure that identified template matches can form a mapping which is consistent with the definition of restricted top-down tree mapping. Different from previous Prüfer sequence-based algorithm, the boundary attribute of a node $A[i]$ here does not give the exact range of the subtree rooted at $A[i]$. However, matches satisfy Condition 1, and 2 still give a mapping which is consistent with the definition of restricted top-down mapping for labeled, rooted and ordered trees. Because each node of a given tree appears exactly once in its corresponding Consolidated Prüfer sequence, each node would be visited at most once. Backtracking from D_p and following Condition 2 is similar to do top-down and right to left traversal on both trees. This guarantees that mapping found is consistent with the definition of the restricted top-down mapping. Since all template match are dominant matches, Similar to previous section, Condition 3 and 4 Condition 3 can be used to find potential non-dominant template matches.

Algorithm 4 gives the procedure of finding template matches. The input array $dm[1, 2, \ldots, p]$ contains p lists of dominant matches and p is the length of the LCS. Each list $dm[i]$,$1 \leq i \leq p$, consists of all dominant matches in D_i. The algorithm starts by marking the *root-root* match in D_p as the template match. Then the boundary attributes of root nodes are computed and the the root-root match is set as current match. Then the algorithm backtracks to D_1 to identify template matches. At Line 9, a match $(A[i\prime], B[j\prime])$ is removed from the head of $dm[k]$. $(A[i\prime], B[j\prime])$ satisfying boundary conditions is marked as the template match if $(A[i\prime].parent, B[j\prime].parent)$ is a template match (Line 11-18). In other cases, we find alternative non-dominant matches that satisfy boundary conditions (Line 26-37). This is the same as Algorithm 3 does.

Figure 4.6 gives an example showing a step-by-step template match identification based on Consolidated Prüfer sequence. The input trees are T_8 and T_9 given in Figure 4.4a. In figure 4.6, entries circled by solid lines are the dominant matches obtained in the subsequence matching phase. In Figure 4.6a, the root-root match $(CS8[6], CS9[4])$ is initialized as template match which is denoted by the dash circle. Since $CS8[6]$ and $CS9[4]$ are root nodes, $CS8[6].bd = 1$ and $CS9[4].bd = 1$. The current boundary conditions are indicated by the rectangular. In Figure 4.6b, $(CS8[5], CS9[3])$ is within the boundaries and $(CS8[5].parent, CS9[3].parent)$ is a template match. Thus, $(CS8[5], CS9[3])$ is marked as the template match. Since $CS8[2].label = CS8[5].label$ and $2 > CS[5].parent.bd = 1$, one boundary $left = 3$. $CS9[3].label$ appears only once in $CS9$, thus another boundary $up = 1$. Since $(CS8[5], CS9[3])$ is current template match, current boundary conditions are $left = 3$,$up = 1$,$right = 4$ and $bottom = 2$. Boundary conditions are denoted by the rectangular in Figure 4.6b. Similarly, $(CS8[4], CS9[2])$ in Figure 4.6c is marked. Since $CS9[1].parent = CS9[3]$ is marked and $CS8[1].parent = CS8[2]$ is not marked, $CS8[1]$ can not be a template node because its parent is not a template node. Thus the algorithm checks the first row to see if there is any non-dominant match satisfying the requirement of template match. Non-dominant match $(S8[5], S9[2])$ is selected and marked as template match in Figure 4.6d. Figure 4.6e shows all node mapping pairs indicated by template matches.

Thus the time complexity of Prüfer sequence-based tree template detection algorithm has a upper bound $O(m + 2n + min\{pm \log s, p(n-p) \log s\} + d + 2d(\log n + \log m))$. n and m are the size of two input trees. Like in the previous algorithm, p is the length of the LCS, s is the size of the alphabet and d is number of dominant matches. The upper bound on d is $p(m + n - 2p + 1)$[31].

Template Retrieval

Once template matches are identified, the template tree structure can be easily obtained by traversing an input tree and removing nodes which are not marked. Algorithm 5 presents the complete procedure of template detection.

Discussion

One potential limitation of our sequence-based algorithms is that they may return a much smaller template structure than the one returned by the top-down template detection algorithm under certain situations. Figure 4.8 and Figure 4.9 give an example. Two input trees T_{10} and T_{11} are shown in Figure 4.7a. In Figure 4.7b, gray nodes of T_{10} and T_{11}

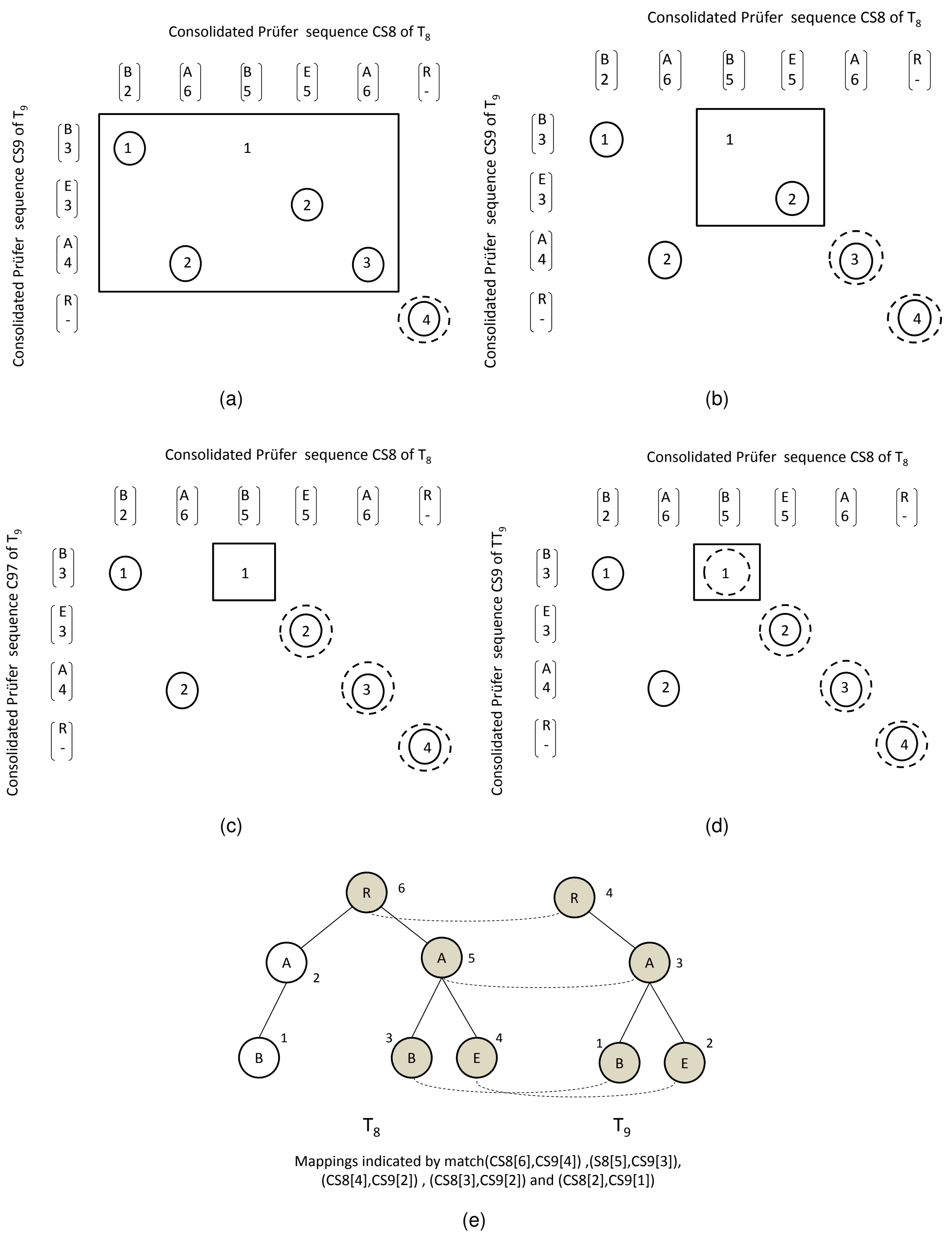

Consolidated Prüfer sequence CS8 of T_8
Consolidated Prüfer sequence CS9 of T_9
(a)
(b)
Consolidated Prüfer sequence C97 of T_9
Consolidated Prüfer sequence CS9 of TT_9
(c)
(d)
T_8
T_9
Mappings indicated by match(CS8[6],CS9[4]) ,(S8[5],CS9[3]), (CS8[4],CS9[2]) , (CS8[3],CS9[2]) and (CS8[2],CS9[1])
(e)

Figure 4.6: Step by step template match identification based on Consolidated Prüfer sequence

Algorithm 4 identifyTemplateMatches-CPS($dm[1, 2, \ldots, p]$)

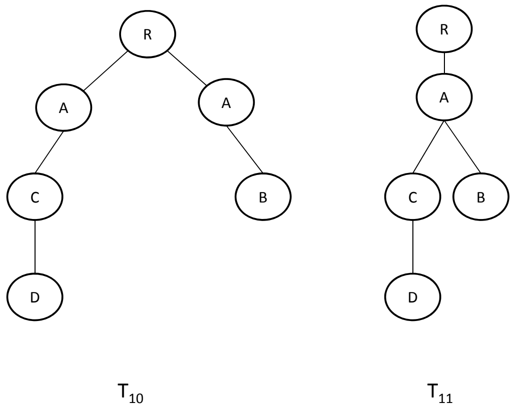
```
1:  $(A[i], B[j]) \leftarrow$ remove the first match from $dm[p]$
2:  mark $(A[i], B[j])$ as a template match
3:  $mapping[A[i]] \leftarrow B[j]$,$rmapping[B[j]] \leftarrow A[i]$
4:  $right \leftarrow j$,$bottom \leftarrow i$
5:  $computeBoundaries((A[i], B[j]))$
6:  for $k \leftarrow p-1$ to 1 do
7:      $found \leftarrow false$
8:      while $found = false$ AND $dm[k] \neq \emptyset$ do
9:          $(A[i\prime], B[j\prime]) \leftarrow$ remove the first match from $dm[k]$
10:         if $i\prime \leq right$ AND $j\prime \leq bottom$ then
11:             if $(A[i\prime].parent, B[j\prime].parent)$ is a template match then
12:                 if $A[i\prime].parent.bd \leq i\prime < bottom$ AND $B[j\prime].parent \leq j\prime < right$ then
13:                     mark $(A[i\prime], B[j\prime])$ as a template match;
14:                     $right \leftarrow j\prime$,$bottom \leftarrow i\prime$
15:                     $mapping[A[i]] \leftarrow B[j]$,$rmapping[B[j]] \leftarrow A[i]$
16:                     $computeBoundaries((A[i\prime], B[j\prime]))$
17:                     $found \leftarrow true$
18:                     continue
19:                 end if
20:                 if either $i\prime$ of $j\prime$ is not in the boundaries then
21:                     $getAlternative \leftarrow true$
22:                 end if
23:                 if at least one of $A[i\prime].parent$ and $B[j\prime].parent$ is mapped to some node then
24:                     $getAlternative \leftarrow true$
25:                 end if
26:                 if $(A[i\prime], B[j\prime])$ is dominant match AND $getAlternative = true$ then
27:                     if $i\prime$ is not in the boundaries then
28:                         $parent\_A \leftarrow rmapping[B[j\prime].parent]$
29:                         find $l$,$l \leftarrow min\{k \mid A[k].label = A[i\prime].label \wedge k \in [parent\_A.bd, bottom]\}$
30:                         append non-dominant match $(A[l], B[j\prime])$ to $dm[k]$
31:                     end if
32:                     if $j\prime$ is not in the boundaries then
33:                         $parent\_B \leftarrow mapping[A[i\prime].parent]$
34:                         find $l$,$l \leftarrow min\{k \mid B[k].label = B[j\prime].label \wedge k \in [parent\_B.bd, right]\}$
35:                         append non-dominant match $(A[i\prime], B[l])$ to $dm[k]$
36:                     end if
37:                 end if
38:             end if
39:         end if
40:     end while
41: end for
```

Algorithm 5 PS/CPS-TD(T_1, T_2)

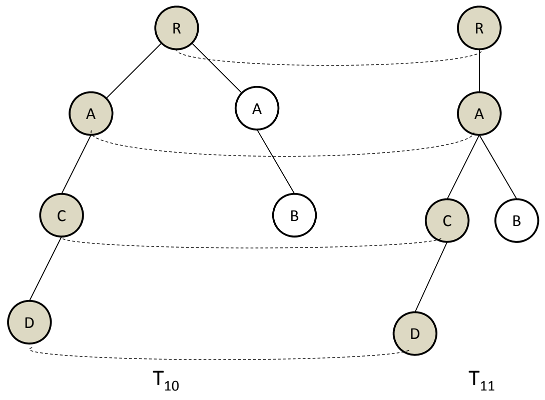
```
1: $S_1 \leftarrow$ Prüfer/Consolidate Prüfer sequence of $T_1$
2: $S_2 \leftarrow$ Prüfer/Consolidate Prüfer sequence of $T_2$
3: $dominan\_matches \leftarrow$ computeLCS($S_1, S_2$)
4: identifyTemplateMatches($dominant\_matches$)
5: $T_S \leftarrow$ traverse $T_1$ and remove nodes which are not marked
6: return $T_S$
```

indicate the template nodes and dash lines are mapping pairs found by TD-TD. The template structure found by PS-TD and CPS-TD are shown in Figure 4.8 and Figure 4.9.

