XML Tree Finder System: a First Step towards XML Data Mining Final Report

Anguo Dong
Supervisor: Dr.Reda Alhajj
Computer Science Department
University of Calgary

April 5, 2004

Abstract

The problem of searching frequent trees from a collection of tree-structured XML data modeling is considered. The aim of this XML Tree Finder system(XTFS) is to find the tree whose exact or perturbed copies are frequent in a collection of the labeled trees. The definition of the labeled tree will be given later.Frequent here means that the tree we find is the Maximal Common Tree of the collection of the labeled tree.

1 Introduction

An XML Tree Finder System(XTFS) is introduced in this paper. This system takes a group of XML documents as input, clusters them into different clusters and find the maximal common tree of each cluster. Some idea behind the Alexandre TreeFinder System are also used in this system. These will be introduced in later sections.

As the Alexandre TreeFinder, the main motivating application of this work is the construction of a tree-based mediated schema for integrating multiple and heterogeneous sources of XML data. The construction can be considered as extraction of useful knowledge and then reorganizing or rebuilding them. This task is important to the actual mining system or application so that we call it the basic step or the first step to XML data mining.

The paper is organized as follows. In the *Formal Backgroud* section, we introduce some important knowledge and terminology used in this paper. In section 3 *Motivation Example*, one specific XML tree structure is illustrated and is used to give the motivated impression about the frequent tree. We also give the definition of *Maximal Common Tree* here. The overview of the System is gone through in section 4 *Overview of the TreeFinder System*. Finally, it is the section 5 *Conclusion and Future Work* which will summarize the whole system and illustrate the future work.

2 Formal Background

2.1 Introduction of Data Mining

Data mining is motivated by the decision support problem faced by most large retail organizations. [3] Bar-code technology made it possible for retail organizations to collect and store massive amounts of sales data, referred to as the basket data. Successful organizations view such databases as important pieces of the marketing infrastructure Organizations are interested in instituting information-driven marketing processes, managed by database technology, that enable marketers to develop and implement customized marketing programs and strategies. [5]

Data mining includes several steps: problem analysis, data extraction, data cleansing, rules develop-

ment, output analysis and review. [8] Data mining sources are typically flat files extracted from on-line sets of files, from data warehouses or other data source. Data may however be derived from almost any source. Whatever the source of data, data mining will often be an iterative process involving these steps.

Data mining tools offer great potential for corporate data warehouses since they *discover* rather than *confirm* trends or patterns in data. [11] Most of these symbolic classifiers are also known as rule-induction programs or decision tree generators. They use statistical algorithms or machine learning algorithms such as ID3, C4.5, AC2, CART, CHAID, CN2, or modifications of these algorithms. [12]

2.2 XML and Modeling XML Data Structure [20]

In a very short space of time, XML has become a hugely popular format for marking up all kinds of data, from web content to data used by applications [9]. It is finding its way across all tiers of development: storage, transport, exchange, and display. So There are rich and compelling reasons for data mining or the knowledge discovery system to keep up with the pace of XML development. [10]

XML can also be used to store data in files or in databases [14]. Applications can be written to store and retrieve information from the store, and generic applications can be used to display the data. XML can be generated from a database without any installed XML software. [22] The XML response from the previous example can easily be modified to fetch its data from a database. [6]

Suppose we have a database called *databse.mdb* which has a table named *GuestBook*. This table contains two fields $fname$ and $lname$. To generate an XML database response from the server, we can simply write the following code and save it as an ASP file: [1]

```
<%
response.ContentType = "text/xml"

set conn=Server.CreateObject("ADODB. Connection")
  conn.provider="Microsoft.Jet.OLEDB.4.0;"
  conn.open
  server.mappath("adodatabase.mdb")
  sql="select fname,lname from GuestBook"
  set rs=Conn.Execute(sql)

  rs.MoveFirst()

  response.write("<?xml version='1.0' encoding='ISO-8859-1'?>")
  response.write("<guestbook>")

  while (not rs.EOF)
    response.write("<guest>")
    response.write("<fname>" & rs ("fname") & "<fname>")
    response.write("<lname>" & rs("lname") & "<lname>")
    response.write("<guest>")
    rs.MoveNext()
  wend

  rs.close()
  conn.close()
  response.write("<guestbook>")
  %>
```

In this system, XML data structure(e.g., a DTD) is modelled as labelled tree [20]. While doing this, we will ignore the ID-references and hyper links which are less useful for the model. In addition, the elements and attributes are not distinguished as well since they can be actually considered to be the same data element in the labelled tree [17]. Another important assumption of this data structure is that there exists a one-to-one correspondence between all labels and real world concepts.

2.3 Parsing XML File and Data Binding [21]

The main point behind data binding is to generate a correspondence or interface between these XML

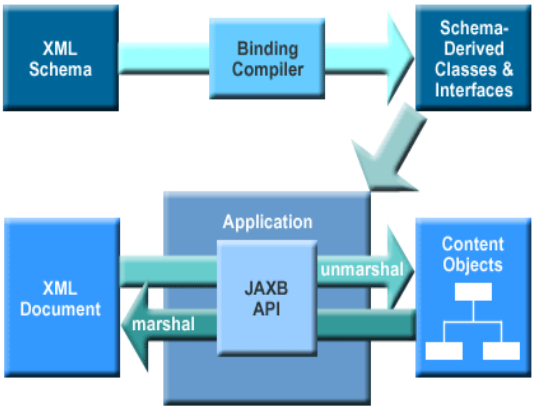

Figure 1: Data Binding Shcema

schemas and Java classes and then exploit this mapping when converting XML documents to and from Java objects. Our goal is to process XML using Java code. The data binding schema can be illustrated in Figure 1.

Among lots of methods of data binding and processing XML schemas, two of the most popular approaches are the Simple API for XML Parsing (SAX) and the Document Object Model (DOM). We can access SAX and DOM through the Java API for XML Parsing (JAXP) that is a part of the Java 1.4 release.

SAX is an event-based approach. As the parser works its way through the XML document, you can have it notify you of certain events.

