Automatic Construction of Online Catalog Topologies

Wing-Kin Sung, David Yang, Siu-Ming Yiu, David W. Cheung, Wai-Shing Ho, and Tak-Wah Lam

***Abstract*—Given a set of products, where each is characterized by a set of attribute values, an online catalog is an organization of a set of product pages on the web through which users can access their required product information. A good online catalog is crucial to the success of an e-commerce web site.**

Traditionally, an online catalog is mainly built by hand. To what extent this can be automated is a challenging problem. Recently, there have been investigations on how to reorganize an existing online catalog based on some criteria, but none of them has addressed the problem of organizing an online catalog automatically *from scratch*. This paper attempts to tackle this problem.

We model an online catalog organization as a decision tree structure and propose a metric, based on the *popularity* of products and the relative *importance* of product attribute values, to evaluate the quality of a catalog organization. The problem is then formulated as a decision tree construction problem. Although traditional decision tree algorithms, such as C4.5, can be used to generate online catalog organization, the catalog constructed is generally not good based on our metric. An efficient greedy algorithm (GENCAT) is thus developed, and the experimental results show that GENCAT produces better catalog organizations based on our metric.

***Index Terms*—Decision tree, metrics, online catalog, tree optimization.**

I. Introduction

MORE and more companies are putting information about their products onto the Web. The users can use search engines or browse the online catalog structures to retrieve product information that interests them. Traditional online catalog research such as [1]–[3] focused on providing better searching strategies for search engines to retrieve the information in an existing online library catalog. Other research such as [4]–[6] focused on building an interface to make the searching easier. However, search engines are considered more useful only for those users who know exactly what they are seeking, while a browsable online catalong is more useful when the users are less certain. A good online catalog that allows users to get required product information easily and efficiently is crucial to the success of an e-commerce website.

Manuscript received September 22, 2000; revised September 17, 2002. This paper was recommended by Associate Editor R. Rada.

W.-K. Sung was with the E-Business Technology Institute, University of Hong Kong. He is now with the School of Computing, National University of Singapore, Singapore (e-mail: ksung@comp.nus.edu.sg).

D. Yang was with the Department of Computer Science and Information Systems, University of Hong Kong, Hong Kong and was on leave from the Department of Mathematics and Computer Science, St. Joseph's University, Philadelphia, PA. He is now with the Department of Mathematics and Computer Science, California State University at Hayward, CA, USA (e-mail: dyang@mcs.csuhayward.edu).

S.-M. Yiu, D. W. Cheung, W.-S. Ho and T.-W. Lam are with the Department of Computer Science and Information Systems, University of Hong Kong, Hong Kong (e-mail: smyiu@csis.hku.hk, dcheung@csis.hku.hk, wsho@csis.hku.hk, twlam@csis.hku.hk).

Digital Object Identifier 10.1109/TSMCC.2002.806055

At present, the organization of an online catalog is largely treated as an art form and typically done by hand (see, for example, Rosenfeld and Morville [7]). To what extent this complicated task can be automated is still a challenging problem, although recent progress has been made. Perkowitz and Etzioni [8] supplement an existing organization by constructing index pages for related pages automatically. These related pages are identified by mining the visitors' logs. Green [9], on the other hand, adds some crosslinks between pages that are likely to be related by studying the contents of the pages.

While these works enhance the existing structure, they do not consider the quality of the existing structure. Garofalakis *et al*. [10] address this quality issue using *page popularity*. They reorganize the existing structure locally by swapping children and parent pages if the child page is more popular. This approach assumes that there are no semantic relationships between pages thus making the swapping of pages inappropriate. In general, this is untrue. None of these approaches has addressed the problem of organizing an online catalog automatically *from scratch*. While consultation with users is of course essential, we suggest that it is possible to do this in conjunction with automatic methods.

An online catalog suggests a topology of products while the root page represents the entire set of products. By following the links to the next level, products are divided into different subsets according to some properties (attributes), and a single product is finally identified. In this sense, the construction of online catalogs bears a resemblance to classification, and the problem is formulated as a decision tree construction problem.

There are many approaches in classification including neural networks [11], example-based classification [12], decision trees [13], and Bayesean classification [14]. However, except for decision trees, these approaches do not construct a catalog-like topology and, therefore, cannot be directly applied to our problem. Even though decision tree construction algorithms can build a catalog, it may not be able to build a *good* online catalog.

A good online catalog organization should allow visitors to locate *popular* products *easily* and *efficiently*. According to our knowledge, however, there does not currently exist any quantitative method to evaluate a catalog tree. We propose, therefore, a metric to evaluate the quality of online catalog organization. The metric takes two important factors into account. One is the *popularity* of each product so that fewer "clicks" are needed to get to more popular products. The other is how likely a visitor can get to the right product by following the links provided by the catalog. In deciding which link to follow, visitors often need to answer a question related to an attribute of that product. If visitors do not care about the question, they could end up following the wrong link. To ask more important attribute questions ear-

TABLE I
CHARACTERISTICS OF FIVE DIFFERENT MOBILE PHONES

ID	size (dimensions in cm)	access wap	call waiting	popularity
A	$5 \times 10 \times 2$ (big)	no	no	25
B	$5 \times 10 \times 2$ (big)	no	yes	20
C	$3 \times 5 \times 0.5$ (small)	no	yes	20
D	$3 \times 5 \times 0.5$ (small)	yes	no	50
E	$3 \times 5 \times 0.5$ (small)	yes	yes	100

lier, that is, nearer to the root page makes more sense. If this were the case, visitors would have a higher chance of getting to the right products.

To ensure the proposed metric is a feasible one, we also develop a quantitative framework using a well-established marketing methodology called *conjoint analysis* [15] to capture reliable measures of the relative importances of attributes.

In general, based on the proposed metric, traditional decision tree algorithms, such as C4.5, do not give us a good solution. An efficient greedy algorithm, (GENCAT), has been developed. Experimental results show that GENCAT does give better catalog organizations under our metric. It is also emphasized that the output from GENCAT can be used as an initial design, and we provide a subsequent approach to modify this design.

Section II presents the problem of automatic online catalog construction. Section III discusses the metric, the greedy algorithm, and the analysis of the proposed solution. Other related issues such as conjoint analysis and how to change the design produced by GENCAT will be discussed in Section IV. Discussion and conclusion will be given in Section V.