If a match $(A[i], B[j]) \in C_k$, this means that the length of the longest common subsequence between $A[1]A[2]\dots A[i]$ and $B[1]B[2]\dots B[j]$ is k. As we discussed in Chapter 4, k can be treated as the upper bound of the size of the common forests of the forests represented by $A[1]A[2]\dots A[i]$ and $B[1]B[2]\dots B[j]$. In our sequence-based algorithms, $(A[i], B[j]) \in C_k$ is chosen as a template match if it qualifies for a template match and k is the largest possible value. However, this may not lead to an optimal result. k only gives the upper bound of the size of the possible common forest between the remaining unvisited parts of input trees. For instance, in Figure 4.8b and Figure 4.9b, if we choose $(S11[3], S10[3])$ instead of $(S11[5], S10[6])$ in Figure 4.8b and $(CS11[3], CS10[3])$ instead of $(CS11[3], CS10[5])$ in Figure 4.9b, we could find the same result shown in Figure 4.7b. Of course, we can try all matches which qualify for template matches and return the template structure having the largest size. This would be impractical since it is similar to enumerating all possible LCSs, which is very costly.

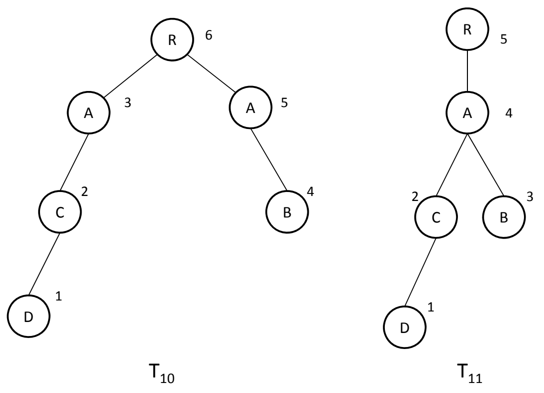


(a) Two input trees T_{10} and T_{11}

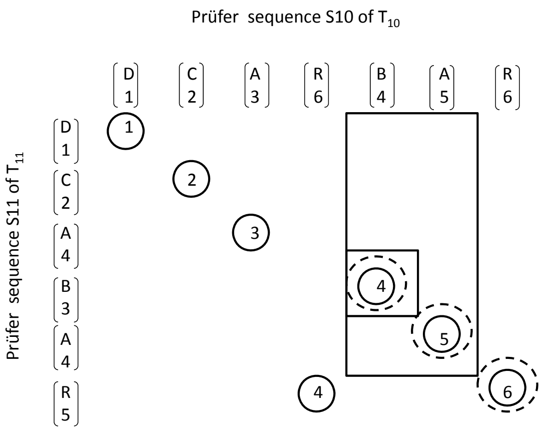


(b) Optimal template found by TD-TD

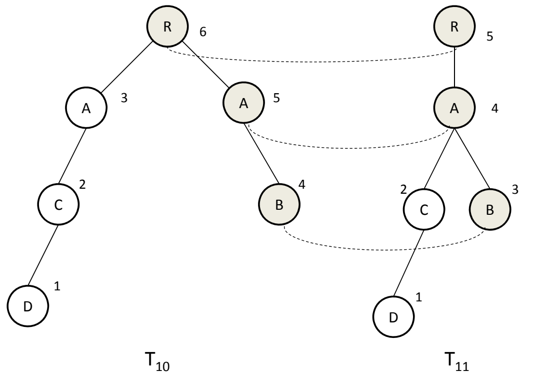
Figure 4.7: Example of limitations of sequence-based algorithms:Input trees and optimal template



(a) Input trees

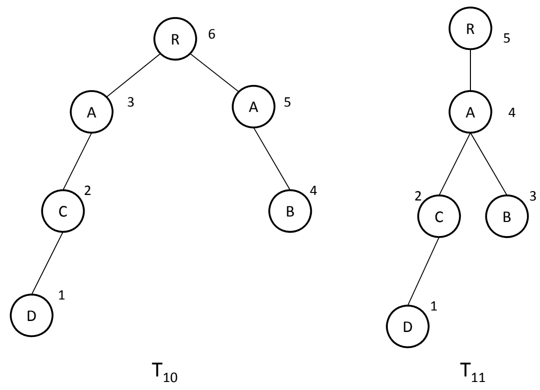


(b) Template matches detected by Algorithm 3

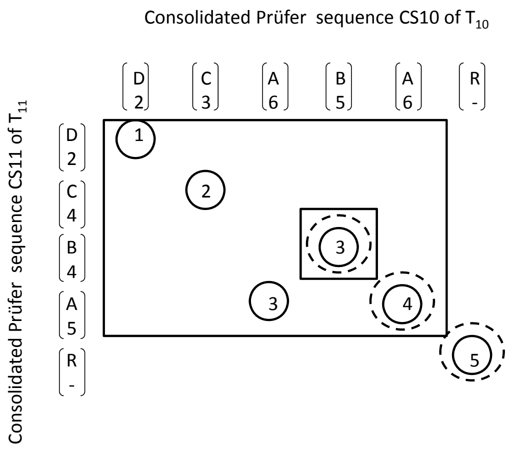


(c) Template found by PS-TD

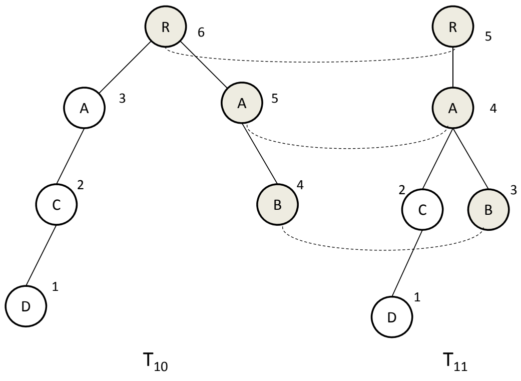
Figure 4.8: Example of limitations of sequence-based algorithms:Template found by PS-TD



(a) Input trees


Consolidated Prüfer sequence CS10 of T10
Consolidated Prüfer sequence CS11 of T11
D 2
C 3
A 6
B 5
A 6
R -
D 2
C 4
B 4
A 5
R -
1
2
3
3
4
5

(b) Template matches detected by Algorithm 4



(c) Template found by CPS-TD

Figure 4.9: Example of limitations of sequence-based algorithms:Template found by CPS-TD

Chapter 5

EXPERIMENTAL EVALUATION

For simplicity, in this section, the restricted top-down template detection algorithm, the restricted bottom-up template detection algorithm, the Prüfer sequence-based template detection algorithm and the Consolidated Prüfer sequence-based template detection algorithm are denoted by TD-TD,BU-TD,PS-TD and CPS-TD accordingly.

Experimental Setup

Our dataset consists of 10 sets of web pages from 5 web sites. Web pages in each set are from the same category of a web site. Table 5.1 shows a brief description of the data set.

For all web pages, we applied the CyberNeko HTML parser to obtain the DOM tree and to fix common HTML errors.

Effectiveness Evaluation

In the effectiveness evaluation, we conducted two types of evaluations.

In the first type of effectiveness evaluation, templates returned by TD-TD were used as the reference set because they were optimal in terms of tree size. We then compared templates detected by other methods with the reference set. In this evaluation,

Site	Category	No. of pairs	Published dates	Name of test set
CNET	Latest News	91	6.4,2010-6.7,2010	CNET-Latest News
CNN	Money	120	3.10,2011-3.28,2011	CNN-Money
CNN	Travel	91	3.25,2011-4.6,2011	CNN-Travel
CNN	Entertainment	120	3.1,2011-3.29,2011	CNN-Entertainment
BBC	Entertainment	21	5.24,2010-5.28,2010	BBC-Entertainment
BBC	Economics	91	5.20,2010-5.25,2010	BBC-Economics
MSN	Business	55	6.2,2010-6.7,2010	MSN-Business
MSN	InGame Blog	105	3.24,2011-3.29,2011	MSN-InGame Blog
Yahoo	News	55	3:00AM-7:30PM,3.29,2011	Yahoo-News
NPR	Music	91	3.16,2011-3.29,2011	NPR-Music

Table 5.1: Brief description of data set

the order of nodes was not considered. In other words, we only evaluated how many overlaps there were between the optimal results and results returned by PS-TD,CPS-TD and BU-TD. There was no penalty if a returned result was not actually a common subtree.

In this evaluation, we first ran the TD-TD on each test pair to generate the reference set R containing the nodes in the template. Then we applied PS-TD,CPS-TD and BU-TD, on the same pair to generate corresponding result sets T_{PS} ,T_{CPS}, and T_{BU}. Each set of T_{PS} ,T_{CPS}, and T_{BU} is compared with R using precision, recall and the F-score.

Figure 5.1 shows the average precision, recall and the F-score of BU-TD, PS-TD and CPS-TD for each set. As we can see from Figure 5.1b and Figure 5.1c, in all cases, PS-TD and CPS-TD achieved precision and recall which were both close to 1. This means that the template nodes detected by the sequence-based algorithms were almost identical to the optimal results obtained by TD-TD.

Between two sequence-based algorithms, the Consolidated Prüfer sequence-based algorithm achieved better results than the classical Prüfer sequence-based algorithm in general. One explanation could be that the dominant matches found in the subsequence matching phase is more close to the optimal restricted top-down node mapping pairs, since all nodes of a tree appear only once in the Consolidated Prüfer sequence.

Since we were trying to find a common subtree of DOM trees, in the second type of effectiveness evaluation, penalties were given if results were not common subtrees. In this evaluation, results of PS-TD,CPS-TD and BU-TD were also compared with results of TD-TD using precision, recall and F-score. Different from the first evaluation, penalty was assigned if a result returned by PS-TD, CPS-TD or BU-TD was not a common subtree between two input trees. If one method gave a result which was not a common subtree of input trees, for that test case, the precision, recall and F-score of that method were assigned 0.