Alternatively, DOM is a tree-based approach. The result of parsing an XML file with a DOM-based parser is a document object that contains a structured representation of the file. DOM is paid more attention to illustrate since we use it in this system. Navigating the DOM usually involves code that looks like conferenceNodeList.item(i).getFirstChild().getFirstChild().getData(). Here's a DOM tree view of the document:

<?xml version="1.0" encoding="UTF-8" ?>
<Conference>
<ConfName>Data Mining</ConfName>
$< ConfDate > Mar.18th, 2004 <$

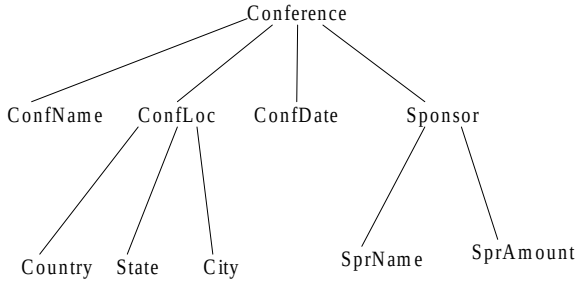

Figure 2: DOM Tree of the Conference XML Shcema

$/ConfDate >$
$< ConfLoc >$
$< Country > Canada < /Country >$
$< State > Alberta < /State >$
$< City > Calgary < /City >$
$< /ConfLoc >$
$< sponsor >$
$< SprName > IASTED < /SprName >$
$< SprAmount > \$158,888 < /SprAmount >$
$< /sponsor >$
$< /Conference >$

We can parse the above XML schema to DOM tree as shown in Figure 2.

2.4 Computing Similarity between XML Documents

There are some methods to compute similarity between two sets of extended-element vectors representing two XML documents. In Preparation for Semantics-Based XML, the author introduce one methodology for computing the similarity by taking account of XML semantics to prepare XML document for XML mining. Mining The computing is divided into three steps which are Generating extended-element vectors, Measure of element similarity, and Constructing of the similarity matrix. [17] Generating the extended-element vectors for an XML document is as follows. [16]

- Parse an XML document to extract elements and generate a DOM(Document Object Model) tree.

- Sift meaningful tokens by filtering delimiters such as space, hyphen, and underscore.
- Delete tokens included in a stop-list.
- Extract stems or original form of the tokens through stemming process.
- Extend elements thus found, using the WordNet thesaurus and a User-defined word library, with synonyms, compound words, and abbreviations.

The basis of the measures is the degree of match between original elements, between an original element and a term in the extended-element vector of another element [24]. The levels are divided into six with Level 0 is the least similar and Level 6 is the most similar.

Finally, the similarity matrix for two set of extended-element vectors representing two XML documents is constructed. One set of extended-element vectors forms the column, and another the row of the matrix. [17]

Based on the above methodology, we ignore the process to compute the similarity and suppose that all the nodes of the labeled tree with the minimal similarity are the same. For example, we consume that "$ConferenceName$", "$ConfName$", and "$MeetingName$" have the same meaning for the name of the $conference$, and we can use only one label to represent them. For instance, we use "$ConfName$" to stand for all these three terms.

2.5 θ-subsumption [7]

θ-subsumption is used to define the ordering of the program clause or the tree nodes. [23] It is the basic term for us to define the Inclusion by Tree Subsumption which is one of the most important term in the TreeFinder system. θ-subsumption is defined below.

Definition (θ-Subsumption)A substitution $\theta = \{V_1/t_1, ..., V_n/t_n\}$ is an assignment of terms t_i to variables V_i. Applying a substitution θ to a term, atom, or clause F yields the instantiated term, atom, or clause Fθ where all occurrences of the variables V_i are simultaneously replaced by the term t_i. Let c and c' be two program clauses. Clause c θ-subsumes c' if there exists a substitution θ, such that c$\theta \subseteq c'$ [19].

To illustrate the above notions, consider the clause c = daughter(X, Y)← parent(Y,X). Applying the substitution $\theta = \{X/mary, Y/ann\}$ to clause c yields
cθ = daughter(mary,ann)← parent(ann, mary).

Clauses can be viewed as sets of literals: the clausal notation daughter(X, Y)← parent(Y, X) thus stands for {daughter(X, Y), ¬parent(Y, X)} where all variables are assumed to be universally quantified, and the commas denote disjunction. According to the definition, clause c θ-subsumes c' if there is a substitution θ that can be applied to c such that every literal in the resulting clause occurs in c'.

θ-subsumption introduces a syntactic notion of generality. [7] Clause c is at least as general as clause c'(c $\leq c'$) if c θ- subsumes c'. Clause c is more general than c' (c $< c'$) if (c $\leq c'$) holds and ($c' \leq$ c) does not. In this case, we say that c' is a specialization of c and c is a generalization of c'. If the clause c' is a specialization of c then c' is also called a refinement of c.

There are two important properties of θ-subsumption: [4]

- If c θ-subsumes c' then c logically entails c', c $\models c'$.
- The relation $\leq$ introduces a lattice on the set of reduced clauses [19]. This means that any two reduced clauses have a least upper bound (lub) and a greatest lower bound (glb). Both the lub and the glb are unique up to equivalence (renaming of variables) under θ-subsumption. Reduced clauses are the minimal representatives of the equivalence classes of clauses defined by θ-subsumption. For example, the clauses daughter(X, Y)← parent(Y, X), parent(W, V) and daughter(X, Y)← parent(Y, X) θ-subsume one another and are thus equivalent. The latter is reduced, while the former is not.

The second property of θ-subsumption leads to

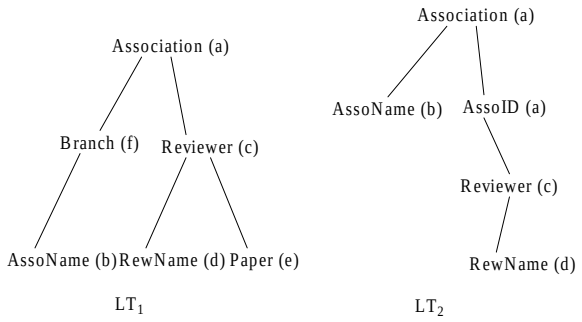

Figure 3: Labeled Tree

the following definition: The least general generalization (lgg) of two clauses c and c', denoted by lgg(c; c'), is the least upper bound of c and c' in the θ-subsumption lattice [19]. The *LGG* will be introduced in more detail in later section. Note that θ-subsumption and least general generalization are purely syntactic notions since they do not take into account any background knowledge [18].