II. PROBLEM DESCRIPTION

A. Model for Online Catalog Design

When visitors try to locate product information from an online catalog, either they make use of the search engine or they follow the structure of the catalog going from one web page to another. Since the vocabulary used by the visitor may not match that used by the site, or the visitor does not know exactly what they are looking for, the search engine, will not always be able to produce satisfactory results. The organization of the catalog is especially important in cases such as these.

Although a real online catalog may have crosslinks between product pages, usually characterized by "related items," the underlying structure of a catalog is basically a tree. The root page (or main page) represents the entire set of products and variety of choices to follow, represented by the parent-child links that indicate a subset of products. In other words, an online catalog can be regarded as a topology of the set of products. When visitors navigate through an online catalog, they essentially traverse a tree to find the product that most interests them. If we ignore all those cross- and backward links, the product information pages are usually seen at the leaves of the tree. We model, therefore, a catalog structure as a tree with leaves corresponding to the product pages and the inner nodes corresponding to *navigational* pages. By navigational, the primary purpose means pages that guide the user toward the appropriate product pages.

Each navigational page implicitly represents a subset of product pages. The links provided in the navigational page further classify the corresponding subset into smaller subsets. The decision regarding which link to follow usually depends upon the answer to the initial question regarding properties (attributes) of products that interest the user. In a clothing web site, for example, a particular navigational page may want to subdivide the set of clothing into men's clothing and women's clothing. This navigational page would make use of the gender attribute to classify its clothing. The gender attribute has two attribute values: man and woman. The set of clothing is partitioned into two sets depending on whether the gender attribute is equal to man or woman. In other words, when a visitor navigates through this page, the question "Are you interested in men's clothing or women's clothing?"may be asked in order for the visitor to choose the next link.

To simplify discussion, the following assumptions on product pages are made. Consider a set of product pages P.

1) All product pages in P are characterized by the same set of non-null attributes A.
2) Each product page can be uniquely identified by its set of corresponding attribute values.

Table I shows an example of five mobile phones which are characterized by three attributes: size, access wap, and call waiting. Note that we regard a mobile phone of dimension $5 \times 10 \times 2$ as large and dimension $3 \times 5 \times 0.5$ as small.

An online catalog organization is formulated as a *catalog tree* as follows.

Definition of Catalog Tree: Given a set of product pages P, a catalog tree for P is a tree $T = (V, E)$ where V is the set of nodes and E, is the set of edges such that we have the following.

1) Every product page in P is mapped to a unique leaf node in V.
2) Each internal node in V is labeled by an attribute in A.
3) For each edge $e = (u, v)$, where u is the parent, if u is labeled by an attribute a, then the edge e must be labeled by a valid attribute value of a.
4) For each product page p, if the attributes $a_1, a_2, \ldots, a_i$ with corresponding attribute values, $x_1, x_2, \ldots, x_i$, appears on the path from the root to p, then p can be uniquely identified by the attribute set, $a_1, a_2, \ldots, a_i$ with corresponding attribute values, $x_1, x_2, \ldots, x_i$.

Fig. 1 shows three different catalog trees for the set of product pages described in Table I. In order to know which catalog tree is better, we need some quantitative measurements. In Section II-B, we will propose the metrics to measure the quality of the catalog trees.

B. Metric

As shown in the mobile phone example, for the same set of products, there are more than one possible catalog trees. The

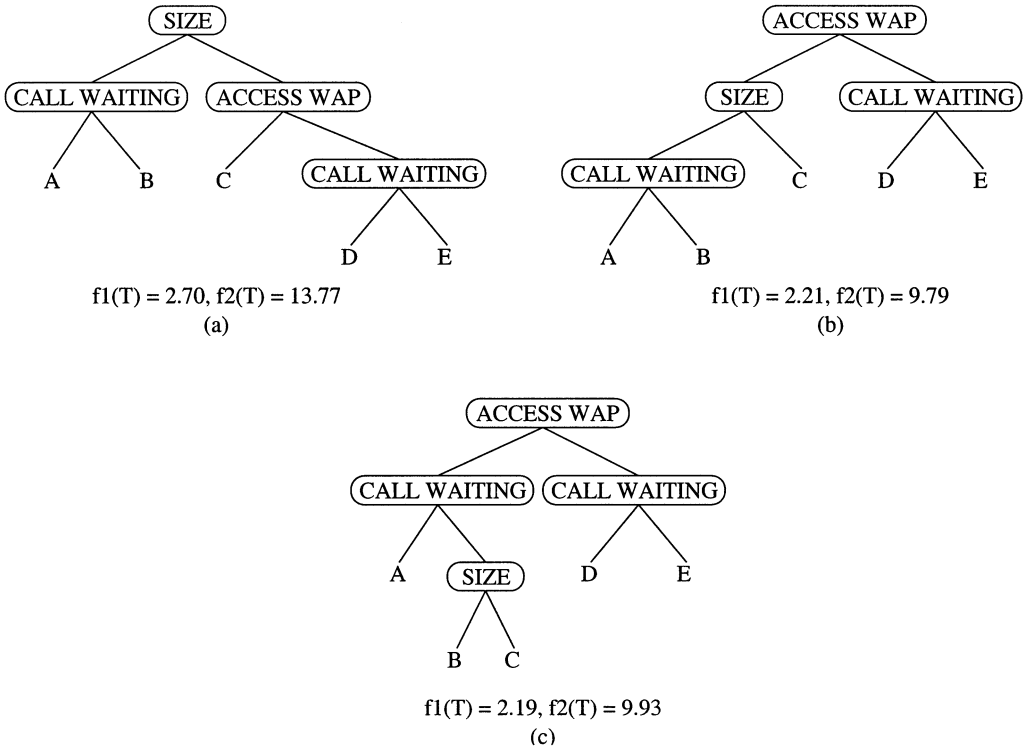

Fig. 1. Different catalog trees for same set of products.

question of which one is the best is not easy to answer. According to our knowledge, there are no quantitative measures proposed to evaluate the quality of a catalog tree. Basically, a good catalog tree should help users locate *popular* product information *efficiently* and *easily*. In this section, based on this idea, we try to define a reasonable metric which allows us to evaluate how good a catalog tree is.

To quantify this measurement idea, we need to have a quantitative measure for the popularity of the product pages. For a company that has been in business offline, the popularity of product pages can be approximated using past sales figures. These figures could either be in number of units sold or in the monetary value of the products sold. For a company that has been online for a while, the number of visits to each product page can provide an approximation to its popularity. For new products, we can project the popularity of each product by using marketing techniques.