From Figure 5.2 and Figure 5.1, we can see that precision, recall and F-socre of

	BU-TD	PS-TD	CPS-TD
CNET-Latest News	0.907	0.968	**0.977**
CNN-Money	0.985	0.995	**0.997**
CNN-Travel	0.951	0.993	**0.994**
CNN-Entertainment	0.977	0.993	**0.994**
BBC-Entertainment	0.884	**0.964**	0.96
BBC-Economics	0.891	0.935	**0.948**
MSN-Bussiness	0.963	0.993	**0.998**
MSN-InGame Blog	0.978	0.994	**0.995**
Yahoo-News	0.939	0.995	**0.998**
NPR-Music	0.951	0.982	**0.988**

(a) F-scores

	BU-TD	PS-TD	CPS-TD
CNET-Latest News	0.905	0.988	**0.992**
CNN-Money	0.981	0.998	**0.998**
CNN-Travel	0.94	0.997	**0.998**
CNN-Entertainment	0.973	0.994	**0.995**
BBC-Entertainment	0.874	0.968	**0.975**
BBC-Economics	0.877	0.952	**0.964**
MSN-Bussiness	0.982	0.996	**0.999**
MSN-InGame Blog	0.989	0.995	**0.996**
Yahoo-News	0.967	0.997	**0.998**
NPR-Music	0.929	0.99	**0.994**

(b) Precision

	BU-TD	PS-TD	CPS-TD
CNET-Latest News	0.915	0.95	**0.963**
CNN-Money	0.989	0.993	**0.996**
CNN-Travel	0.962	0.989	**0.99**
CNN-Entertainment	0.982	0.992	**0.993**
BBC-Entertainment	0.899	**0.96**	0.947
BBC-Economics	0.91	0.923	**0.934**
MSN-Bussiness	0.946	0.991	**0.998**
MSN-InGame Blog	0.968	0.993	**0.994**
Yahoo-News	0.913	0.994	**0.997**
NPR-Music	0.975	0.974	**0.983**

(c) Recall

Figure 5.1: Effectiveness evaluation without penalty

	BU-TD	PS-TD	CPS-TD
CNET-Latest News	0	0.968	**0.977**
CNN-Money	0.008	0.995	**0.997**
CNN-Travel	0	0.993	**0.994**
CNN-Entertainment	0	0.993	**0.994**
BBC-Entertainment	0	**0.964**	0.96
BBC-Economics	0	0.935	**0.948**
MSN-Bussiness	0	0.993	**0.998**
MSN-InGame Blog	0.022	0.994	**0.995**
Yahoo-News	0	0.995	**0.998**
NPR-Music	0	0.982	**0.988**

(a) F-scores

	BU-TD	PS-TD	CPS-TD
CNET-Latest News	0	0.988	**0.992**
CNN-Money	0.008	0.998	**0.998**
CNN-Travel	0	0.997	**0.998**
CNN-Entertainment	0	0.994	**0.995**
BBC-Entertainment	0	0.968	**0.975**
BBC-Economics	0	0.952	**0.964**
MSN-Bussiness	0	0.996	**0.999**
MSN-InGame Blog	0.022	0.995	**0.996**
Yahoo-News	0	0.997	**0.998**
NPR-Music	0	0.99	**0.994**

(b) Precision

	BU-TD	PS-TD	CPS-TD
CNET-Latest News	0	0.95	**0.963**
CNN-Money	0.008	0.993	**0.996**
CNN-Travel	0	**0.989**	0.99
CNN-Entertainment	0	0.992	**0.993**
BBC-Entertainment	0	**0.96**	0.947
BBC-Economics	0	0.923	**0.934**
MSN-Bussiness	0	0.991	**0.998**
MSN-InGame Blog	0.022	0.993	**0.994**
Yahoo-News	0	0.994	**0.997**
NPR-Music	0	0.974	**0.983**

(c) Recall

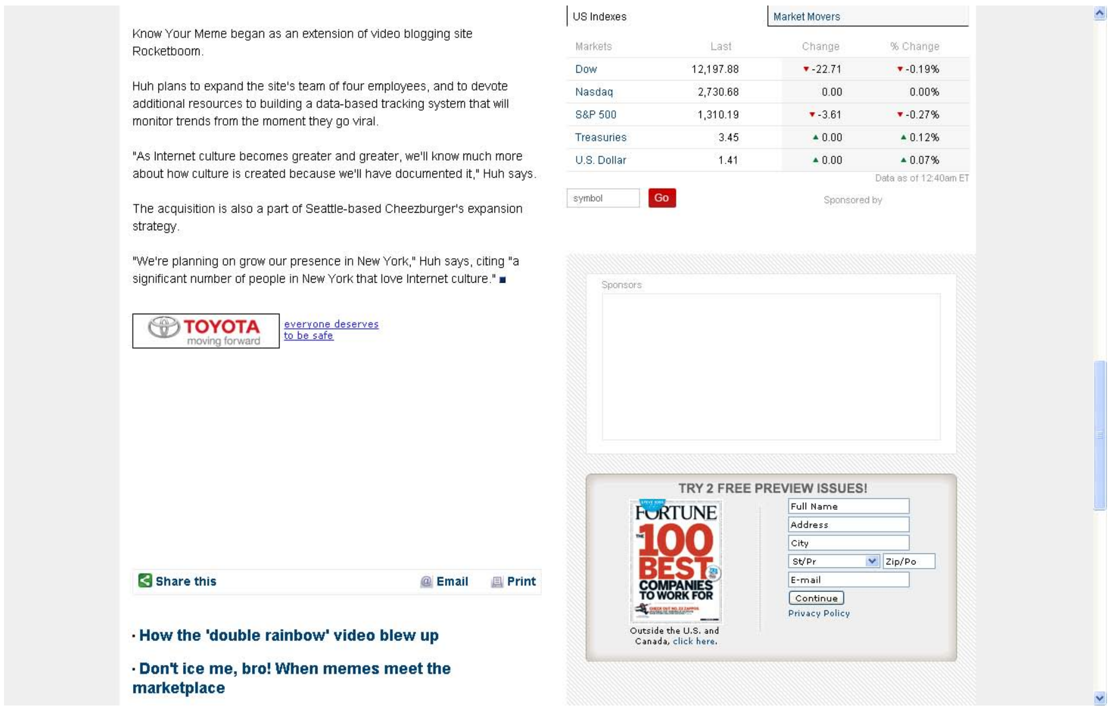
Figure 5.2: Effectiveness evaluation with penalty

PS-TD and CPS-TD are the same. It means that PS-TD and CPS-TD guarantee to give a common subtree of two input trees. On the other hand, we can see from Figure 5.2 that for 10 test sets, BU-TD could hardly return common subtrees of input trees. The main reason for BU-TD to achieve very poor results was that results violated the order constraints mensioned in the Background section. Figure 5.3 and Figure 5.4 shows an example. Figure 5.3a and Figure 5.3b are parts of 2 Web pages from test set “CNN-Money”. Both pages have tables showing “US Indexes”. Figure 5.4a and Figure 5.4b give parts of corresponding DOM structures of tables. Since the “change” of “Nasdaq” in Figure 5.3a and the “change” of “Treasuries” in Figure 5.3b have the same value and both are the first ones having the value “0.00” in the tables, BU-TD mapped the corresponding node of “change” of “Nasdaq” in Figure 5.3a to the corresponding node of “change” of “Treasuries” in Figure 5.3b. Due to the same reason, “Nasdaq” in Figure 5.3a was mapped to “Nasdaq” in Figure 5.3b and “2,730.68” in Figure 5.3a was mapped to “2,730.68” in Figure 5.3b. As shown in Figure 5.4a and Figure 5.4b, gray nodes denote to the template nodes marked by BU-TD. Obviously, the subtree consisted of template nodes in Figure 5.3a was not a common subtree of two DOM fragments in Figure 5.4.

Efficiency Evaluation

In this section, the efficiency of TD-TD, BU-TD, and our sequence-based algorithms is evaluated. Similar to the effectiveness evaluation, for each test set, all algorithms were ran on every pair of pages in that set. The top-down mapping algorithm, TD-TD, is usually the slowest. It also showed quadratic running time behaviors on test sets like “CNN-Entertainment”, “MSN-InGame Blog”,“NPR-Music”, and “CNET-LatestNews”. The execution time of BU-TD, grew linearly as the size of trees increases. PS-TD took more time than BU-TD in most cases, but it took much less time than TD-TD in all cases. For CPS-TD, the running time grew almost linearly.

From the results, we can see that the sequence-based algorithms were much faster than TD-TD. CPS-TD was even better than the bottom-up mapping algorithm in terms of execution time. One explanation for this is that BU-TD visited each node at least 6 times and sequence-based algorithms visited each node at most 3 times. As we expected,



Know Your Meme began as an extension of video blogging site Rocketboom.

Huh plans to expand the site's team of four employees, and to devote additional resources to building a data-based tracking system that will monitor trends from the moment they go viral.

"As Internet culture becomes greater and greater, we'll know much more about how culture is created because we'll have documented it," Huh says.

The acquisition is also a part of Seattle-based Cheezburger's expansion strategy.

"We're planning on grow our presence in New York," Huh says, citing "a significant number of people in New York that love Internet culture." ■

everyone deserves to be safe

Share this Email Print

· How the 'double rainbow' video blew up

· Don't ice me, bro! When memes meet the marketplace

US Indexes | Market Movers

Markets	Last	Change	% Change
Dow	12,197.88	▼-22.71	▼-0.19%
Nasdaq	2,730.68	0.00	0.00%
S&P 500	1,310.19	▼-3.61	▼-0.27%
Treasuries	3.45	▲0.00	▲0.12%
U.S. Dollar	1.41	▲0.00	▲0.07%

Data as of 12:40am ET

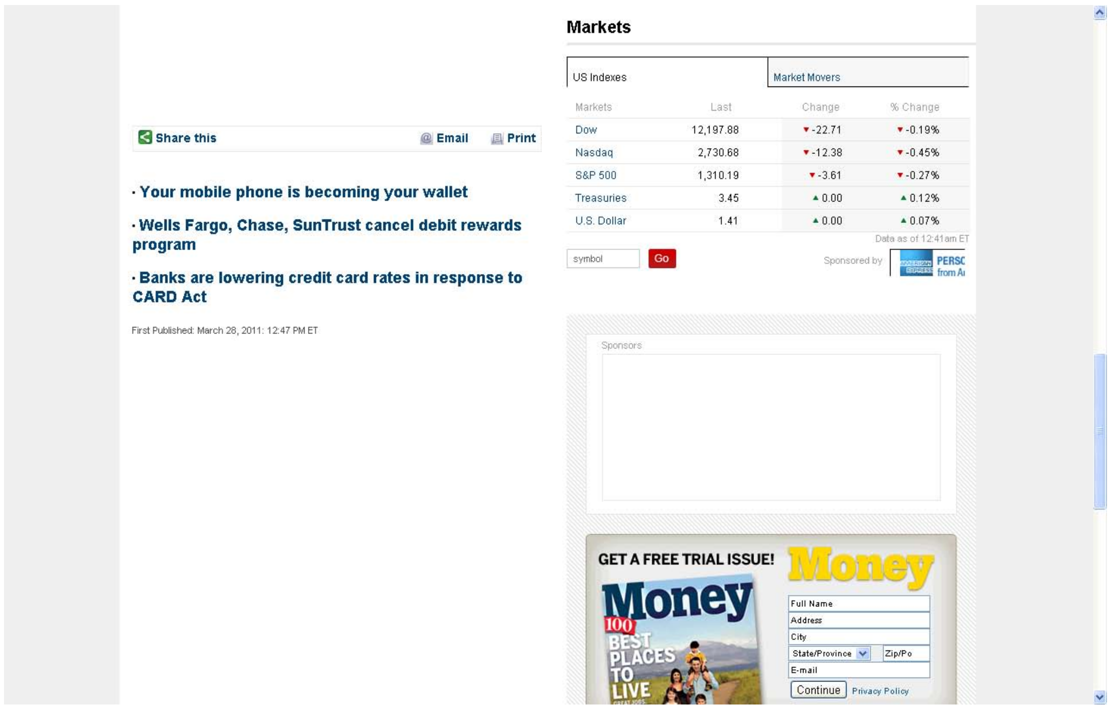
symbol Go

Sponsored by

Sponsors

TRY 2 FREE PREVIEW ISSUES!
FORTUNE
100 BEST COMPANIES TO WORK FOR
Outside the U.S. and Canada, click here.
Full Name
Address
City
St/Pr
Zip/Po
E-mail
Continue
Privacy Policy

(a) A page from CNN-Money



Share this Email Print

· Your mobile phone is becoming your wallet

· Wells Fargo, Chase, SunTrust cancel debit rewards program

· Banks are lowering credit card rates in response to CARD Act

First Published: March 28, 2011: 12:47 PM ET

Markets

US Indexes | Market Movers

Markets	Last	Change	% Change
Dow	12,197.88	▼-22.71	▼-0.19%
Nasdaq	2,730.68	▼-12.38	▼-0.45%
S&P 500	1,310.19	▼-3.61	▼-0.27%
Treasuries	3.45	▲0.00	▲0.12%
U.S. Dollar	1.41	▲0.00	▲0.07%