2.6 Tree inclusion and the relationship with θ-subsumption [7]

Definition (Labelled Tree) A *labelled tree* is a pair ⟨t,label⟩ where (i)t is a finite tree whose nodes are in $\mathcal{N}$. (ii)label is a *labelling function* that assigns a label to each node in it. [20]

Figure 3 gives some examples of labeled trees. The labels are the key point here since they play the role of the begin-end tag in XML data models. In this project, the labelled tree model for XML data structure will be used too. From there, the efficient and scalable Tree Finder System will be constructed.

Definition (Inclusion by tree subsumption) [20] let t and t' be two labeled trees. We say that t is *included according to tree subsumption* (or included when no confusion is possible) in t' if there exists a mapping f from the nodes of t into the set of nodes of t' such that f preserve the ancestor relation:

$\forall\ u$ in t, label(u) = label$(f(u))$ and

$\forall u, v$ in t, $anc(u, v) \longrightarrow anc(f(u), f(v))$.

The advantage of the above definition is the following: if we choose to represent labeled trees as relational formulas, then tree subsumption is equivalent to the θ-suubsumption relation defined by ([7]).

Definition (Relational description of labeled trees) [20] Let t be a labeled tree. $Rel(t)$ is the conjunction of all atoms $ab(u, v)$, such that u and v are nodes in t, with $label(u) = a$, $label(v) = b$ and u is the parent node of v. $Rel^+(t)$ is the conjunction of atoms $a * b(u, v)$, such that u and v are nodes in t, with $label(u) = a$, $label(v) = b$ and u is the ancestor node of v.

Figure 4 illustrates the two encoding function *Rel* and Rel^+ for two labeled trees LT_1 and LT_2. Using the encoding function, we will get the final tree as shown in Figure 5 going through the XML Tree Finder System.

2.7 Method of Discovering Large Item Sets-*AprioriAlgorithm*

2.7.1 Formal Definition of Data Mining Association Rules

Algorithms for discovering large item sets make multiple passes over the data. In the first pass, we count the support of individual items and determine which of them are large (with minimum support). In each subsequent pass, we start with a seed set of item sets found to be large in the previous pass, then use this seed set for generating new potentially large item sets, called candidate item sets, and count the actual support for these candidate item sets during the pass over the data. At the end of the pass, we determine which of the candidate item sets are actually large, and they become the seed for the next pass. This process continues until no new large item sets are found. [15].

Definition(Transaction) Let I = $\{i_1, i_2, ..., i_m\}$ be a set of literals, called items. Let D be a set of transactions, where each transaction T in D is a set

$Rel(LT_1)$	$Rel^+(LT_1)$
af(Association, Branch)	a*f(Association, Branch)
ac(Association, Reviewer)	a*c(Association, Reviewer)
fb(Branch, AssoName)	a*b(Association, AssoName)
cd(Reviewer, RewName)	a*d(Association, RewName)
ce(Reviewer, Paper)	a*e(Association, Paper)
	f*b(Branch, AssoName)
	c*d(Reviewer, RewName)
	c*e(Reviewer, Paper)

$Rel(LT_2)$	$Rel^+(LT_2)$
ab(Association, AssoName)	a*b(Association, AssoName)
aa(Association, AssoID)	a*a(Association, AssoID)
ac(AssoID, Reviewer)	a*c(Association, Reviewer)
cd(Reviewer, RewName)	a*d(Association, RewName)
	a*c(AssoID, Reviewer)
	a*d(AssoID, RewName)
	c*d(Reviewer, RewName)

Figure 4: LT_1 and LT_2 relational encoding

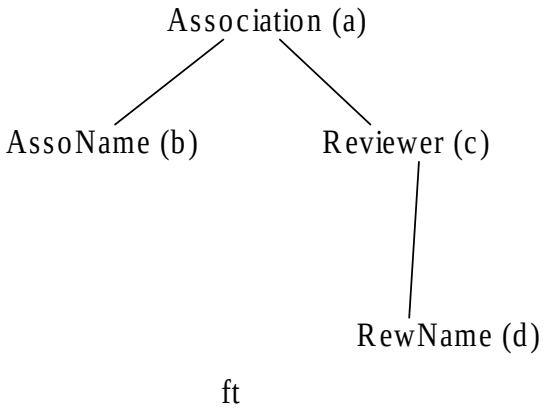

Figure 5: Final Tree

of items such that $T \subseteq$ I. [13]

We say a transaction T contains X, a set of some items in I, if X⊆T.

Definition(Association Rule) An association rule is an implication of the form X $\Longrightarrow$ Y, where X⊂I, Y⊂I, and X∩Y =∅. [13]

Definition(Support)X$\Longrightarrow$Y has support s in the transaction set D if s% of transactions in D contain X⇒Y. [13]

2.7.2 *AprioriAlgorithm* [2]

Some notation used in the *AprioriAlgorithm*:

K-itemset An itemset having k items.

L_k Set of large k-itemsets (those with minimum support). Each member of this set has two fields: i) itemset and ii) support count.

C_k Set of candidate k-itemsets (potentially large itemsets). Each member of this set has two fields: i) itemset and ii) support count.

The *AprioriAlgorithm*:

L_1 = {large 1-itemsets};

for (k = 2; $L_{k-1} \neq \emptyset$; k++) do begin

C_k = apriori-gen(L_{k-1}); // New candidates

forall transactions t ∈ D do begin

C_t = subset(C_k, t); Candidates contained in t

forall candidates c ∈ C_t do

c.count++;

end

$L_k = \{c \in C_k | c.count \geq minsup\}$

Answer = $\cup_k L_k$;

2.7.3 The *apriori − gen* function

The *apriori − gen* function takes as argument L_{k-1}, the set of all large (k-1)-itemsets. It returns a superset of the set of all large k-itemsets.