To address the efficiency issue, one approach is to minimize the number of "clicks" the visitor uses to get to the popular product pages. Minimizing this number also relieves the loading of the web server as visitors do not then need to traverse through too many web pages before getting to their target page. In our model, the number of clicks to get to a product page can be measured by the depth of the page in the catalog tree. For example, if referring to Table I, Fig. 1(a) is certainly not a good tree as E is a popular product, but the corresponding product page has the longest path. Based on this observation, the next paragraph proposes the metric $f_1(T)$, which tries to measure the average weighted depth of a catalog tree. A good catalog tree should try to minimize $f_1(T)$.

Definition of the Average Weighted Depth Metric: Consider a catalog tree T for a set of product pages P. For every product page $p \in P$, let pop_p be the popularity of p and let $depth(p)$ be the depth of p in T. We define the average weighted depth of T, $f_1(T)$ as

$$\frac{\sum_{p \in P} pop_p \times \text{depth}(p)}{\sum_{p \in P} pop_p}.$$

Out of the three catalog trees in the mobile phone example, Fig. 1(c) is the best based on $f_1()$. We believe that in [10], the rationale behind swapping the parent and child page if the child page is more popular is similar to the underlying concept of $f_1()$.

However, $f_1()$ does miss one element for good catalog trees, that is, it does not address the easiness issue. When visitors browse through the catalog tree level-by-level, they need to pick a link in each level by answering an abstract question: "which value of attribute a interests you the most?" If the attribute does not appear to be of interest, it may not be easy for the visitor to locate the right product page. For example, in Fig. 1(c), consider the left child of the root; the user is asked to select the type of phone with or without the call-waiting function. If the user does not really care about this function, but wants to get a small size mobile phone without access to WAP, however, then the user may end up with product page A instead of product page C.

To make this concept more concrete among the attribute values which describe a particular product, some are important (key features), while others are not. A good catalog tree should avoid using attributes in the navigational pages that are not important.

The importance of an attribute value can be assigned by the designer of the online catalog but this is undesirable and can be quite subjective. Instead, let us borrow a concept, called

part-worth, from the marketing research. Part-worth of an attribute value measures the contribution this particular attribute value has on the overall utility of a product [16]. In other words, it reflects upon the attribute-value's relative importance to the product. The higher the part-worth of an attribute value, the more people will think that this attribute value is important (see a more in-depth discussion of part-worth in Section IV-A). Assuming that we have obtained the part-worths of the attribute values, we show a better metric which also takes this into account.

Let P be a set of products and A be the set of attributes used to characterize P. For every attribute $a \in A$, let V_a be the set of attribute values of a. Observe that each product $p \in P$ can be described using an attribute value vector $(x_{a_1}, x_{a_2}, \ldots)$, where $x_{a_i} \in V_{a_i}$ for every attribute $a_i \in A$. Among the attribute values in the vector, some of them are critical (important) characteristic of p while the other are noncritical (unimportant). Our aim is to build a catalog tree T for P (i.e., P is the set of leaves of T) which tries to achieve two things. Let I_p be the set of attribute values appearing along the path from the root of T to p for any product page p.

- We try to reduce the depth of every product in T, especially for the popular products.
- For each product p, we try to minimize the number of unimportant attribute values in I_p.

To achieve the above two things, we proposes the metric $f_2(T)$, which tries to measure the average weighted unimportance of a catalog tree.

Definition of the Average Weighted Unimportance Metric: We define a measurement unimport_v, for every attribute value_v, such that unimport_v has a large value if v is unimportant. Then, we define the metric as

$$f_2(T) = \frac{\sum_{p \in P} pop_p \left(\sum_{v \in I_p} \text{unimport}_v \right)}{\sum_{p \in P} pop_p}.$$

Subsequently, for any product $p \in P$, if $\sum_{v \in I_p} \text{unimport}_v$ is large; then p may have a high depth in T or the number of unimportant attribute values in I_p is large. So, reducing $\sum_{v \in I_p} \text{unimport}_v$ captures the above two criteria.

What remains is to define the measurement unimport_v for every attribute value v. There are many different ways to achieve this. Here, we try to define unimport_v based on the part-worth of v, that is, u_v. It is nature to deduce that the larger the u_v, the more important the v is. Hence, we denote $unimport_v = m - u_v$ where m is a constant which is larger than the part-worth of all the attribute values.

Table II shows the values of part-worths and unimportances for the mobile phone example. Based on this table, Fig. 1(b) should give the best organization under metric $f_2()$.

C. Problem Definition

Formally speaking, the automatic online catalog construction problem can be defined as follows. Let A be the set of attributes that characterize the set of product pages P.

Input: A set of product pages P and the metric $f_2()$;
Output: A catalog tree T for P which minimizes $f_2()$.

TABLE II
PART-WORTHS OF ATTRIBUTE VALUES IN THE MOBILE PHONE EXAMPLE

attribute	value	part-worths	unimportance
size	big	0	6
size	small	4	2
access wap	yes	5	1
access wap	no	0	6
call waiting	yes	2	4
call waiting	no	0	6

The automatic online catalog construction problem is NP-hard, so in Section III, we will present a greedy algorithm for constructing a local optimal catalog tree under the proposed metric.

III. OUR PROPOSED SOLUTION

Since a catalog tree bears a resemblance to a decision tree, traditional decision tree construction algorithms, such as C4.5 [17], can be used to build a catalog tree. The catalog constructed, however, is generally not good based on our metric because the information gained through the importance measurement was not considered. In this section, we present the greedy algorithm GENCAT for constructing a local optimal catalog tree under the proposed metric. The running time of the algorithm will then be analyzed and the performance of the algorithm will be compared with those of other decision tree construction algorithms.

A. Greedy Algorithm–GENCAT

The algorithm consists of two steps. The first step generates an initial catalog tree T for the product pages. The next step restructures the tree by considering different attribute choices at each node in an in-depth manner. The output of the algorithm will then be a local optimal catalog tree with respect to the metric, $f_2(T)$.[1] The following gives the details of these steps.

We start by building an initial catalog tree. Different initial trees can be used. One way to generate the initial tree is that we start building it by arbitrarily choosing an attribute at the root. The product pages will be partitioned into different subsets according to the corresponding attribute values. A child node will be created for each subset. For each child node with more than one product page, we arbitrarily pick another attribute and continue the partitioning process until every leaf node contains a single product page. Fig. 2 shows an initial catalog tree based on a selection of tires from Table IV.