Data as of 12:41am ET

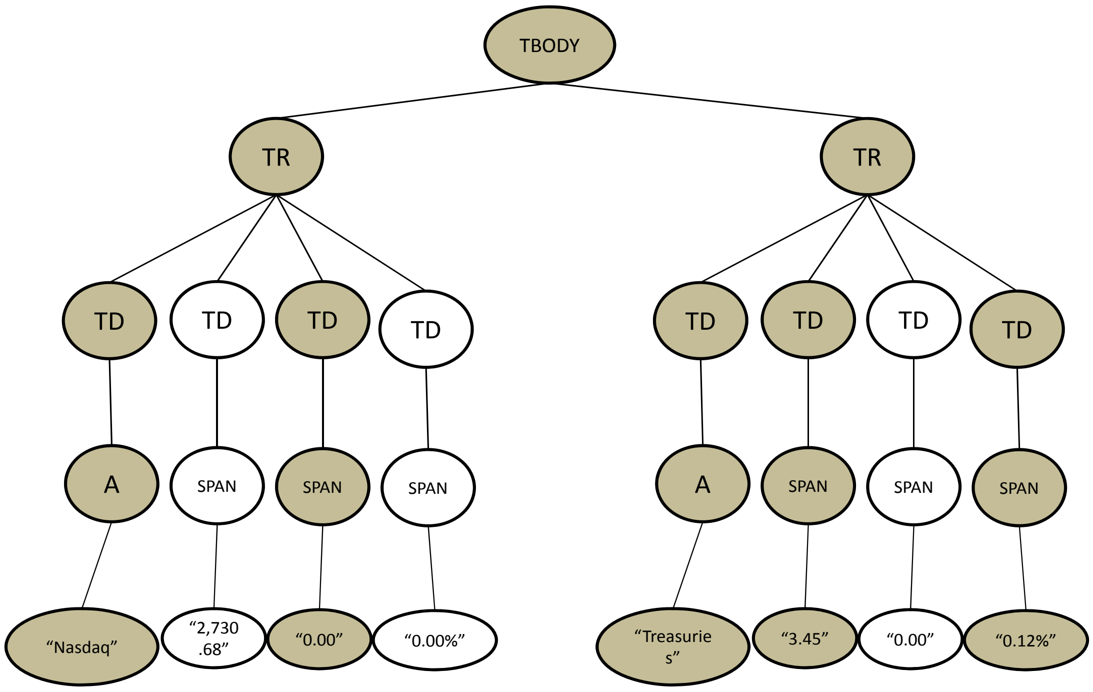
symbol Go

Sponsored by

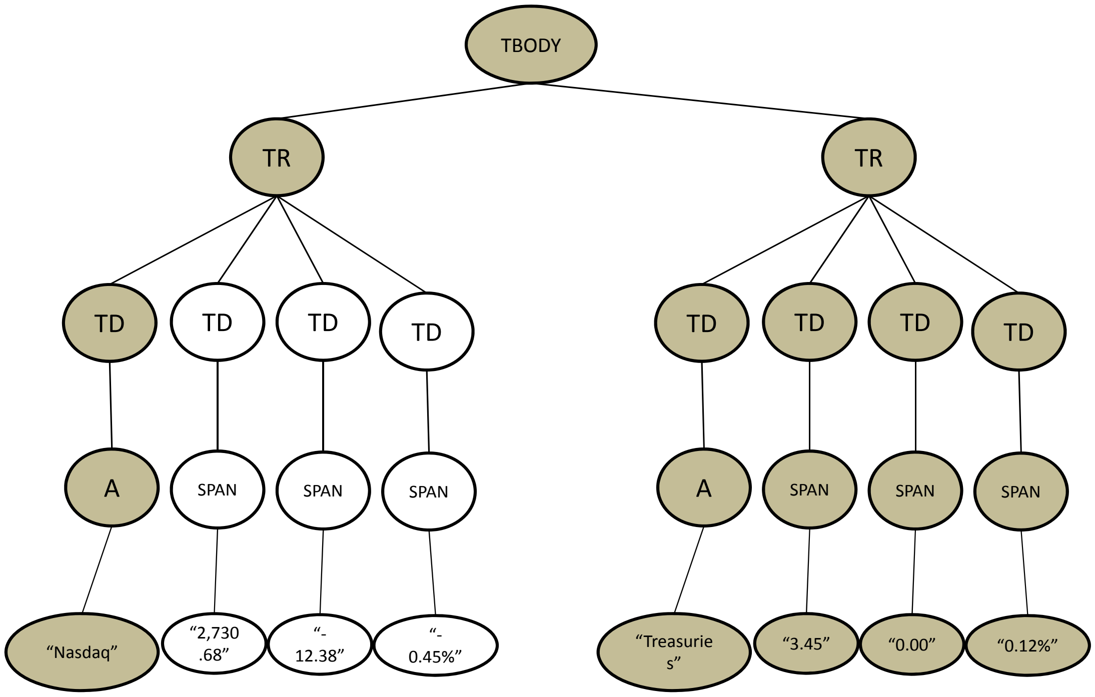
Sponsors

(b) Another page from CNN-Money

Figure 5.3: An example of BU-TD result



(a) Part of DOM trees of Figure 5.3a



(b) Part of DOM trees of Figure 5.3b

Figure 5.4: Part of Template nodes identified by BU-TD

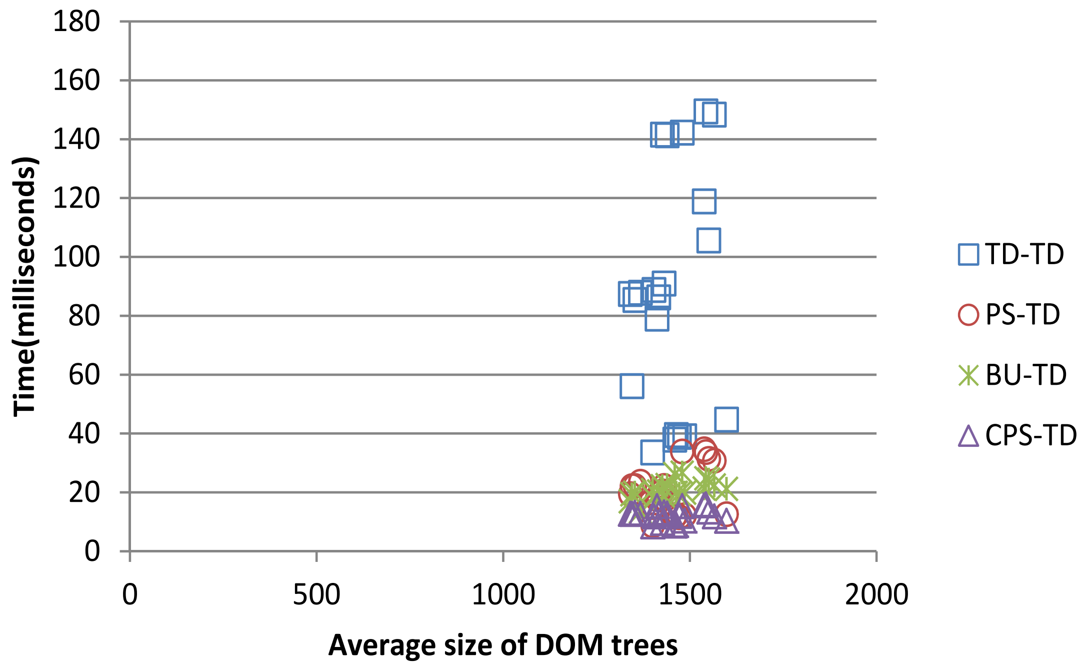
the CPS-TD had a better performance than PS-TD since Consolidated Prüfer sequence of a given tree is shorter than the corresponding Prüfer sequence.

The relationships between running time and F-score obtained without penalization are shown in Figure 5.10a and Figure 5.10b. Figure 5.10a shows for each test set, the average F-score and average execution time of all four approaches. From Figure 5.10a, we can see that sequence-based algorithms achieved F-score which was over 0.9 in a relatively short time. Figure 5.10b shows for all test pairs, the average F-score and average execution time of all four approaches. Figure 5.10b shows that on average, PS-TD and CPS-TD took about $1/4$ and $1/10$ running time of TD-TD to obtain near optimal results. Although BU-TD was also fast, the quality of its results were not as good as PS-TD and CPS-TD. Thus, CPS-TD could be a better choice for time-sensitive applications."Time vs F-score with penalization" are showed in Figure 5.11. From Figure 5.11, we can see that BU-TD was also fast, but it could hardly identify common subtrees of input DOM trees.

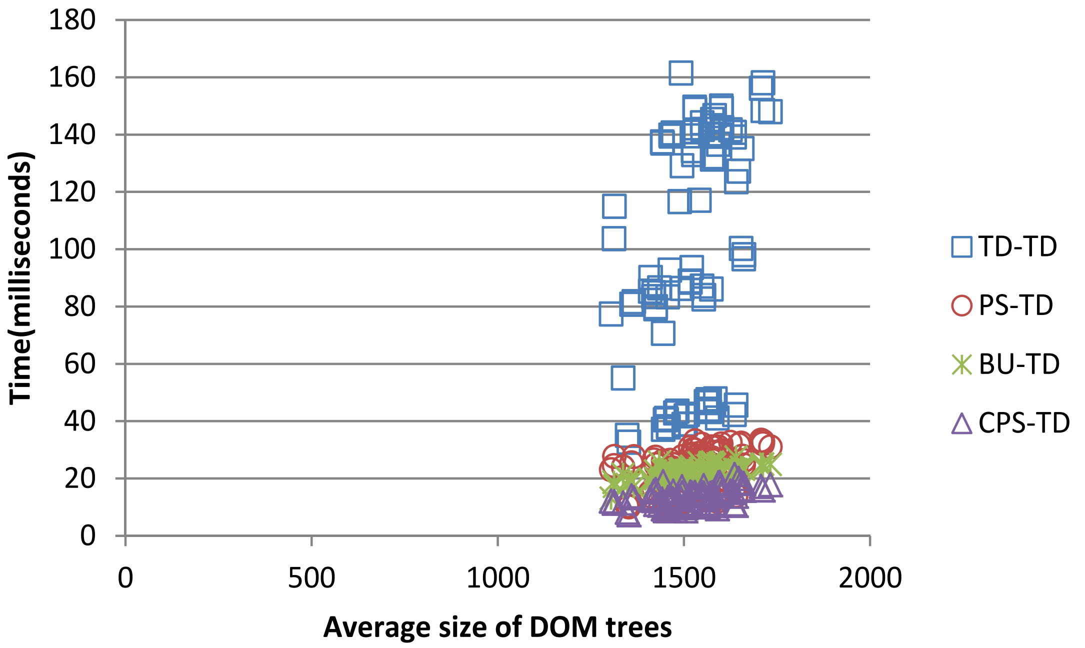
As we can see from Figure 5.10 and Figure 5.11, TD-TD could return optimal results at the cost of much more time than PS-TD and CPS-TD. On the hand, PS-TD and CPS-TD could give results of good quality with much less time than TD-TD. Though BU-TD was efficient, it can not guarantee to give common subtrees of input DOM trees.

Discussion

In this section, we investigate how changes between DOM tree could have impact on the running time behaviors of TD-TD, BU-TD, PS-TD and CPS-TD. Theoretically, the running time of each of the four algorithms is related to the number of nodes of input DOM trees. In addition, according to Vieira, the running time performance of TD-TD could be downgraded if most of changes between DOM trees are at leaf nodes [2]. Thus, we investigated the correlations between the running time and the following six parameters: *tree size*, *leaf size*, *total changes*, *leaf changes*, *total-change ratio* and *leaf-change ratio*.