First, in the join step, we join L_{k-1} with L_{k-1}:

insert into C_k

select p.item$_1$, p.item$_2$,... , p.item$_{k-1}$, q.item$_{k-1}$

from L_{k-1}p, L_{k-1}q

where p.item$_1$ = q.item$_1$, ... ,p.item$_{k-2}$ = q.item$_{k-2}$, p.item$_{k-1}$ < q.item$_{k-1}$

Next, in the prune step, we delete all itemsets c ∈ C_k such that some (k-1)-subset of c is not in L_{k-1}:

forall itemsets c ∈ C_k do

forall (k-1)-subsets s of c do

if (s ∉ L_{k-1}) then

delete c from C_k;

2.7.4 *apriori − gen* Function Example

Let L_3 be {{1 2 3}, {1 2 4}, {1 3 4}, {1 3 5}, {2 3 4}}

1. In the join step:
   - {1 2 3} joins with {1 2 4} to produce {1 2 3 4}
   - {1 3 4} joins with {1 3 5} to produce {1 3 4 5}
   - After the join step, C_4 will be {{1 2 3 4}, {1 3 4 5}}

Database D

TID	Items
100	A C D
200	B C E
300	A B C E
400	B E

Figure 6: An example transaction database for data mining

2. In the prune step:
   - {1 2 3 4} is tested for existence of 3-item subsets within L_3, thus for {1 2 3}, {1 3 4}, and {2 3 4}
   - {1 3 4 5} is tested for {1 3 4}, {1 4 5}, and {3 4 5}, with {1 4 5} not found, and thus this set is deleted
3. We then will be left with only {1 2 3 4} in C_4

Consider an example transaction database given in Figure 6. In each iteration (or each pass), Apriori construct a candidate set of large itemsets, counts the number of occurences of each candidate itemsets, and then determines large itemsets based on a predetermined minimum support. In the first iteration, Apriori simply scans all the transactions to count the number of occurrences for each item. The set of candidate 1-itemsets, C_1, obtained is shown in Figure 7, Assuming that the minimum transaction support required is 2 (i.e., s=40%), the set of large 1-itemsets, L_1, composed of candidate 1-itemssets with the minimum support required, can then be determined. To discover the set of large 2-itemsets, in view of the fact that any subset of a large itmeset must also have minimum support, Apriori uses L_1*L_1 to generate a candidate set of itemsets C_2 where * is an operation for concatenation in this case. C_2 consists of

$|L_1|$ 2-itemsets. Next, the four transactions in D are scanned and the support of each candidate itemset in C_2 is counted. The middle table of the second row in Figure 7 represents the result from such counting in C_2. The set of large 2-itemsets, L_2, is therefore determined based on the support of each candidate 2-itemset in C_2.

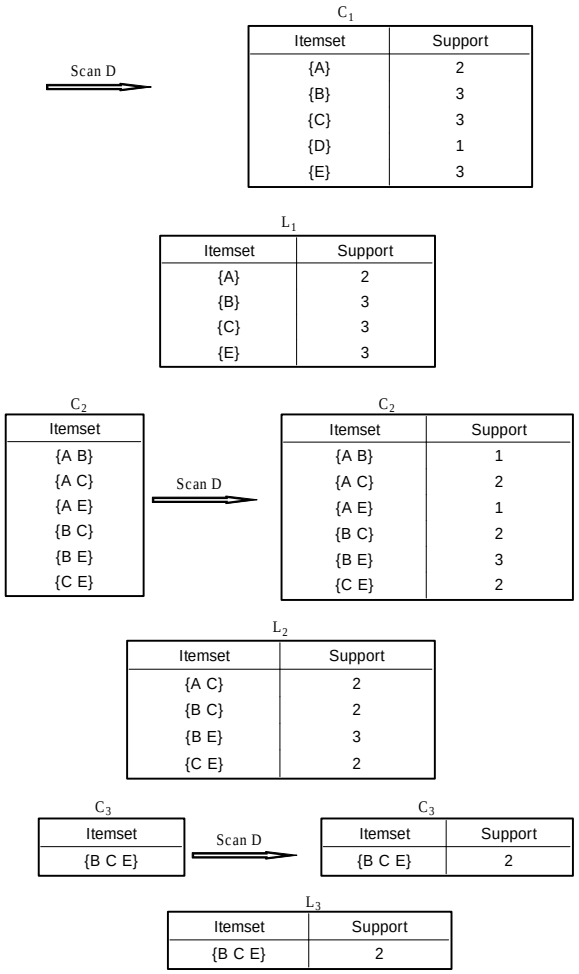

Figure 7: Generation of candidate itemsets and large itemsets

The set of candidate itemsets, C_3, is generated from L_2 as follows. From L_2, two large 2-itemsets with the same first item, such as {B C} and {B E}, are identified first. Then, Apriori tests whether the 2-itemset {C E}, which consists of their second items, constitutes a large 2-itemset or not. Since {C E} is a large itemset by itself, we know that all the subsets of {B C E} are large and then {B C E} becomes a candidate 3-itemset. There is no other candidate 3-itemset from L_2. Apriori then scans all the transactions and discovers the large 3-itmesets L_3 in Figure 7. Since there is no candidate 4-itemset to be constituted from L_3, Apriori ends the process of discovering large itemsets.

3 Motivating Example

We can use the tree structure in Figure 8 to illustrate the various XML database or XML documents. For instance, the root node $Conference$ in Figure7.A relates to the root element in the XML structures, $ConfID$ to the inner node, and $City$ to the most inner node.

Although the structures of the trees in Figure 8 vary so much, we still can group them into two groups since the tree t_1 in Figure 9 is commonly included by tree A, B, and C in Figure 8 and the tree t_2 in Figure 9 is commonly included by tree D, and E. We call t_1 and t_2 frequent trees corresponding to the input trees of Figure 8.

The common tree for tree A, B, and C is tree t_1 in Figure 9. The common tree for tree C, and D is tree t_2 in Figure 9.

Definition (Maximal common Tree) Let $t, t_1, t_2, ..., t_n$ be labelled trees. We say that t is a *maximal common tree* of $t, t_1, t_2, ..., t_n$ iff:

- $\forall i \in$ [1..n] t is included in t_i
- t is maximal for the previous property, $i.e$ if there is a labelled tree t' such as t *is included in* t' and $\forall i \in$ [1..n] t' *is included in* t_i then t' is identical to t.