After getting the initial tree T, the second step, which is the core of the algorithm, tries to restructure T to optimize the value of $f_2(T)$. Starting from the root, we do the restructuring in an in-depth manner. For each node v, we consider every possible attribute a. Then, the subtree rooted at v is reorganized using a as the attribute for the node v. This reorganization will modify the structure of the subtree based on *the original structure of* T. This restructuring will be shown as always feasible using a procedure called Rebuild (). Among all these possible alternatives, we choose the one that gives the best value of $f_2(T)$. When the

[1]In fact, the algorithm is quite general in the sense that if we substitute the metric $f_2(T)$ by another appropriately defined metric, the algorithm still works

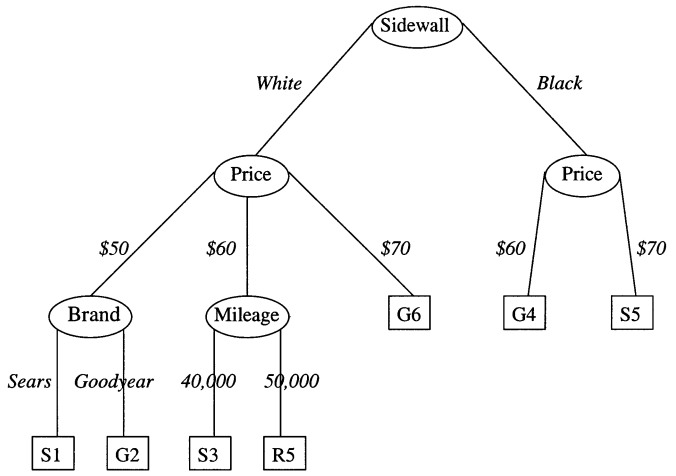

Fig. 2. Initial catalog tree for tires S1, S3, S5, G2, G4, G6, and R5 ($m = 10$, $f_2(T) = 22.24$).

TABLE III
COMPARISON BETWEEN GENCAT AND C4.5

	C4.5		GENCAT	
	mean	std dev	mean	std dev
$f_1(T)$	4.845	0.043	4.638	0.084
$f_2(T)$	25.56	0.95	14.88	1.10

algorithm stops, we end up with a local optimal catalog tree with respect to the metric $f_2(T)$.

The procedure Rebuild (T, v, a) will restructure the subtree of T rooted at v using attribute a in node v. The details are as follows.

1) Change the attribute in node v of T to a.
2) Partition the product pages in the leaves of the subtree rooted at v into groups $G_1, G_2, \ldots, G_k$ according to the corresponding attribute values of a
3) For each group G_i, let S_i be a tree constructed from T as follows.
   a) All the product pages not in G_i are removed.
   b) All the inner nodes which have one child are removed.
4) Replace all the subtrees which are attached to v by $S_1, S_2, \ldots, S_k$.

Fig. 3 shows an example how Rebuild () restructures the subtree rooted at w using attribute A_5 to replace A_2 in node w. The second step shows the result of partitioning leaf pages rooted at w according to the attribute values of A_5. The third step shows the result of restructing the subtree rooted at w based on the original structure of T. The final output is obtained by removing internal nodes which have only one child (for example, the nodes using attributes A_3 and A_4).

Fig. 4 shows the algorithm for GENCAT. We can make use of GENCAT even if not all attributes apply to every product. We can deal with this by first limiting ourselves to the attributes which do apply to all products. Next, the remaining products within each of the subtrees can be grouped by the groups of products which each have the same more specialized attributes.

GENCAT can also help if the catalog tree must start with a standard classification. An example of this situation would be a clothing retailer who wants to pre-divide the products into groups, say clothing for women, men, girls, boys, and infants. There can be shirts or blouses in each of these categories with the same set of attribute values. In such a case, the site designer would divide up the products into the appropriate subsets, then run GENCAT on each subset.

B. Example

This section shows an example of the resulting catalog after the restructing by GENCAT. Consider the table of tires in Table IV again. If for simplicity we assume the popularity of the tires is all the same (we arbitrarily set the popularity to 1), our catalog organization routine yields the catalog tree in Fig. 5. The aim of this example is to see the effect of unimportance of attribute values on the structure of the catalog tree.

In comparing Fig. 5 with the original tree, we note that using the sidewall attribute at the root produces a tree with a longer path for four of the seven products. Also, according to the part-worth values, brand and tread mileage are more important than sidewall and price, therefore, it is no good to put sidewall and price near the top of the catalog tree.

C. Performance Analysis

In this section, we will discuss the performance of the algorithm. It is shown that the time complexity of GENCAT is polynomial in the number of product pages and the number of attributes per product page. Then, we will compare the performance of our algorithm with those of other decision tree construction algorithms.

1) Time Complexity: Before analyzing the whole algorithm, we first show that Rebuild (T, v, a) can be computed in $O(|T^v|)$ time where T^v is the subtree of T rooted at v.

Lemma 1: Rebuild (T, v, a) can be computed in $O(|T^v|)$ time.

Proof: It can be proved using the same technique as in [18] Lemma 4.1. ■

It is obvious that the metric $f_2(T)$ can be computed in $O(|T|)$ time. The next lemma concludes the time complexity of GENCAT.

Lemma 2: GENCAT requires $O(|A||P|^2)$ time where A is the set of attributes and P is the set of products in the catalog.

Proof: Recall that GENCAT can be divided into two parts. The first part tries to build an initial catalog tree. By picking an arbitrary attribute to partition the product pages recursively until every partition contains only one product, we can build the initial tree in $O(|A||P|)$ time. Note that the size of the initial tree constructed is smaller than $2|P|$.

The second part tries to refine the structure of the catalog tree by the procedure Improve (T, r, f). As shown in Fig. 4, the first step of Improve (T, r, f) is to compute $T_a =$ Rebuild (T, r, a) for every attribute $a \in A$. It requires $O(|A||P|)$ time based on Lemma 1. The second step finds $R = T_a$ which minimizes $f_2(T_a)$ for all $a \in A$. Since $f_2(T_a)$ can be computed in $O(|T_a|) = O(|P|)$ time, this step needs $O(|A||P|)$ time. Hence, R can be computed in $O(|A||P|)$ time. After updating the attribute used in r, we call Improve () recursively to update the attributes used in the descendents of r. As the catalog tree