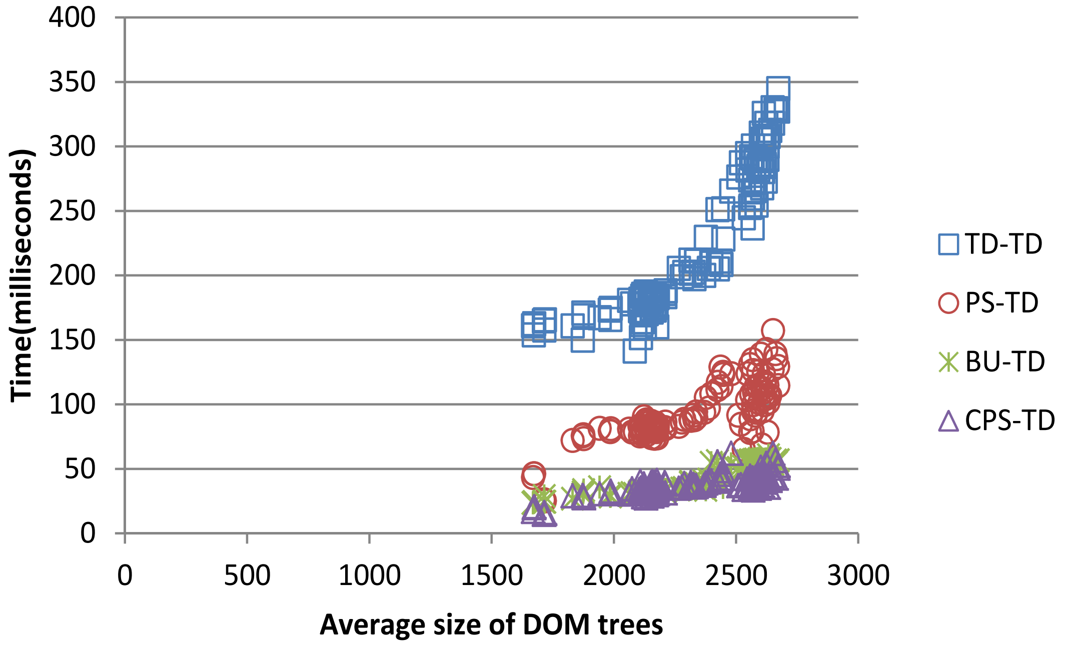


(a) BBC-Entertainment

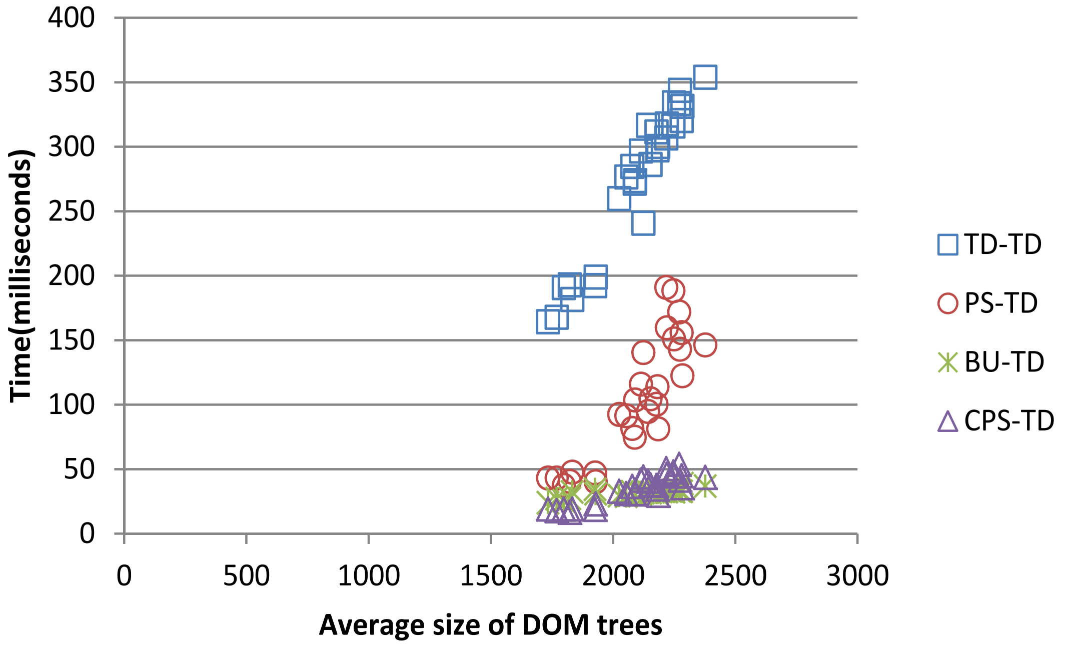


(b) BBC-Economics

Figure 5.5: Efficiency evaluation results-1

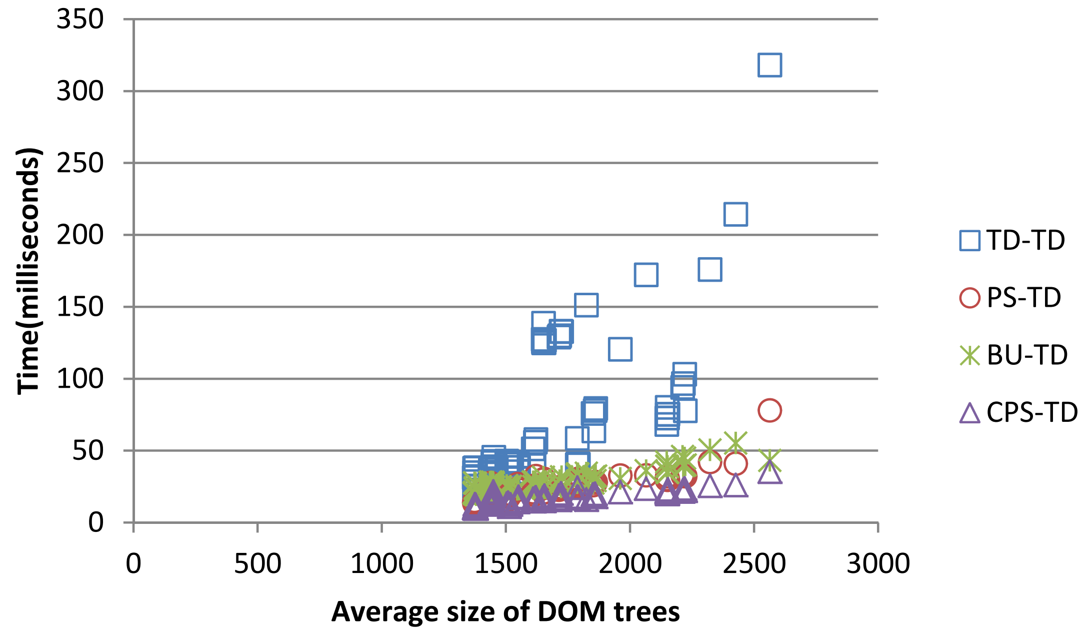


(a) CNN-Entertainment

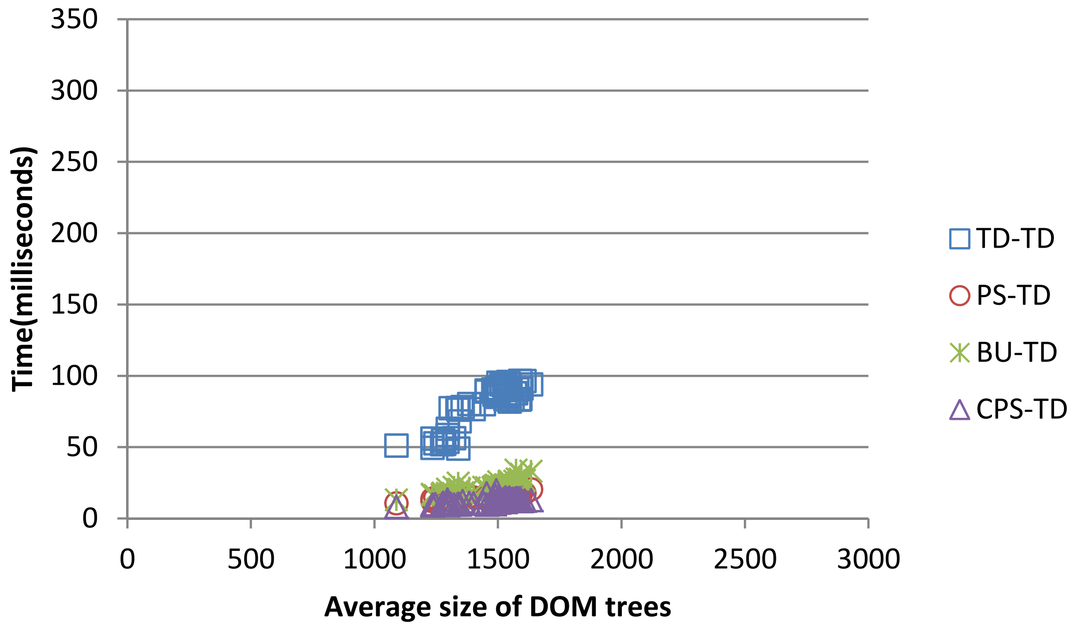


(b) CNN-Travel

Figure 5.6: Efficiency evaluation results-2

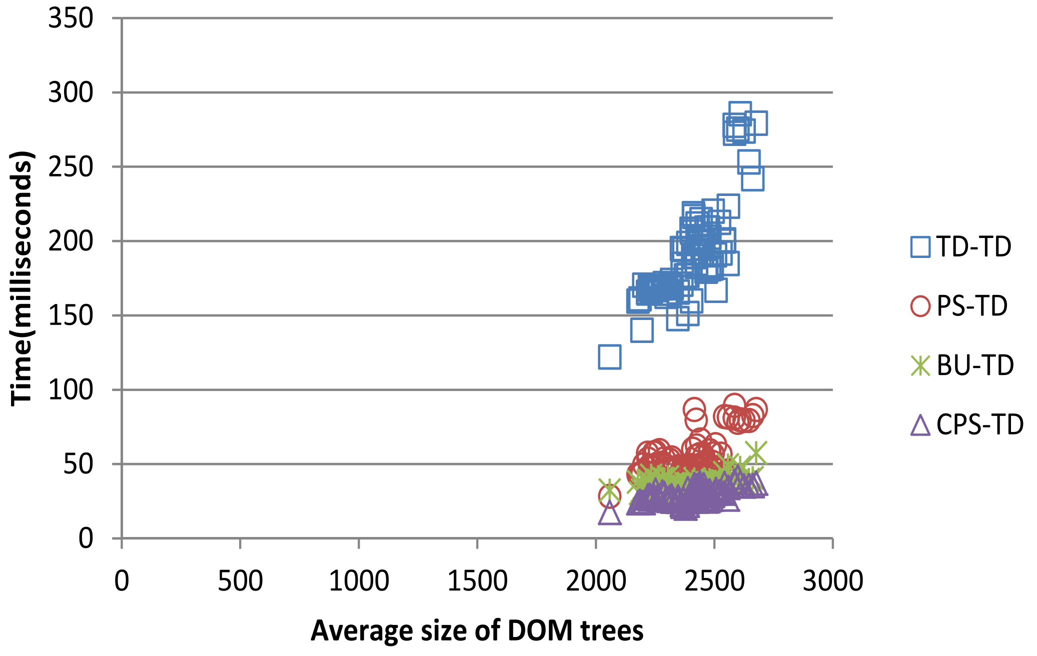


(a) MSN-InGame Blog

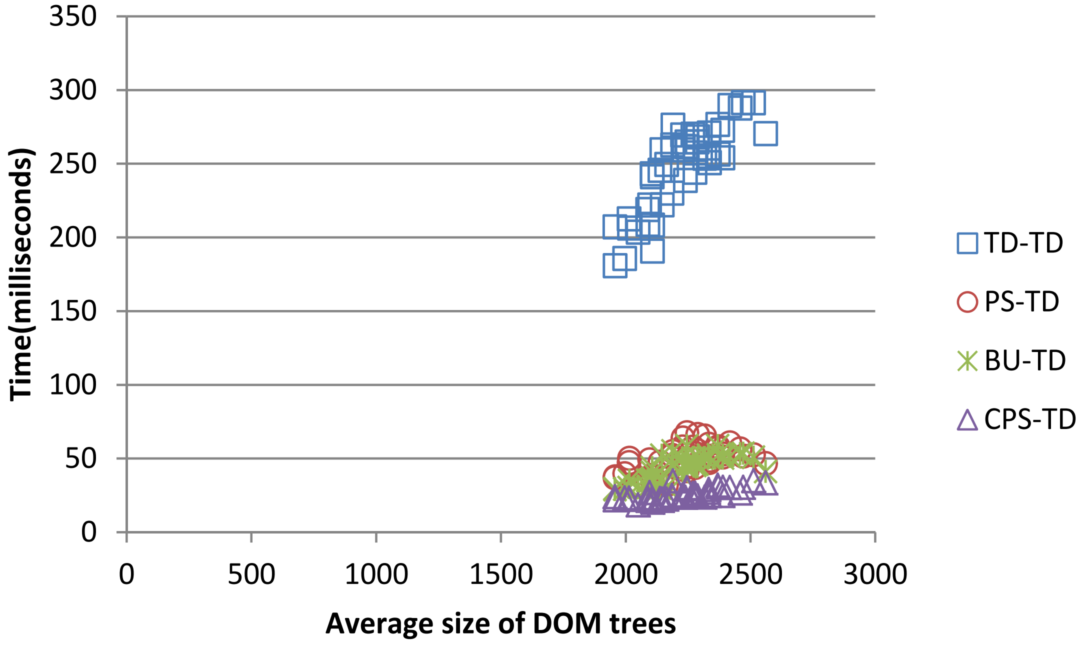


(b) MSN-Business

Figure 5.7: Efficiency evaluation results-3

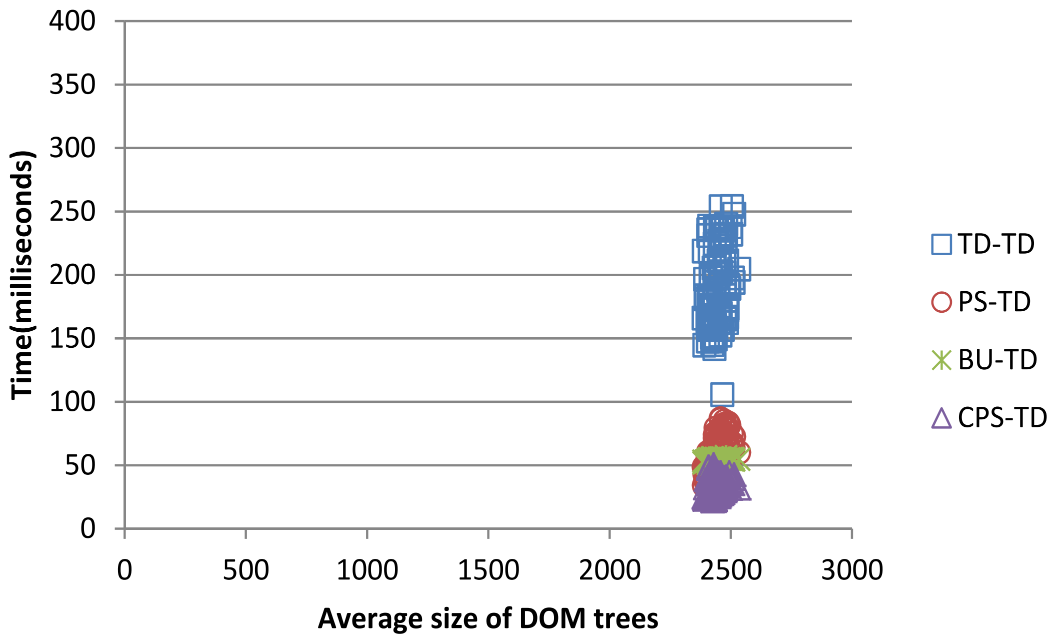


(a) NPR-Music

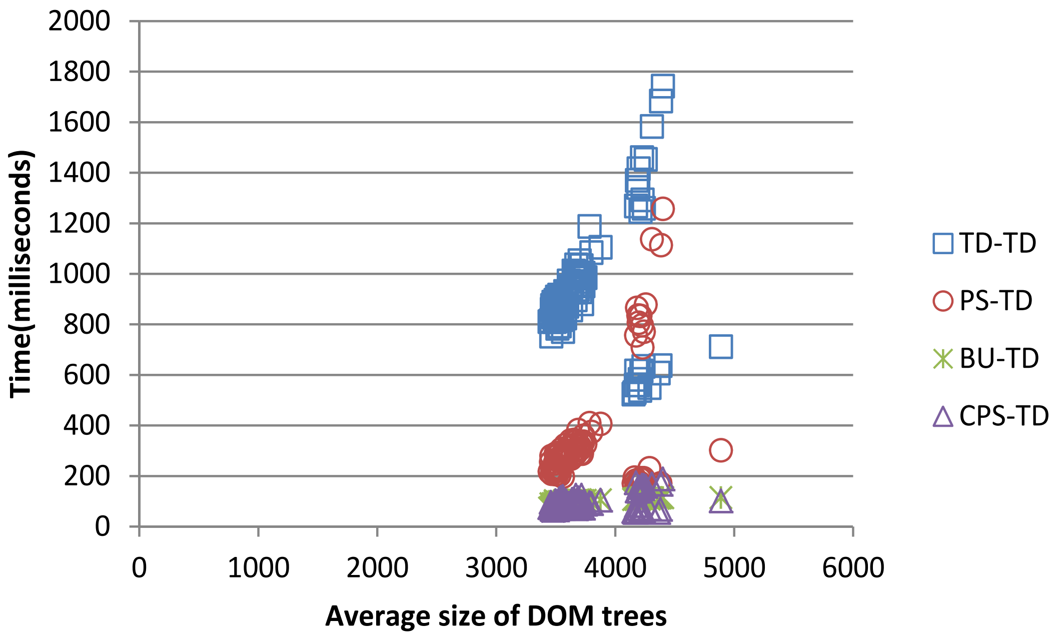


(b) Yahoo-News

Figure 5.8: Efficiency evaluation results-4

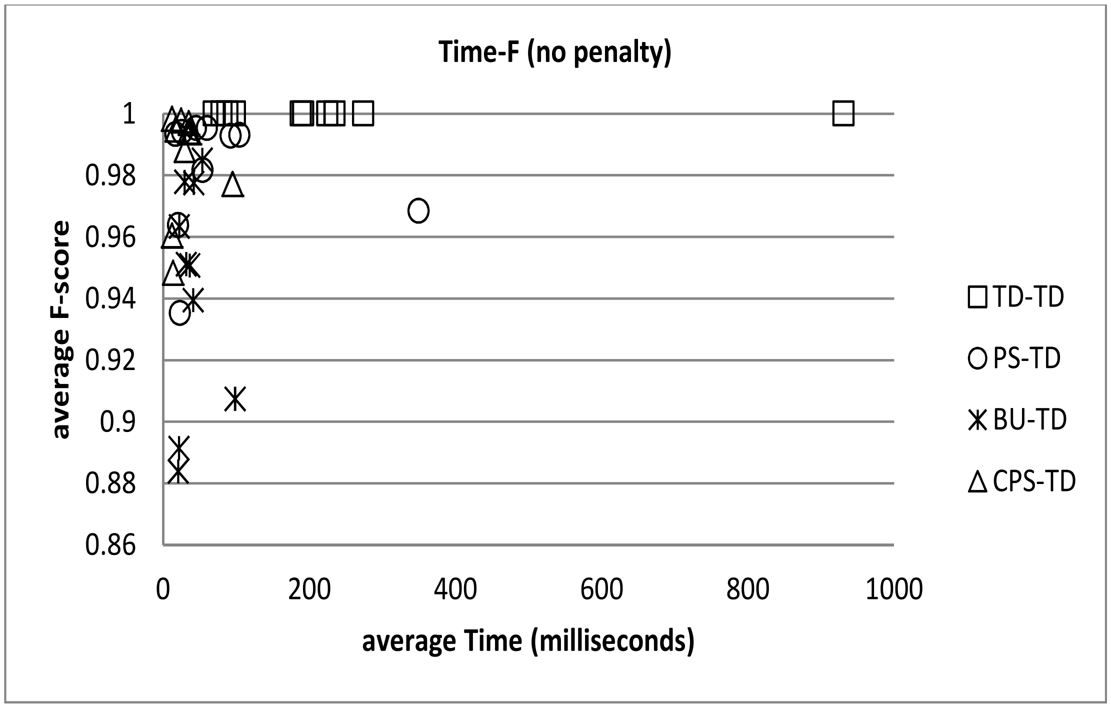


(a) CNN-Money

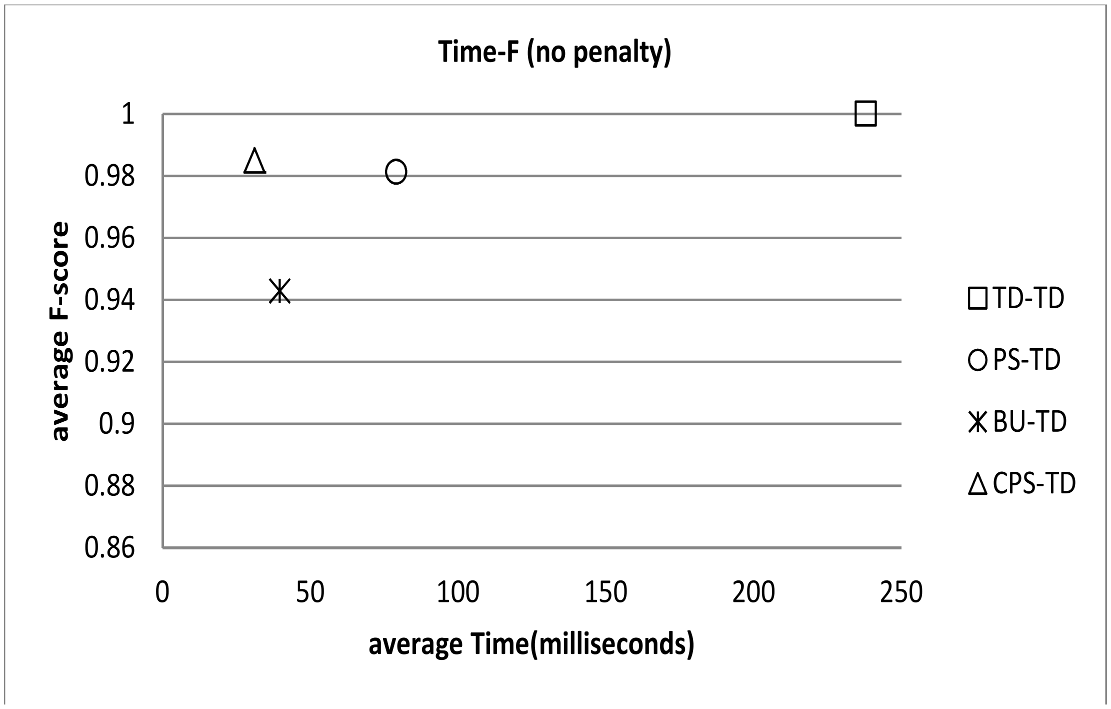


(b) CNET-Latest

Figure 5.9: Efficiency evaluation results-5

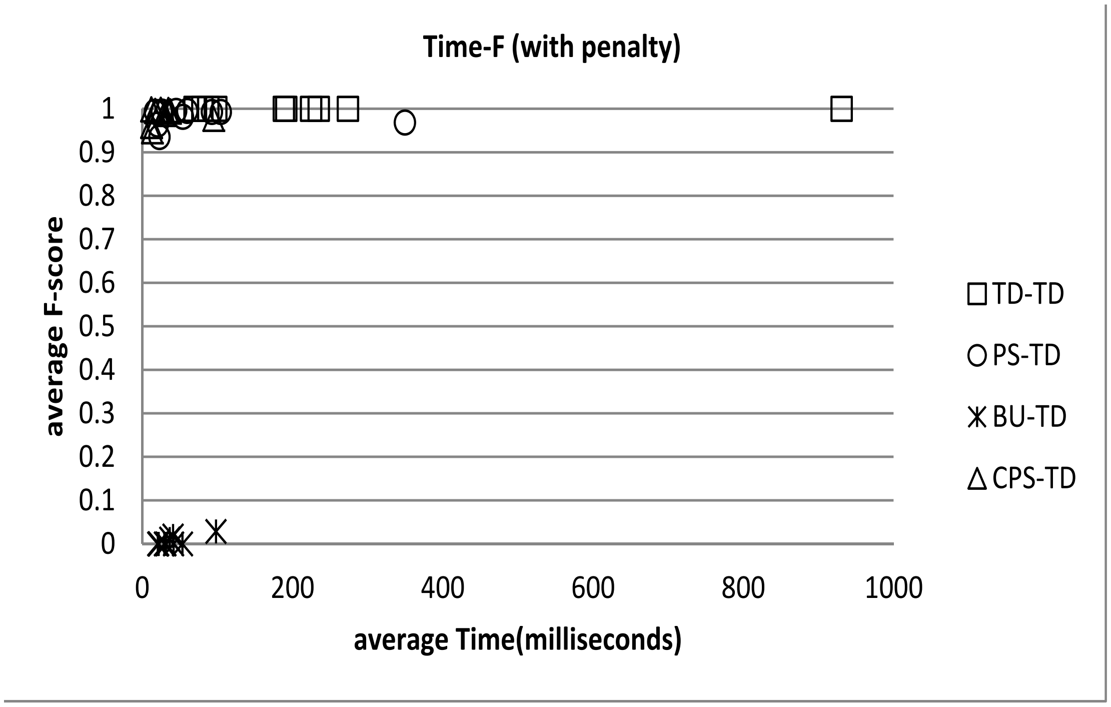


(a) Average Time vs average F-score for each test set

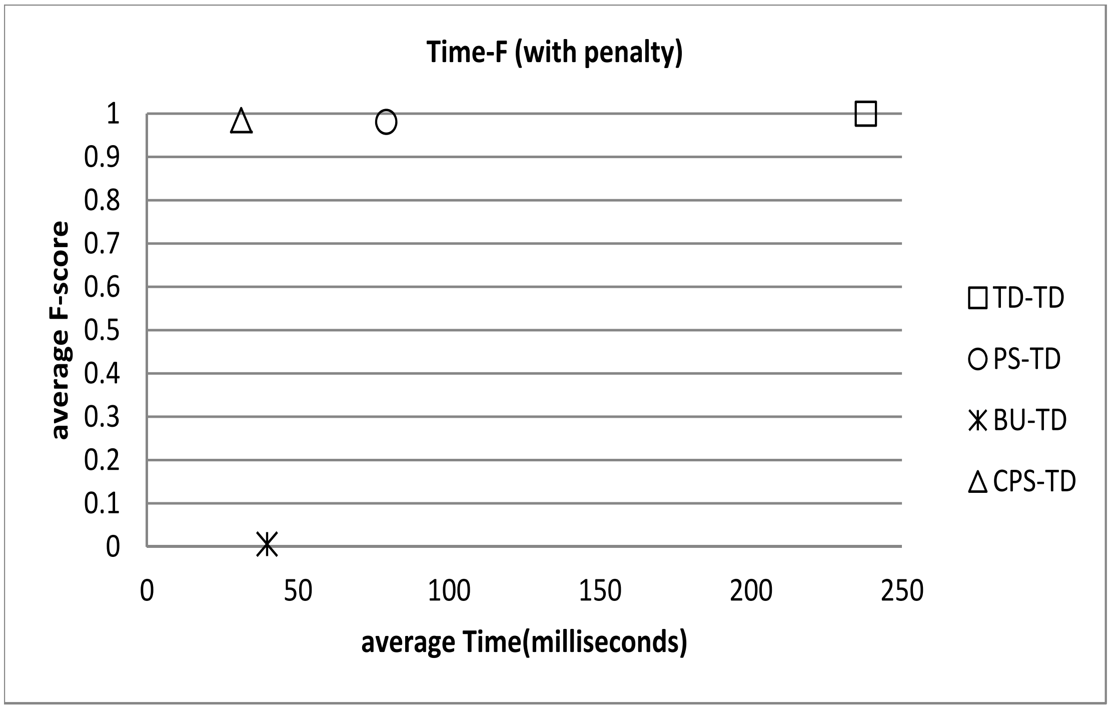


(b) Average Time vs average F-score for all test sets

Figure 5.10: Average Time vs F-score



(a) Average Time vs average F-score for each test set



(b) Average Time vs average F-score for all test sets

Figure 5.11: Average Time vs F-score

The six parameters were defined as following:

tree size = number of DOM tree nodes

leaf size = number of leaf nodes

total changes = number of non-template nodes