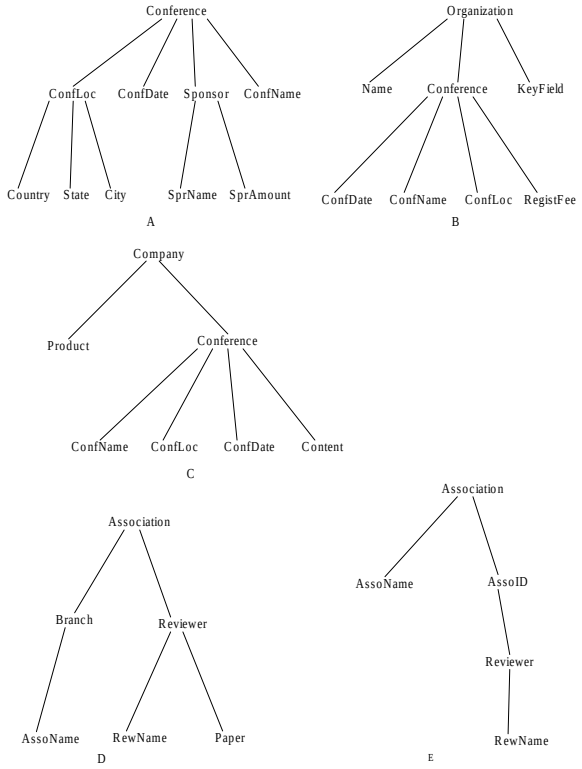

Figure 8: Various Tree Structure.

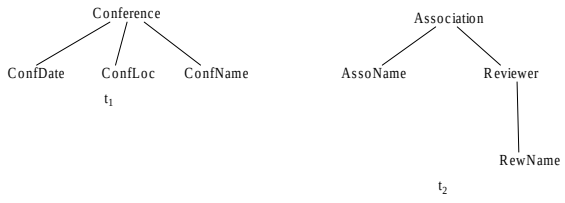

Figure 9: Frequent Tree

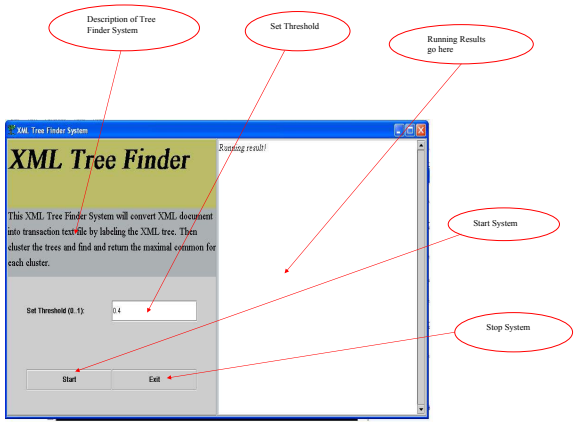

Figure 10: Interface of the Tree Finder System

For instance, the tree t_1 in Figure 9 is the maximal common tree of A, B, and C in Figure 8, and t_2 is the maximal common tree of D and E in Figure 8.

4 Overview of the XML Tree Finder System

4.1 Interface Illustration of the XML Tree Finder System

A friendly interface of this XML Tree Finder System can be viewed in Figure 10. As shown, there is one simple description of the system, and one text area to show the sequent running results. Two buttons *Start* and *Exit* can be used to start and exit the system. As for the threshold text field, user can specify the threshold between 0 and 1 here.

4.2 Parse XML schema into DOM Tree

The first step of this XML Tree Finder System will parse each XML schema of a group of XML documents into DOM trees. Therefore, it traverses the DOM tree and generates the abstraction of it T = $\{t_1, ..., t_n\}$, where each t_i is viewed as a transaction

item made of all the items $l * m$ such that l is the label of an ancestor of a node labeled by m in t_i. Each item $l * m$ has a unique identifiers. We call this abstraction of the DOM tree as $XML\ node-based$ input transaction corresponding to the XML schema from which it is generated.

For instance, the $XMl\ node-based$ input transaction corresponding to tree A of Figure 8 is:

A= $\{Conference * ConfName,\ Conference * ConfLoc,\ Conference * ConfDate,\ Conference * Sponser,\ Conference * Country,\ Conference * City,\ Conference * State,\ Conference * SprName,\ Conference * SprAmount,\ ConfLoc * Country,\ ConfLoc * State,\ ConfLoc * City,\ Sponser * SprName,\ Sponser * SprAmount\}$

This splitting makes possible the use of a standard frequent item sets algorithm for discovering frequent label pairs in the input trees. This $XML\ node-based$ input transaction will be also used to generate the transaction item pool defined later.

4.3 Generate the Transaction Item Pool

Definition (Item Pool) An *ItemPool* is a container that is made up of the whole distinct items found in all of the $XML\ node-based$ transactions. Each item in item pool has an identified item ID in order to distinguish one item from others.

We put all the items in the $XML\ node-based$ transactions into the item pool. During the inserting, the item pool will delete the duplicate item such that an item pool is the container that hold all of the distinct items that a group of XML documents have. In addition, each item in the item pool is given an fixed and distinct item number called item ID. Figure 11 shows the item pool of the specific example we explore of the DOM trees in Figure 4. There are totally 40 items in this item pool, each of which has an item ID. For instance, the item ID of $Conference * ConfDate$ is 1, $Conference * City$ is 8, and $Organization * ConfLoc$ is 18 etc. This item

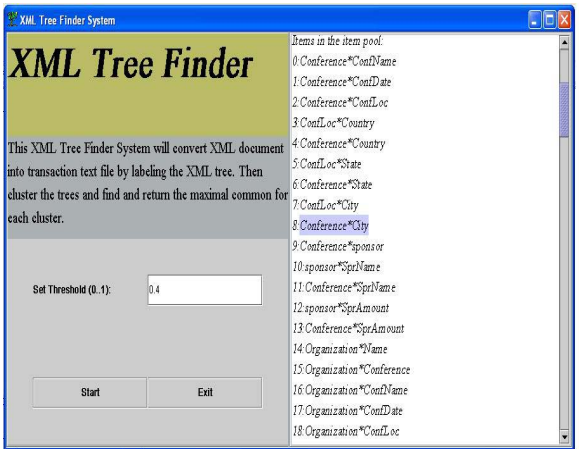

Figure 11: Item Pool of a group of XML Document

ID is kept same as the item ID in the later converted item ID in the $XML\ text-based$ transaction. In addition, the system will use the item ID to retrieve the corresponding item name which is the node pair name in the final maximal common tree.

4.4 Convert XML Node-based Transaction to Text-based Transaction

The purpose of this conversion is to simplify the computation for the next step which implement the *apriori* algorithm to cluster the XML documents and return the maximal common tree for each cluster. Therefore, it will increase the efficiency of the tree finder system too.