TABLE IV
TIRES TABLES

ID #	Brand	Tread Mileage (miles)	Price ($)	Sidewall	Utility
S1	*Sears*	*30,000*	*50*	*White*	5.2
S2	*Sears*	*30,000*	*70*	*Black*	2.2
S3	*Sears*	*40,000*	*60*	*White*	7.3
S4	*Sears*	*40,000*	*50*	*White*	8.1
S5	*Sears*	*50,000*	*70*	*Black*	5.7
S6	*Sears*	*50,000*	*60*	*White*	8.3
G1	*Goodyear*	*30,000*	*60*	*Black*	4.8
G2	*Goodyear*	*30,000*	*50*	*White*	6.3
G3	*Goodyear*	*40,000*	*70*	*White*	7.2
G4	*Goodyear*	*40,000*	*60*	*Black*	7.4
G5	*Goodyear*	*50,000*	*50*	*White*	9.3
G6	*Goodyear*	*50,000*	*70*	*White*	7.3
R1	*Goodrich*	*30,000*	*70*	*White*	0.8
R2	*Goodrich*	*30,000*	*60*	*White*	2.2
R3	*Goodrich*	*40,000*	*50*	*Black*	3.2
R4	*Goodrich*	*40,000*	*70*	*White*	4.3
R5	*Goodrich*	*50,000*	*60*	*White*	6.4
R6	*Goodrich*	*50,000*	*50*	*Black*	5.7

attribute	value	part-worth
Brand	*Sears*	2.4
Brand	*Goodyear*	3.3
Brand	*Goodrich*	0.0
Tread Mileage	*30,000*	0.0
Tread Mileage	*40,000*	2.7
Tread Mileage	*50,000*	3.5
Price	*$50*	1.7
Price	*$60*	1.5
Price	*$70*	0.0
Sidewall	*White*	1.2
Sidewall	*Black*	0.0

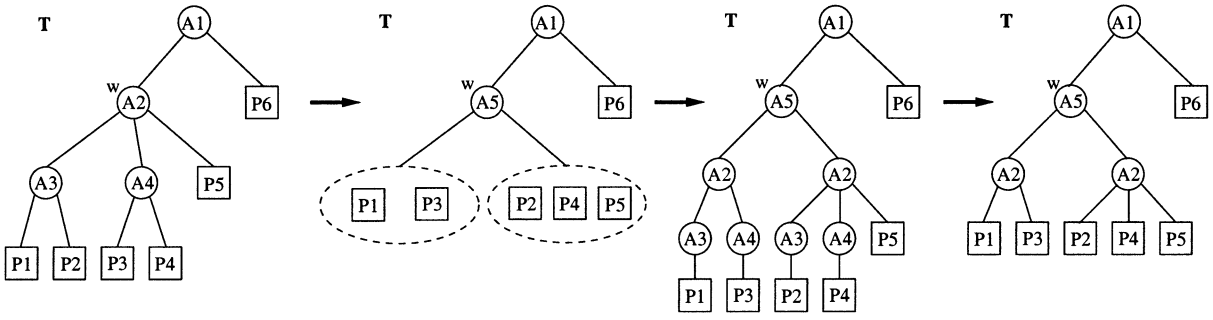

Fig. 3. Example which demonstrates Rebuild (T, w, A_5).

has at most $2|P|$ nodes, the total time required by Improve () is $O(|A||P|^2)$. ■

2) Performance Tests on Synthetic Data: The performance tests were divided into two parts. We first compared the results from GENCAT with those from C4.5. Then, we tested the performance of GENCAT over different input data sets.

Data Generation: The input data were generated using the following parameters:

- number of product pages n_p;
- number of attributes per product page n_a;
- maximum number of values per attribute n_v.

The number of product pages in a test set is ranged from 20 to 1000. The number of attributes per page is fixed for each test set. The number of attributes per product page is ranged from 4 to over 30. The maximum number of values per attribute is ranged from 1 to over 30.

The actual number of attribute values is chosen randomly. The attribute values are assigned according to either, a uniform distribution or a normal distribution. The uniform distribution models those cases where manufacturers offer all, or almost all, permutations of the attribute values. The normal distribution, models those cases where some attribute values are more popular. Popularity of the product pages are assigned according to the Zipf distribution.

The part-worth for each attribute value is generated randomly. Each attribute is also chosen to be, on the whole, important or unimportant. An attribute which is important has a single attribute value with a part-worth of zero and several attribute

GENCAT
INPUT: the set of products P and the metric $f_2()$
OUTPUT: a locally optimal catalog tree T for P with respect to $f_2(T)$
1. Build a decision tree T which consists of all the product pages;
2. $R = \text{Improve}(T, r, f)$ where r is the root of T;
3. Return R;

Improve(T, v, f)
1. For every attribute a, let $T_a = \text{Rebuild}(T, v, a)$;
2. Among all the T_a, let R be the T_a such that $f_2(T_a)$ is minimized;
3. For every non-leaf child u of v, let $R = \text{Improve}(R, u, f)$;
4. Return R;

Fig. 4. GENCAT algorithm for automatic catalog organization.

values which all have relatively high part-worths. An unimportant attribute has a single value with a part-worth of zero as well, but the other values of this attribute all have low part-worths. For our tests, 30% of the attributes were chosen to be important while 70% were chosen to be unimportant. The boundaries of the ranges of part-worths can also be specified. An unimportant attribute had part-worths ranged from 0 to 4 while an important attribute had part-worths ranged from 4 to 8 in our tests.

Comparison Between C4.5 and GENCAT: For the comparison between C4.5 and GENCAT, we tried 50 data sets, and each data set has 500 pages, and 20 attributes, and there are at most four possible values per attribute. Table III shows the means and standard deviations of the experiment results. It shows that although C4.5 could give similar result as GENCAT in terms of $f_1(T)$, the trees from C4.5 gives a much higher $f_2()$ values than those from GENCAT. This means that C4.5 does not consider the unimportance of the attribute values and tries to ask the users more questions on unimportant attributes, forcing a choice between unimportant attribute values in the internal nodes of the catalog trees.

Performance of GENCAT: We also tested two other aspects of the performance (performance ratio and computation time) of GENCAT over different input data sets. The performance ratio is defined as the ratio of $f_2()$ value of the catalog tree produced by GENCAT to that by an optimal tree generator which uses a brute force optimal algorithm with dynamic programming. Figs. 6 and 7 show the effects on the changes of n_p and n_a to the performance ratios.

We found that the increase of n_p or n_a would make GENCAT slightly less effective. However, GENCAT still has an average ratio less than 1.1 in our test cases, which means that our trees are just 10% deeper than the optimal ones. We only performed tests for up to 100 pages or 12 attributes because it is nearly infeasible (the running time for a single test case is more than 2 hrs.) to find an optimal tree for test cases with more product pages or attributes.