leaf changes = number of non-template nodes which are leaf nodes and parents are template nodes

$$\textit{total-change ratio} = \frac{\textit{total changes}}{\textit{tree size}}$$

$$\textit{leaf-change ratio} = \frac{\textit{leaf changes}}{\textit{total changes}}$$

Figure 5.12 - 5.15 shows correlations between the above six parameters and the running time of a specific method. For a given method and a given data set, we computed Pearson's correlation coefficients between six parameters and the running time.

From Figure 5.12 - 5.15, we noticed that "tree size" and "leaf size" showed very similar behaviors in terms of their impacts on running time of TD-TD,PS-TD,BU-TD and CPS-TD. For all methods and all data sets, "tree size" and "leaf size" were positively related to the running time. For a given data set and a given method, the correlation coefficient between "tree size" and the runnng time was almost the same as the correlation coefficient between "leaf size" and the running time. Thus, we could consider only one of them as a potential parameter that may determine the running time. Similar to "tree size" and "leaf size", "total-change ratio" and "total changes" had similar behaviors which are shown in Figure 5.12 - 5.15. Similarly, only one of them could be considered as a potential parameter that may determine the running time. In addition, we observed that, for a given method, correlations between "leaf-change ratio" and the running time were sometimes opposite to correlations between the running time and "leaf changes". For example, as shown in Figure 5.12, the Pearson's correlation coefficients between the running time of TD-TD and "leaf change" were all positive while there were some negative correlation coefficients between the running time of TD-TD and "leaf-change ratio". Thus, "leaf-change ratio" and "leaf changes" may have different impacts on the running time. So, the running time behaviors of all four approaches could be related to the following four parameters: *tree size*,*total changes*,*leaf changes* and *leaf-change ratio*.

Though we know the running time behaviors of TD-TD,BU-TD,PS-TD and CPS-TD could be related to above four parameters, the problem of finding which parameters are more important to a specific approach still needs to be furthur studied.

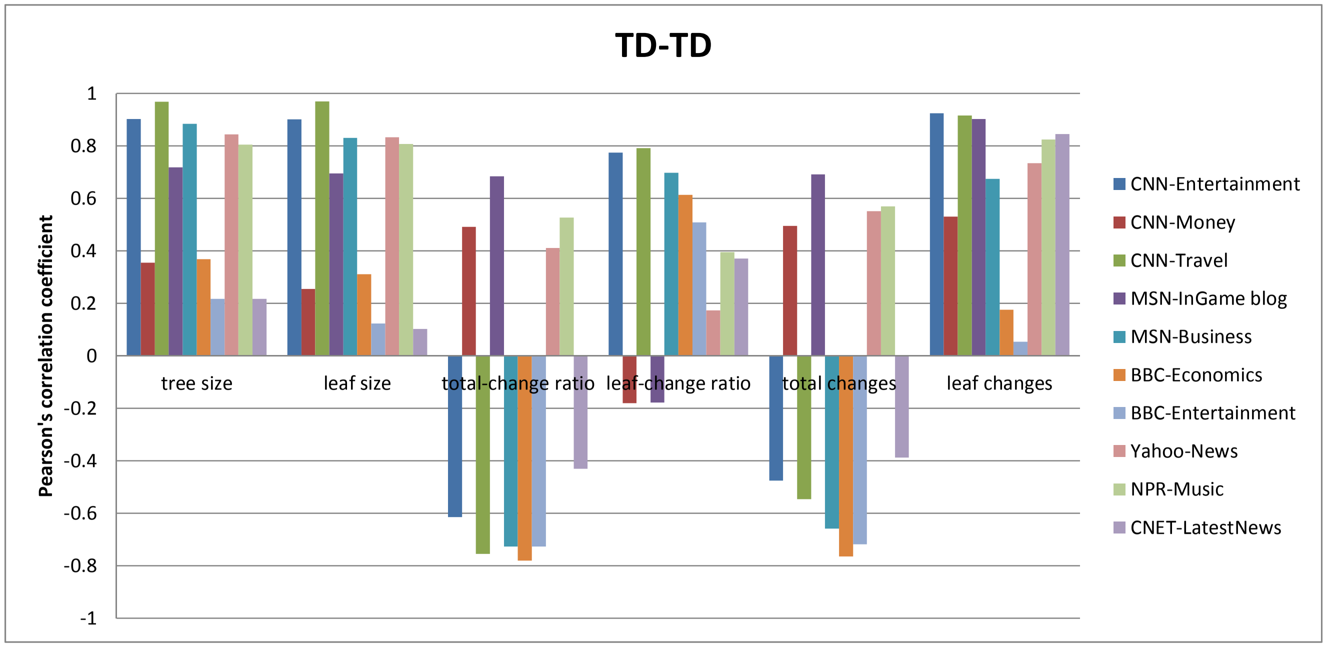


Figure 5.12: Correlations between running time of TD-TD and tree size,leaf size,total-change ratio,leaf-change ratio,total changes and leaf changes

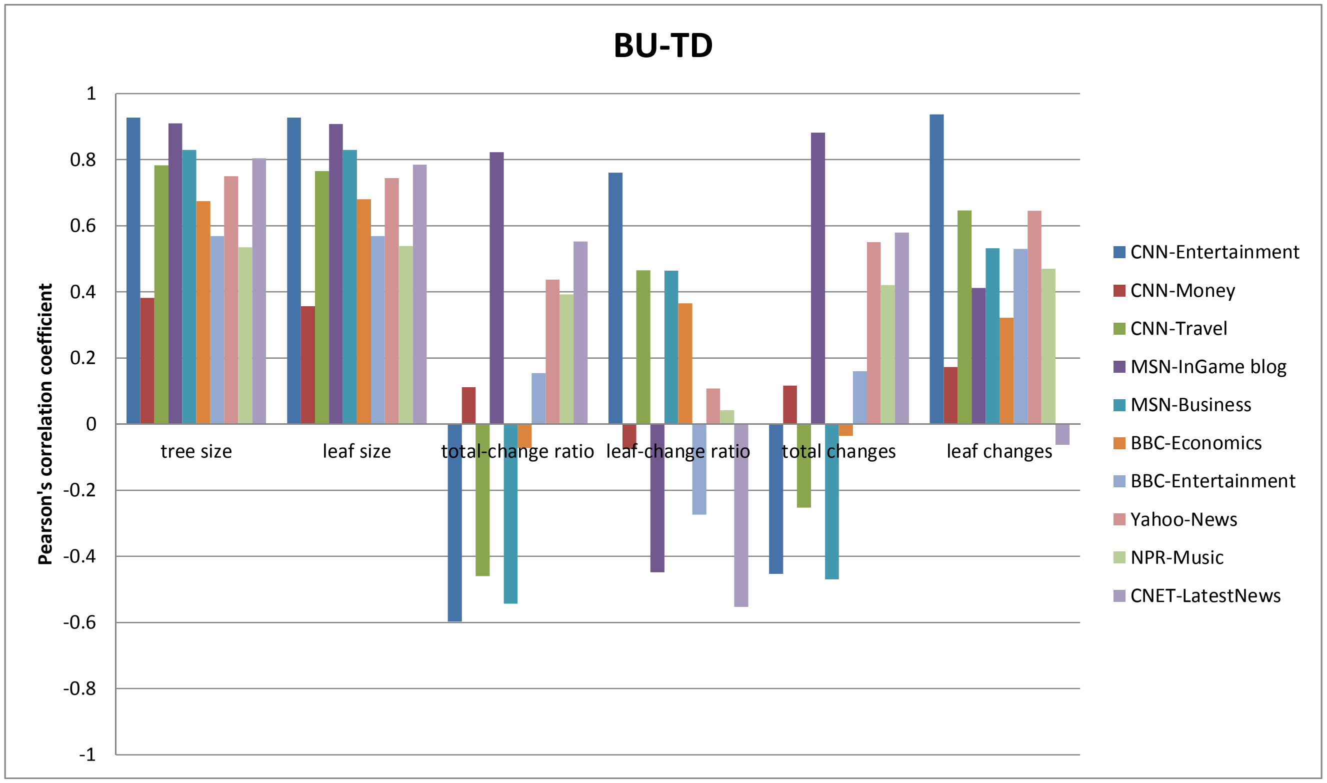


Figure 5.13: Correlations between running time of BU-TD and tree size,leaf size,total-change ratio,leaf-change ratio,total changes and leaf changes

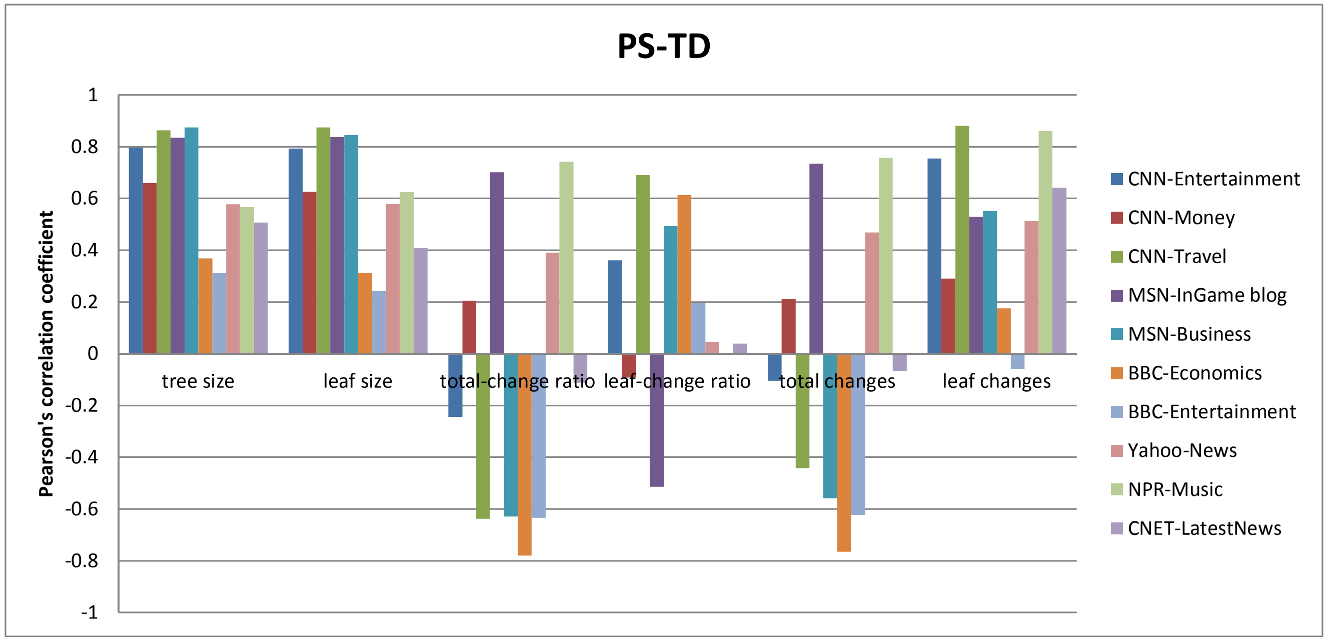


Figure 5.14: Correlations between running time of PS-TD and tree size,leaf size,total-change ratio,leaf-change ratio,total changes and leaf changes