The structure of the $text-based$ transaction of the XML document is a list of item node whose data structure is a pair $(itemID, status)$. Each transaction has the same total number of items as in the item pool. For the case we exam above, each transaction has 40 items whose $itemIDs$ among 0 to 39. The *status* field indicates whether the transaction contains this item or not. O stands for not containing, and 1 for containing. All item statuses of each transaction are initialized to 0 which means that transaction doesn't contain any item from the beginning.

The DOM trees then are traversed again one by one and compare their node pairs with the node pairs in

itemID	Node Pair
0	Conference*ConfName
1	Conference*ConfDate
2	Conference*ConfLoc
3	ConfLoc*Country
4	Conference*Country
5	ConfLoc*State
6	Conference*State
7	ConfLoc*City
8	Conference*City
9	Conference*sponsor
10	sponsor*SprName
11	Conference*SprName
12	sponsor*SprAmount
13	Conference*SprAmount
14	Organization*Name
15	Organization*Conference
16	Organization*ConfName
17	Organization*ConfDate
18	Organization*ConfLoc
19	Conference*RegistFee
20	Organization*RegistFee
21	Organization*KeyField
22	Company*Product
...	...
38	AssoID*Reviewer
39	AssoID*RewName

Figure 12: ItemId and Corresponding Node Pair in Item Pool

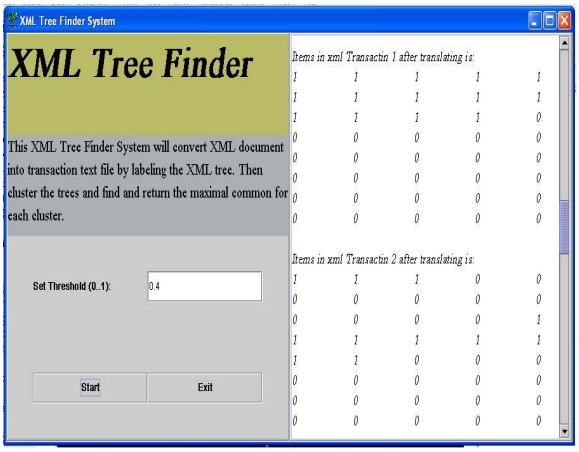

Figure 13: XMl Text-based Input Transaction after Conversion

the item pool. If the node pair in the DOM tree can be found in the item pool, the corresponding status is changed to 1 which means that transaction contain this item (node pair). Otherwise, it keeps 0 indicating that there is no this item in this transaction. In the running example, the item pool is as Figure 11. We can get the table to show the item ID and corresponding node pair as in Figure 12. The $XML\ text-based$ input transaction after conversion corresponding to tree A of Figure 8 is as figure 13. We call it $xmlT$. In this $xmlT$, we can tell that the status of of $item\ ID$ 2 is 1 because the DOM tree of tree A of Figure 8 contains the node pair $Conference*ConfLoc$ whose ID in item pool is 2. But the status of $item\ ID$ 18 is 0 because there is no node pair $Organization*ConfLoc$ whose ID is 18 in the item pool for the DOM tree of tree A of Figure 8.

The running screen shoot can be seen in Figure 14 in which the first two XML document's $text-based$ input transaction are shown.

4.5 Compute and Return the Maximal Common Tree of each Cluster of the Labeled Tree

The *Apriori Algorithm* [23] which is one of the famous clustering method is implemented to apply to

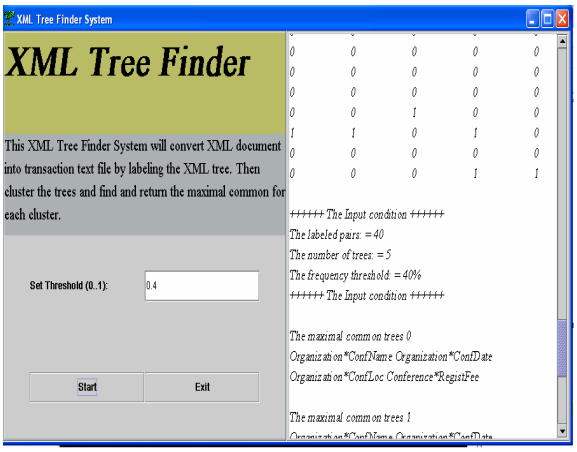

Figure 14: Input Condition of the Tree Finder System

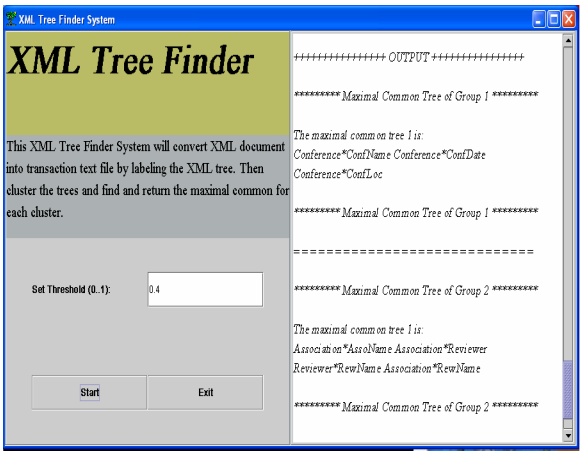

Figure 15: Output of Tree Finder System with Threshold 0.4

the set of $XML\ text-based$ input transaction over the items I identifying the pairs of labels, with the frequency threshold (a.k.a minimum support) set to ϵ. The screen shoot of input condition can be seen in Figure 14 for the specific example.

As for the tree structure in Figure 8, the five XML DOM trees will be divided into two clusters as follows. Cluster c_1 is made up of tree A, B, and C. Whereas, cluster c_2 contains tree D and E. Let LFI be the abbreviation of Largest Frequent Item. Therefore, we will get the following output after this step with the threshold $\epsilon = 0.4$. While, 0, 1, 2, 31, 32, 33, and 34 are the item ID.

$\text{LFI}(c_1) = \{0, 1, 2\}$
$\text{LFI}(c_2) = \{31, 32, 33, 34\}$

If the frequent threshold ϵ is set to 0.6, the output should be as following. The reason is the minimum support of item 0, 1, and 2 is 0.6. On the other hand, the minimum support of item 31, 32, 33, and 34 is only 0.4 so that they won't show up in the final output.