Figs. 8–10 show that GENCAT has a good scalability over the parameters. The running time was only slightly quadratic (nealy linear) to the number of product pages n_p or the number of attributes per page n_a. For a data set with 1500 product pages and 20 attributes per product page, the time for running GENCAT was only 20 min.

IV. Other Issues

In this section, we will describe a framework based on a well-developed marketing methodology, called conjoint analysis, to capture the necessary data, namely, the importance of attributes, for the calculation of $f_2()$. Then, we will show how the output from GENCAT can be used as an initial design and be altered by the designer.

A. Conjoint Analysis

It is not desirable to leave the assignment of the actual importance values to a site-designer. There are many different ways to achieve this. One way is to use the marketing concept of *part-worth*, which measures the contribution of a particular attribute value to the overall *utility* of a product [16]. This establishes a uniform scale for comparing the contribution of different attributes to the overall valuation of the product. The higher the part-worths of the attribute values of the product, the more people will be willing to spend for that product.

To accurately determine the values of part-worths is not easy because it is not feasible to ask humans to rate each individual attribute value. Researchers in marketing will make use of a well-developed methodology called conjoint analysis to deduce the values of part-worth. In our case, based on *metric conjoint analysis* [16], a variant of conjoint analysis, we can capture the part-worths of the attribute values for the calculation of unimport_v. The idea is quite simple. A selected set of products is chosen and a test group of consumers is asked to rate each product on a numerical scale, such as one to ten. For each attribute value, we assign a variable to denote the value of the corresponding part-worth. Depending on the underlying model, a set of equations will be obtained.

For example, assuming the effects of any two attribute values are independent, then the utility of a product is the sum of the part-worths of all its attribute values. Consider a product with two attributes a_0 and a_1. The possible attribute values for a_0 are v and w, while the possible attribute values for a_1 are x and y. We are thus trying to solve for four variables u_v, u_w, u_x, and u_y, which are the part-worths of the four attribute values. If we were to generate the four products with the possible combinations, we would have the following overdetermined set of equations:

$$u_{v,x} = u_v + u_x \quad (1)$$

$$u_{v,y} = u_v + u_y \quad (2)$$

$$u_{w,x} = u_w + u_x \quad (3)$$

$$u_{w,y} = u_w + u_y \quad (4)$$

where $u_{i,j}$ represents the utility of the product with the attribute values of a_0 and a_1 equal i and j, respectively. We can find a solution with the least-squared error by using regression.

One concern is that the number of equations can become very large as the number of attribute values increases. Part of the

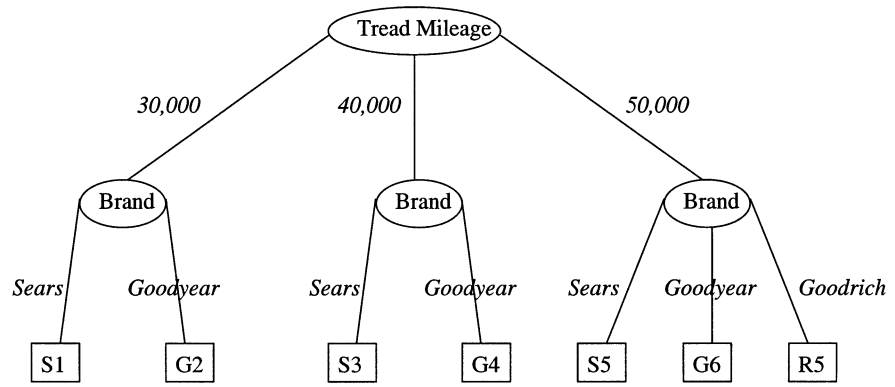

Fig. 5. Decision tree obtained based on average weighted unimportance ($m = 10$, $f_2(T) = 15.29$).

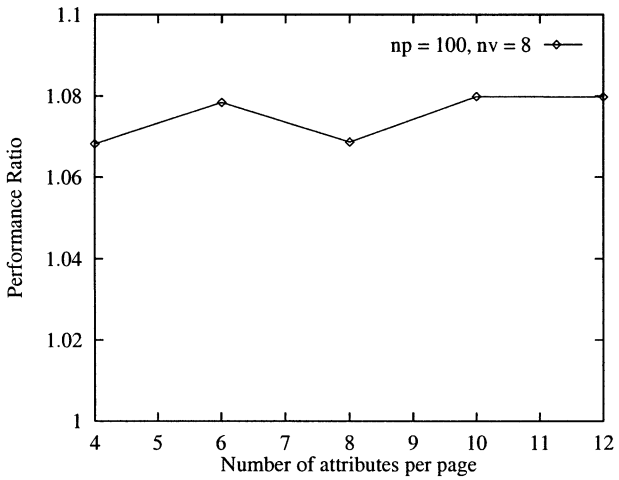

Fig. 6. Effect of number of attributes on the performance ratio.

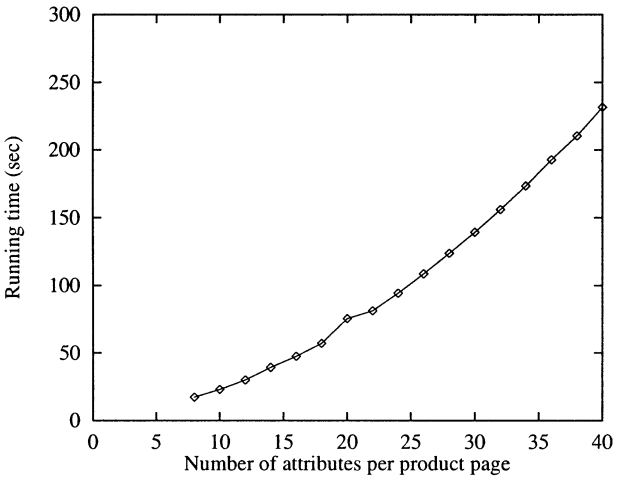

Fig. 8. Effect of number of attributes on running time.