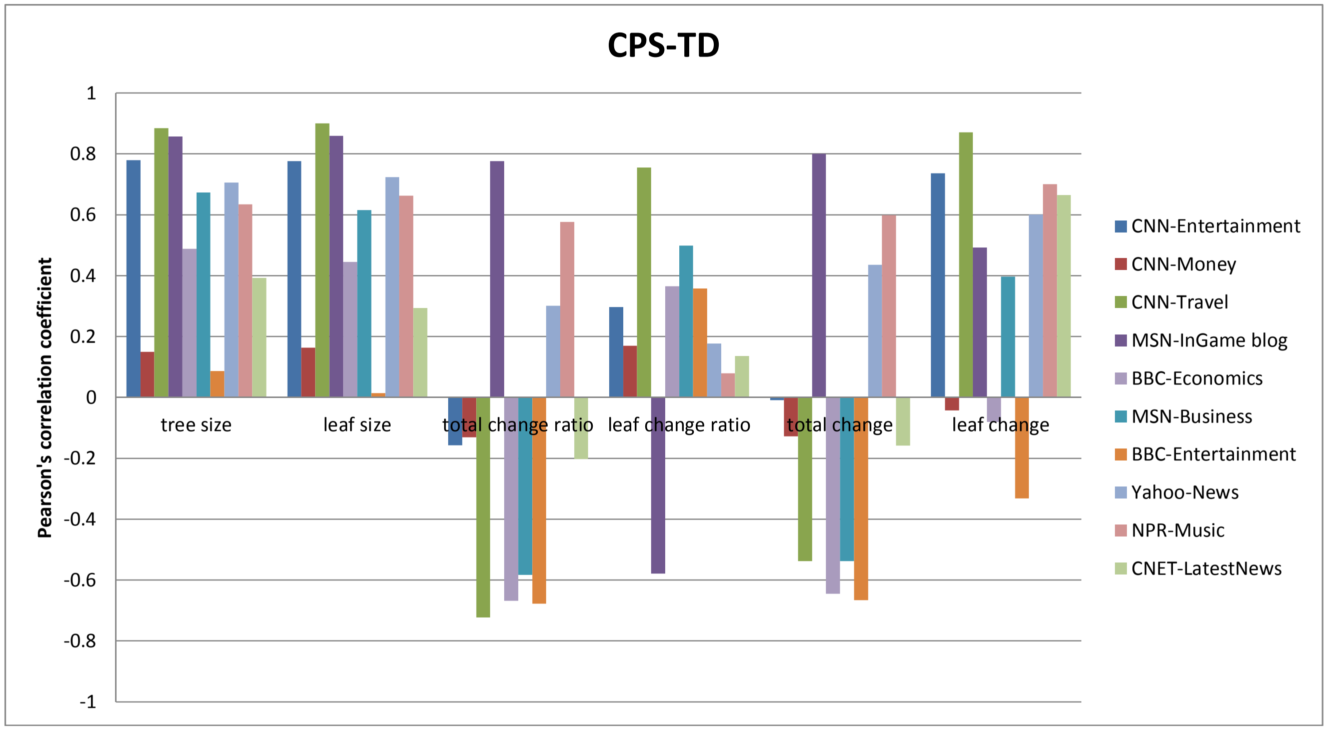


Figure 5.15: Correlations between running time of CPS-TD and tree size,leaf size,total-change ratio,leaf-change ratio,total changes and leaf changes

Chapter 6

CONCLUSION

In this thesis, we presented new approaches to the problem of detecting templates of web pages. Different from existing tree mapping algorithms, our algorithms first transform trees to unique sequential representations apply longest common subsequence matching algorithm on them. Then the template is obtained by finding a sequence of a subtree which is common to all trees. Experimental evaluations showed that our sequence-based template detection algorithms returned results which are almost identical to optimal results in most cases. It is showed that sequence-based algorithms are faster than existing tree mapping algorithms. Limitations of sequence-based algorithms are also discussed in this thesis.

REFERENCES

[1] David Gibson, Kunal Punera, and Andrew Tomkins, “The volume and evolution of web page templates,” in *WWW ’05: Special interest tracks and posters of the 14th international conference on World Wide Web*, New York, NY, USA, 2005, pp. 830–839, ACM.

[2] Karane Vieira, Altigran S. da Silva, Nick Pinto, Edleno S. de Moura, ao M. B. Cavalcanti, Jo and Juliana Freire, “A fast and robust method for web page template detection and removal,” in *CIKM ’06: Proceedings of the 15th ACM international conference on Information and knowledge management*, New York, NY, USA, 2006, pp. 258–267, ACM.

[3] Deepayan Chakrabarti, Ravi Kumar, and Kunal Punera, “Page-level template detection via isotonic smoothing,” in *WWW ’07: Proceedings of the 16th international conference on World Wide Web*, New York, NY, USA, 2007, pp. 61–70, ACM.

[4] Ziv Bar-Yossef and Sridhar Rajagopalan, “Template detection via data mining and its applications,” in *WWW ’02: Proceedings of the 11th international conference on World Wide Web*, New York, NY, USA, 2002, pp. 580–591, ACM.

[5] Karane Vieira, André Luiz Costa Carvalho, Klessius Berlt, Edleno S. Moura, Altigran S. Silva, and Juliana Freire, “On finding templates on web collections,” *World Wide Web*, vol. 12, no. 2, pp. 171–211, 2009.

[6] Dennis Fetterly, Mark Manasse, Marc Najork, and Janet Wiener, “A large-scale study of the evolution of web pages,” in *Proceedings of the 12th international conference on World Wide Web*, New York, NY, USA, 2003, WWW ’03, pp. 669–678, ACM.

[7] Eytan Adar, Jaime Teevan, and Susan T. Dumais, “Resonance on the web: web dynamics and revisitation patterns,” in *Proceedings of the 27th international conference on Human factors in computing systems*, New York, NY, USA, 2009, CHI ’09, pp. 1381–1390, ACM.

[8] Shian-Hua Lin and Jan-Ming Ho, “Discovering informative content blocks from web documents,” in *KDD ’02: Proceedings of the eighth ACM SIGKDD international conference on Knowledge discovery and data mining*, New York, NY, USA, 2002, pp. 588–593, ACM.

[9] Lan Yi, Bing Liu, and Xiaoli Li, “Eliminating noisy information in web pages for data mining,” in *KDD ’03: Proceedings of the ninth ACM SIGKDD international conference on Knowledge discovery and data mining*, New York, NY, USA, 2003, pp. 296–305, ACM.

[10] Ruihua Song, Haifeng Liu, Ji-Rong Wen, and Wei-Ying Ma, “Learning block importance models for web pages,” in *WWW ’04: Proceedings of the 13th international conference on World Wide Web*, New York, NY, USA, 2004, pp. 203–211, ACM.

[11] Hung-Yu Kao, Jan-Ming Ho, and Ming-Syan Chen, “Wisdom: Web intrapage informative structure mining based on document object model,” *IEEE Trans. on Knowl. and Data Eng.*, vol. 17, no. 5, pp. 614–627, 2005.

[12] Sandip Debnath, Prasenjit Mitra, Nirmal Pal, and C. Lee Giles, “Automatic identification of informative sections of web pages,” *IEEE Trans. on Knowl. and Data Eng.*, vol. 17, no. 9, pp. 1233–1246, 2005.

[13] W3C, “Document object model (dom) level 2 html specification,” http://www.w3.org/TR/2003/REC-DOM-Level-2-HTML-20030109.

[14] Sudarshan S. Chawathe, “Comparing hierarchical data in external memory,” in *Proceedings of the 25th International Conference on Very Large Data Bases*, San Francisco, CA, USA, 1999, VLDB ’99, pp. 90–101, Morgan Kaufmann Publishers Inc.

[15] Davi de Castro Reis, Paulo Braz Golgher, Altigran Soares da Silva, and Alberto H. F. Laender, “Automatic web news extraction using tree edit distance,” in *WWW*, 2004, pp. 502–511.

[16] Wuu Yang, “Identifying syntactic differences between two programs,” *Software - Practice and Experience*, vol. 21, pp. 739–755, 1991.

[17] Gabriel Valiente, “An efficient bottom-up distance between trees,” in *SPIRE*, 2001, pp. 212–219.

[18] Paul F. Dietz, “Maintaining order in a linked list,” in *Proceedings of the fourteenth annual ACM symposium on Theory of computing*, New York, NY, USA, 1982, STOC ’82, pp. 122–127, ACM.

[19] Quanzhong Li and Bongki Moon, “Indexing and querying xml data for regular path expressions,” in *Proceedings of the 27th International Conference on Very Large Data Bases*, San Francisco, CA, USA, 2001, VLDB ’01, pp. 361–370, Morgan Kaufmann Publishers Inc.

[20] Masatoshi Yoshikawa, Toshiyuki Amagasa, Takeyuki Shimura, and Shunsuke Uemura, “Xrel: a path-based approach to storage and retrieval of xml documents using relational databases,” *ACM Trans. Internet Technol.*, vol. 1, pp. 110–141, August 2001.

[21] Chun Zhang, Jeffrey Naughton, David DeWitt, Qiong Luo, and Guy Lohman, “On supporting containment queries in relational database management systems,” in *Proceedings of the 2001 ACM SIGMOD international conference on Management of data*, New York, NY, USA, 2001, SIGMOD ’01, pp. 425–436, ACM.

[22] Praveen Rao and Bongki Moon, “Prix: Indexing and querying xml using prüfer sequences,” in *ICDE ’04: Proceedings of the 20th International Conference on Data Engineering*, Washington, DC, USA, 2004, p. 288, IEEE Computer Society.

[23] Shirish Tatikonda, Srinivasan Parthasarathy, and Matthew Goyder, “Lcs-trim: dynamic programming meets xml indexing and querying,” in *VLDB ’07: Proceedings of the 33rd international conference on Very large data bases*. 2007, pp. 63–74, VLDB Endowment.

[24] Robert A. Wagner and Michael J. Fischer, “The string-to-string correction problem,” *J. ACM*, vol. 21, pp. 168–173, January 1974.

[25] William J. Masek and Mike Paterson, “A faster algorithm computing string edit distances,” *J. Comput. Syst. Sci.*, vol. 20, no. 1, pp. 18–31, 1980.

[26] Eugene W. Myers, “An o(nd) difference algorithm and its variations,” *Algorithmica*, vol. 1, pp. 251–266, 1986.

[27] Esko Ukkonen, “Algorithms for approximate string matching,” *Inf. Control*, vol. 64, pp. 100–118, March 1985.

[28] S. Wu, U. Manber, G. Myers, and W. Miller, “An o(np) sequence comparison algorithm,” *Inf. Process. Lett.*, vol. 35, pp. 317–323, September 1990.

[29] Daniel S. Hirschberg, “Algorithms for the longest common subsequence problem,” *J. ACM*, vol. 24, pp. 664–675, October 1977.

[30] James W. Hunt and Thomas G. Szymanski, “A fast algorithm for computing longest common subsequences,” *Commun. ACM*, vol. 20, pp. 350–353, May 1977.

[31] Claus Rick, “New algorithms for the longest common subsequence problem,” Tech. Rep., 1994.

[32] Alberto Apostolico and Concettina Guerra, “The longest common subsequence problem revisited,” *Algorithmica*, vol. 2, pp. 316–336, 1987.