$\text{LFI}(c_1) = \{0, 1, 2\}$
$\text{LFI}(c_2) = \emptyset$

When the frequent threshold ϵ is set to 0.8, both $\text{LFI}(c_1)$ and $\text{LFI}(c_2)$ will become $\emptyset$, since no item has the minimum support above 0.8.

$\text{LFI}(c_1) = \emptyset$
$\text{LFI}(c_2) = \emptyset$

4.6 Retrieve the Node Pairs from the Item Pool

The output of the above step is text-based maximal common tree which is composed of a collection of item ID. For instance, {0, 1, 2}. We need to retrieve the corresponding node pair based on the original item pool we got from the XML document. The last step of the XML Tree Finder System will do this job. Let MCT stand for Maximal Common Tree. For the case with $\epsilon = 0.4$, the system will return the following maximal common tree, because $Conference * ConfName$ has item ID 0, $Conference * ConfDate$ has item ID 1 etc.

$\text{MCT}(c_1) = \{Conference * ConfName, Conference * ConfDate, Conference * ConfLoc\}$
$\text{MCT}(c_2) = \{Association * AssoName, Association * Reviewer, Reviewer * RewName, Association * RewName\}$

The screen shoot of this case can be seen in Figure 15.

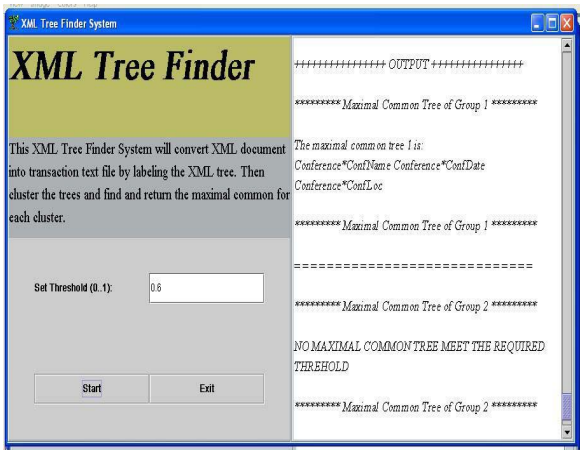

Figure 16: Output of Tree Finder System with Threshold 0.6

When the frequent threshold ϵ goes to 0.6, the output is as following, because only the item $Conference*ConfName$, $Conference*ConfDate$, $Conference*ConfLoc$ have the minimum support above 0.6. The screen shoot of this case is showed in Figure 16.

MCT(c_1) = $\{Conference * ConfName$, $Conference*ConfDate$, $Conference*ConfLoc\}$
MCT(c_2) = $\emptyset$

When the frequent threshold ϵ increases to 0.8, both MCT(c_1) and MCT(c_2) become $\emptyset$, since no item in both clusters has the minimum support above 0.8. The screen shoot of this case is showed in Figure 17.

MCT(c_1) = $\emptyset$
MCT(c_2) = $\emptyset$

One extreme case is when the frequent threshold ϵ has the top value 1 that makes both MCT(c_1) and MCT(c_2) become $\emptyset$ also, since no item in both clusters has the minimum support 1. The screen shoot of this case is showed in Figure 18.

MCT(c_1) = $\emptyset$
MCT(c_2) = $\emptyset$

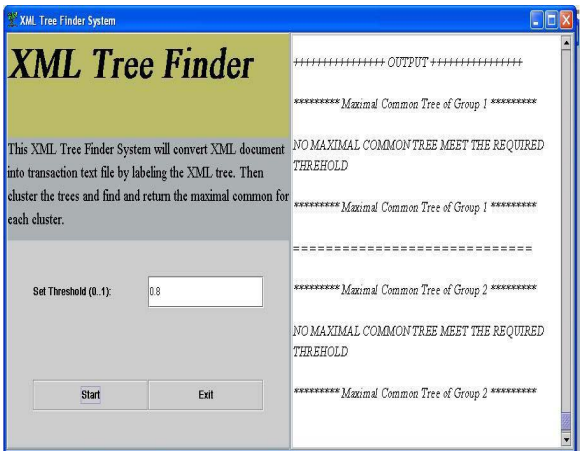

Figure 17: Output of Tree Finder System with Threshold 0.8

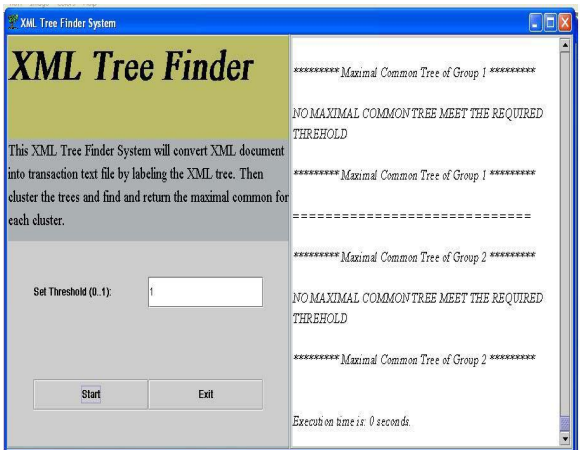

Figure 18: Output of Tree Finder System with Threshold 1

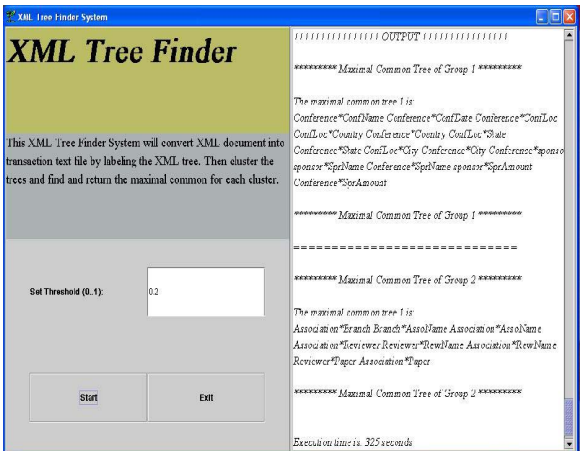

Figure 19: Output of Tree Finder System with Threshold 0.2

Another significant test case is when the frequent threshold ϵ has the lower value like 0.2 that returns both MCT(c_1) and MCT(c_2) bigger maximal common trees. In addition, it increases the system running time sharply to get the result. For all of the cases with minimum support more than 0.4, the running time is around 3 seconds. But it takes 325 seconds to return the maximal common tree with $\epsilon = 0.2$. The screen shoot of this case is showed in Figure 19.

$$\text{MCT}(c_1) = \emptyset$$
$$\text{MCT}(c_2) = \emptyset$$

5 Conclusion and Future Work

In this paper, we proposed an XML Tree Finder System which is the fundamental step for follow-up XML data mining. Based on the formal background knowledge, the whole process of the system was illustrated. Some test cases were given and analyzed.