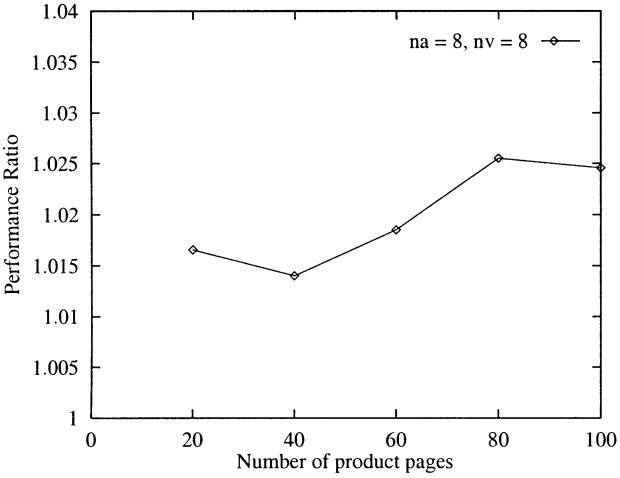

Fig. 7. Effect of number of product pages on the performance ratio.

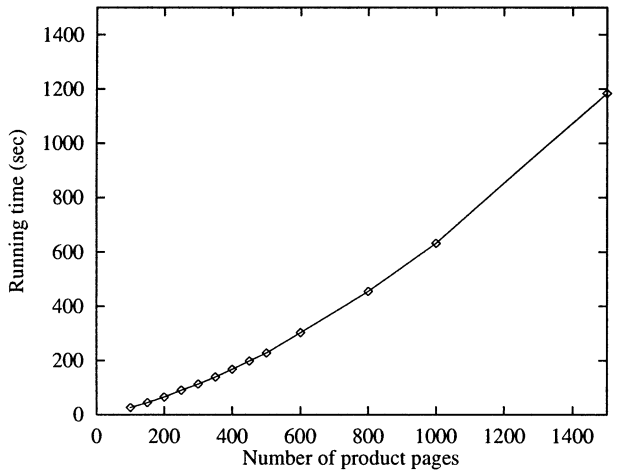

Fig. 9. Effect of number of product pages on running time.

reason conjoint analysis has succeeded is that the number of permutations of attribute values tested is much smaller than the total possible permutations.

Table IV shows an example adapted from [16] which can be used to calculate unimport_v in our case. The products in this example are automobile tires made by three manufacturers (the Brand attribute). The Tread Mileage attribute indicates how many miles the tire is expected to last. Three possible prices are considered. The Sidewall attribute indicates if there are white radial strips on the sides of the tires.

The Utility column represents the rating given to the tire. The 18 listed combinations in the table are designed to extract the rater's valuation of the individual attribute values. Note that this is only a third of the total possible combinations, but it is sufficient for determining the part-worths, which are found in the second table.

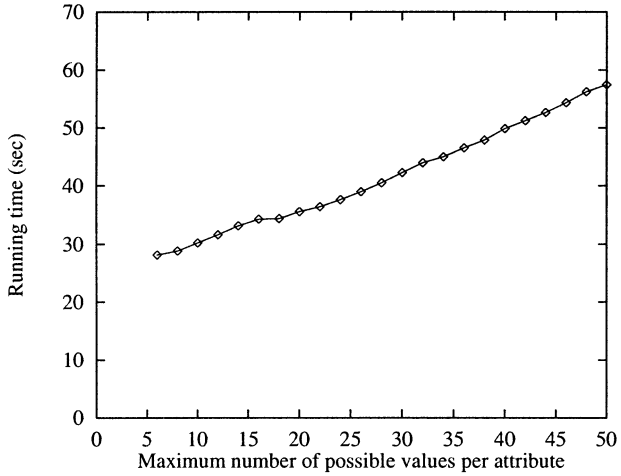

Fig. 10. Effect of number of attribute values on running time.

Subtracting the part-worths from a constant m yields a measure of unimportance. The unimportance becomes the weight on each edge in the catalog tree. If we set the constant $m = 10$ and let the popularity of each page equals 1, the initial tree in Fig. 2 thus has an average weighted unimportance $= 22.24$.

B. Changing the Design

Design is subjective. The catalog tree proposed by our algorithm may not suit the taste of the web designer. However, the output can be used as an initial design and modified by the designer. In this section, we give one approach on how the designer can modify the catalog tree T. More precisely, one way to reorganize the tree is to replace the attribute of a node u by an attribute a. This can be done easily by calling the procedure Rebuild (T, u, a) in Fig. 4. This routine not only replaces the attribute in u by a but also maintains the correctness of the topology of T. In other words, designers have the flexibility to choose any of the attributes to be used in any of the internal nodes.

V. Discussion and Conclusion

This paper introduces the problem of automatic online catalog construction which is formulated as a decision tree construction problem. A metric is proposed to evaluate the quality of a catalog organization. The metric is defined on the popularity of product pages and the unimportance of product attribute values. It is found that traditional decision tree algorithms do not give good catalog organizations under the proposed metric. An efficient greedy algorithm (GENCAT) is developed. Experiments show that GENCAT does product better catalog organizations. As part of the solution, we develop a methodology using a well-established marketing technique to obtain the values of unimportance of attribute values which are then used in the calculation of the metric.

GENCAT is intended to produce a correct and efficient organization but may of course be modified further by users. The procedure Rebuild () can readily be modified to do straightforward restructuring as decided by the site designer. We note that restructuring an online catalog or any other part of a website is not generally a trivial process as pages need to be reclassified. It is a simple matter to create semantically nonsensical results with pages being put where they do not belong or being made inaccessible.

Our approach does assume that the attributes for the information pages are provided. We have several justifications for this assumption. Many companies already use a database to store product information. Each field of a relational table corresponds to an attribute. Furthermore, XML seems destined to overtake HTML as the language of choice for web data. XML supports easy extraction of attributes from a page. There is already great support in the marketplace for XML (e.g., *http://www.xmledi.org*, *http://www.biztalk.org*).

For existing product pages which are written in HTML, we note that Sahuguet and Azavant [19] have already presented a wrapper-based solution for extracting complicated structure from HTML documents to convert them to XML.

GENCAT can process a large database of products but not in real-time. We are interested in finding faster alternatives that still produce an efficient organization. We also want to study the impact of the initial tree on the results and develop a better way to create the initial tree.

While multivalued attributes (e.g., the color of a striped shirt) can be processed by GENCAT, the impact of these attributes on the metrics has not been fully analyzed. Another potential problem is how to handle users who do not care about, or do not understand, an attribute used on a navigational page or one of the attribute's values.

A practical implementation of our approach should also consider attributes with a large domain. The standard solution of using ranges of values or subsets of the domain may not always be satisfactory. A consumer is more likely to be interested in a set of choices like, "less than $20," "less than $40," and "any price" instead of "less than $20," "$20–$40," and "over $40."

Even without further features, GENCAT provides a useful tool for website design. It helps select a good subset of attributes to use and supports revision of the results if the designer is dissatisfied with them or simply wants to consider alternatives.