As for the future work, there are many related and future ones that can be done based on this system. First of all, the data can be grasped directly from XML database, since the current system assume that those XML documents are already captured from the database. Secondly, it can use SAX other than DOM for the data binding because using DOM is more memory consumed so that it will make the system worse when it compute massive data. Thirdly, the current return result is the node pairs of the maximal common tree. It will be better if the true tree structure is shown on screen. Finally, the standard apriori-algorithm is implemented in this system, but there is some point in this algorithm which can be improved in order to make the system faster.

References

[1] Using jdbc to insert data from xml into a database. Presented by developerWorks of IBM.

[2] Industry applications of data mining, 1999. This chapter 8 contains examples of how data mining is used in banking/finance, retailing, healthcare, and telecommunications. The purpose of this chapter is to give the user some ideas of the types of activities in which data mining is already being used and what companies are using them.

[3] Iliana Avila-Campillo, Todd J. Greeny, Ashish Gupta, Makoto Onizukaz, Demian Raven, and Dan Suciu. Xmltk: An xml toolkit for scalable xml stream processing. Describe a toolkit for highly scalable XML data processing, consisting of two components.

[4] Akmal B.Chaudhri, Awais Rashid, and Roberto Zicari. *XML Data Management: Native XML and XML-Enabled Database Systems*. Addison Wesley, 2003.

[5] Ming-Syan Chen, Jiawei Han, and Philip S. Yu. Data mining: An overview from database perspective.

[6] Two Crows Corporation, editor. *Introduction to Data Mining and Knowledge Discovery*. 10500 Falls Road, Potomac, MD 20854 (U.S.A.), third edition, 1999.

[7] Saso Dzeroski. Multi-relational data mining: An introduction. Jozef Stefan Institute, Jamova 39, SI-1000 Ljubljana, Slovenia.

[8] Michael Gilman. Nuggets® and data mining. Data Mining Technologies Inc.1055 Stewart Avenue, Suite 1, Bethpage, NY 11714 Email mgilman@data-mine.com.

[9] Mark Graves. *Designing XML Databases*. Prentice Hall PTR, Upper Saddle River, NJ 07458, 2002.

[10] David Hand, Heikki Mannila, and Padhraic Smyth. *Principles of Data Mining*. A Bradford Book The MIT Press, Cambridge Massachusetts and London England, 2001. A comprehensive, highly technical look at the math and science behind extracting useful information from large databases.

[11] Djoerd Hiemstra. A database approach to content-based xml retrieval. University of Twente, Centre for Telematics and Information Technology P.O. Box 217, 7500 AE Enschede, The Netherlands.

[12] Shelley Higgins. *Oracle9i XML Database Developer's Guide - Oracle XML DB*. Oracle Corporation, 2002.

[13] Zachary G. Ives, Alon Y. Halevy, and Daniel S. Weld. An xml query engine for network-bound data. *VLDB*. Department of Computer Science and Engineering, Box 352350, University of Washington, Seattle, WA 98195-2350.

[14] Hillol Kargupta, Byung-Hoon Park, Daryl Hershberger, and Erik Johnson. Collective data mining: A new perspective toward distributed data mining. School of Electrical Engineering & Computing Science Wsshington State University Pullman, WA99164-2752, USA.

[15] Edwin M. Knorr and Raymond T. Ng. Algorithms for mining distance-based outliers in large datasets. *VLDB*, pages 392–403, 1998. New York, August 24-27.

[16] Jung-Won Lee, Kiho Lee, and Won Kim. Preparation for semantics-based xml mining. *IEEE*, 0-7695-1119-8/01:345–352, 2001. Department of Computer Science & Engineering, Ewha Institute of Science and Technology, Cyber Database Solutions Anstin, Texas.

[17] Rosa Meo and Giuseppe Psaila. Toward xml-based knowledge discovery systems. *IEEE*, 0-7695-1754-4/02:665–668, 2002. Universitá degli Studi di Torino Dipartimento di Informatica Universitá degli Studi di Bergamo Facoltá di Ingegneria.

[18] T. Niblett. A study of generalisation in logic programs. In Proceedings of the Third European Working Session on Learning, pages 131138. Pitman, London, 1988.,.

[19] G. Plotkin. A note on inductive generalization. B. Meltzer and D. Michie, editors, Machine Intelligence 5, pages 153163. Edinburgh University Press, Edin- burgh, 1969.,.

[20] Alexandre Termier, Marie-Christine Rousset, and Michéle Sebag. Treefinder: a first step towards xml data mining. *IEEE*, 0-7695-1745-4/02:450–457, 2002. LRI-CNRS UMR 8623, Université Paris-Sud, 91405 Orsay.

[21] W3School. https://www6.software.ibm.com/developerworks/edu jaxb/index.html? Introduction to Data Binding, Unmashalling: from XML to Java objects, Mashalling: from Java objects to XML, and future exploration in Data Binding.

[22] W3School. www.w3schools.com. W3Schools - Full Web Building Tutorials. At W3Schools you will find all the Web-building tutorials you need, from basic HTML and XHTML to advanced XML, XSL, Multimedia and WAP.

[23] John Wang. *Data Mining: Opportunities and Challenges*. Idea Group Publishing, 2003. In this text, an international team of 44 data mining experts specifically explore new methodologies or examine case studies in this new and multi-disciplinary topic.

[24] Kevin Williams, Michael Brundage, Patrick Dengler, Jeff Gabriel, Andy Hoskinson, Michael Kay, Thomas Maxwell, Marcelo Ochoa, Johnny Papa, and Mohan Vanmane. *Professional XML Databases*. Wrox Press Ltd, Arden House, 1102 Warwick Road, Acocks Green, Birmingham, B27 6BH, UK, 2000.