Acknowledgment

The authors would like to thank V. Iyengar and C.-K. Yim for their useful comments.

References

[1] M. K. Buckland, M. H. Buttler, B. A. Norgard, and C. Plaunt, "OASIS: A front-end for prototyping catalog enhancements," *Library Hi Tech*, vol. 10, no. 4, pp. 7–22, 1993.

[2] C. R. Hildreth, "Online Catalog Design Models: Are we Moving in the Right Direction?," Council Library Res., 1995.

[3] B. A. Norgard, M. G. Berger, M. K. Buckland, and C. Plaunt, "The online catalog: From technical services to access services," *Ad. Librarianship*, vol. 17, no. 1, pp. 111–148, Jan. 1994.

[4] M. Spenke, C. Beilken, and T. Berlage, "FOCUS: The interactive table for product comparison and selection," in *ACM Symp. User Interface Software Technol.*, 1996, pp. 41–50.

[5] M. Stolze, "Soft navigation in product catalogs," in *in Proc. Res. Adv. Technol. Digital Libraries, Second Euro. Conf.*, 1998, pp. 385–396.

[6] J. Lee, H. S. Lee, and P. Wang, "analytical product selection using a highly-dense interface for online product catalogs," IBM Inst. Advan. Comm., 2001.

[7] L. Rosenfield and R. Morville, *Information Architecture for the World Wide Web*. Cambridge, MA: O'Reilly, 1998.

[8] M. Perkowitz and O. Etzioni, "Adaptive web sites: Automatically synthesizing web pages," in *AAAI*, 1998, pp. 727–732.
[9] S. J. Green, "Automated link generation: Can we do better than term repetition?," *Comput. Net. ISDN Syst.*, vol. 30, no. 1–7, pp. 75–87, 1998.
[10] J. Garofalakis, P. Kappos, and D. Mourloukos, "Web site optimization using page popularity," *IEEE Inter. Comp.*, vol. 3, no. 4, pp. 22–29, 1999.
[11] G. P. Zhang, "Neural networks for classification: A survey," *IEEE Trans. Syst., Man Cybern, C*, vol. 30, pp. 451–462, Nov. 2000.
[12] R. Rada, H. Mili, E. Bicknell, and M. Blettner, "Development and application of a metric on semantic nets," *IEEE Trans. Syst., Man Cybern.*, vol. 19, pp. 17–30, Jan./Feb. 1989.
[13] J. R. Quinlan, "Introduction of decision trees," *Machine Learning*, vol. 1, no. 1, pp. 81–106, 1986.
[14] J. Han and M. Kamber, *Data Mining: Concepts and Techniques*. San Francisco, CA: Morgan Kaufmann, 2000.
[15] J. P. Guilford, *Psychometric Methods*, 2nd ed. New York: McGraw-Hill, 1954.
[16] P. E. Green, D. S. Tull, and G. Albaum, *Research for Marketing Decisions*, 5th ed. Englewood Cliffs, NJ: Prentice Hall, 1988.
[17] J. R. Quinlan, *C4.5: Programs for Machine Learning*. San Mateo, CA: Morgan Kaufman, 1993.
[18] M. Farach and M. Thorup, "Optimal evolutionary tree comparison by sparse dynamic programming," in *Proc. 35th Annu. IEEE Symp. Foundations Comput. Sci.*, 1994, pp. 770–779.
[19] A. Sahuguet and F. Azavant, "Building light-weight wrappers for legacy web data-sources using w4f," in *Proc. VLDB*, 1999, pp. 730–733.

Wing-Kin Sung received the B.Sc. degree and Ph.D. degree in computer science from the University of Hong Kong, Hong Kong. He is currently an Assistant Professor with the Department of Computer Science, National University of Singapore (NUS). Prior to joining NUS, He worked as a Post-Doctoral Fellow at Yale University, New Haven, CT, and worked as a Senior Technology Officer in the E-Business Technology Institute (ETI) Hong Kong. His research interests include algorithms, combinatorial optimization, and computational biology.

David Yang received the B.A. degree in 1986, from Harvard University, Cambridge, MA, the M.S. degree in 1989, from the University of Illinois at Urbana-Champaign, and the Ph.D. degree in 1994 from Columbia University New York, NY in computer science.

He was visiting the CIS Department, University of Hong Kong, and the e-business Technology Institute, Philadephia, PA, on leave from a faculty position at St. Joseph's University at the time of this work. He is currently an assistant professor of computer science at California State-Hayward. His research interests are in the areas of computer science education and databases.

Siu-Ming Yiu received the B.Sc. degree in computer science from the Chinese University of Hong Kong, Hong Kong, the M.S. degree in computer and information science from Temple University, Philadephia, PA, and the Ph.D. degree in computer science from the University of Hong Kong, Hong Kong.

He is currently a Teaching Consultant with the Department of Computer Science and Information Systems of the University of Hong Kong. His research interests include computational biology and data mining.

David W. Cheung received the B.Sc. degree in mathematics from the Chinese University of Hong Kong. He also received the M.Sc. and Ph.D. degrees in computer science from Simon Fraser University, Barnaby, BC, Canada, in 1985 and 1989, respectively.

From 1989 to 1993, he was with Bell Northern Research, Ottawa, Canada, where he was a senior member of scientific staff. Since 1994, he has been a faculty member of the Department of Computer Science and Information Systems, The University of Hong Kong, Hong Kong. He is also the Director of the Center for E-Commerce Infrastructure Development at HKU. His research interests include database, data mining, XML technology and e-commerce standardization.

Dr. Cheung was the Program Committee Chairman of the Fifth Pacific-Asia Conference on Knowledge Discovery and Data Mining (PAKDD01) and an Industry Track Chair of VLDB 2002. He is also the program chair of the Hong Kong International Computer Conference 2003 (HKICC2003) and is the steering committee chair of the Freebxml.org, which promotes ebXML open source projects. He was a recipient of the HKU Outstanding Researcher Award in 1998.

Wai-Shing Ho received the B.Eng. degree in computer engineering from the University of Hong Kong in 1999. He is pursuing the Ph.D. degree with the Department of Computer Science of the University of Hong Kong.

His research interests include XML indexing, XML query optimization, XML storage, and data mining.

Tak-Wah Lam received the M.S. and Ph.D. degrees in computer science from the University of Washington, Seattle.

He is now an Associate Professor with the Department of Computer Science, the University of Hong Kong. His research interests include algorithms, computational biology, parallel computation, and data mining.