Classifying Elements for XML Query Transformation

© Ke Geng

University of Auckland, New Zealand
ke@cs.auckland.ac.nz

Abstract

Research into XML query transformation has become important with the increased use of XML. Currently the research into XML query transformation concentrates on transformation based on structure of the XML document. In this paper, we will introduce our research which transforms queries using the classification of elements of XML documents. The experiments have shown that our method improves query execution dramatically.

1 Introduction

Query transformation [1] is an important branch of semantic query optimization [13]. Some research into XML query transformation has been carried out [5] [9] [4], but this research concentrates on transforming XML queries based on the structure of the XML document and there is little research into transforming XML queries based on the content of the data. This is because XML documents are too complex and the complexity limits the implementation of existing analysis technologies to glean information from the content of the XML document.

In this paper, we introduce our research into transforming XML queries based on the data in XML documents. The aim of our research is to improve query execution by transforming the queries based on the elements' classification. Elements in the queried XML document will be analyzed and their value distribution characteristics will be abstracted. Then classifications will be generated based on the elements' value distribution charactestics. With the information of the elements' classification, the input XML queries will be evaluated and transformed to a query, which will return the same results as the original query but can be executed faster. A series of experiments have been carried out and the results show that query execution can be improved dramatically with our method. Our early contributions are: First, we introduce a method to transform XML queries based on the content of the XML document [6]. Second, we introduce possible query transformations in our method [6]. Third, we present experimental results to examine the practicality of transforming XML queries based on element classification and study the influence of query transformation on query execution [7]. Based on these contributions, the novel contributions in this papaer are: First, we discuss the challenges of XML data analysis and the possible solution. Second, we propose a possible XML data analysis method. Previously, we assumed a technique existed that analyses the XML data and classifies the elements. In this paper , we propose that XMeans can be used to do the classification.

The remainder of this paper is structured as follows. Background information is introduced in section 2. Our query transformation method is introduced in section 3. Section 4 introduces the query transformations in our method. In section 5, we introduce the experiments and analyze the experimental results. In section 6, we discuss related work. Section 7 draws conclusions and discusses the problems that we are working on.

Proceedings of the Spring Young Researcher's Colloquium on Database and Information Systems, Moscow, Russia, 2006

2 Background

In this section, we discuss the existing semantic query optimization methods in XML and the challenges that arise in XML analysis.

XML semantic query optimization can be classified into two main groups: optimization based on the structure of the document, and optimization based on the content of the document. Optimization based on the structure of the document uses information that is found in the document's schema to optimize XML query processing. In [9], an ordered graph schema is introduced. Input queries are evaluated against the schema and query conditions that will not influence the final result are eliminated. Another implementation of optimization based on structure is outlined in [5], which checks target XML documents against the schema and skips unnecessary computations. The authors in [11] introduce methods to transform XPath queries based on unique location path constraints abstracted from the XML Schema.

Optimizations based on the content use the characteristics of values of elements to improve XML query execution. Currently most of this kind of research concentrates on statistical information. An example of this kind of optimization is implemented in Lore [12], which uses statistical information of elements to improve query execution by rearranging the sub query execution order. Pattern trees [2] are used in another important implementation of query optimization based on statistics.

How to analyse XML documents and abstract element characteristics is an obstacle to transforming queries based on the content of XML. This obstacle arises because XML is very flexible and complex, limiting the existing XML analysis:

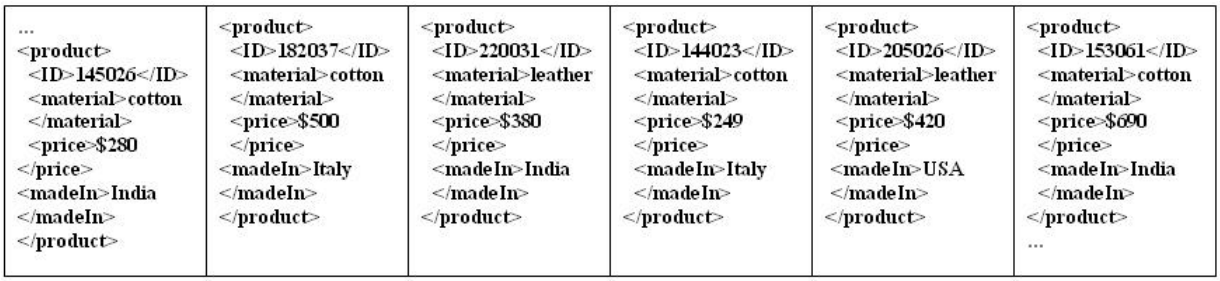

Figure 1: XML document store.xml

age	totalPay	importance	area
18-29	100-5000	1	1
30-39	5001-10000	1	2
40-49	10001-15000	2	2
50-59	15001-20000	2	2
60-79	20001-30000	3	3

Figure 2: The values of inserted elements

1. the structure of XML documents is unpredictable and users can compose their XML documents using any arbitrary structure. Also, the same content can be expressed in various ways, such as "$< name > JohnGoldberg < /name >$" and "$< name >< firstName > John < /firstName >< lastName > Goldberg < /lastName >< /name >$".

2. the relationship between elements can be complex. The relationship can be siblings, parent-child or ancestor-descendant.

3. same name elements may appear within different parent nodes and be distributed at different levels in an XML document.

3 Methodology

Our method for transforming XML queries is to classify elements based on distribution characteristics of the values of elements and transform the queries based on the classifications. As we discussed in the previous section, the biggest obstacle is the flexibility of XML documents. However if we concentrate only on specific elements instead of the whole document, it is possible to abstract their value distribution characteristics, and related elements may be classified based on the distribution characteristics. Then query transformation may be carried out based on the classification.

Consider the XML file shown in Fig. 1. It may be difficult to classify the **product** elements based on the values of all their subelements. This problem can be overcome by choosing the most important elements for the classification. For example we may analyze two elements **ID** and **material**, and find that all products that are made of cotton have **ID**s smaller than 200000 and the **ID**s of products that are made of leather are greater than 200000. So we can classify the elements **product** to two groups: **product of cotton**, which have **ID**s smaller than 200000, and **product of leather**, which have **ID**s bigger than 200000. With this classification, a query "$document('store.xml')//product[ID > 200000 \, and \, material = `leather']$" can be transformed to "$document('store.xml')//product[material = `leather']$" and the transformed query will return same results as the original when executed against the data in Fig. 1.

The element classification results and the related information of elements, including both element value characteristics and the number of the elements of each group, are all useful in query transformation. With the classification information, queries will be evaluated and possible query transformations may be generated. If there are multiple transformations, the number of elements in each group may be used to choose the most efficient transformation.

4 Transform XML query with element classification

Element classification based XML query transformations can be classified into three categories: **Elimination**, which removes mutually exclusive query conditions; **Reduction**, which removes the semantically redundant query conditions; and **Introduction**, which introduces extra conditions to the original query conditions. Below, we will discuss these transformations in detail. In order to explain the transformations clearly, we will use "$Condition_M(e)$" to represent the result of evaluating query condition "M" on element "e" and $\sigma_{Con(M)}(E)$ to represent all the elements that satisfy query condition "M" when applied to the elements in E.

4.1 Elimination Transformation

Elimination is the operation that eliminates queries with mutually exclusive query conditions. Consider an XML document in which "all the managers are paid more than $5000", then the condition of query "document('record.xml')//person [position = 'manager' and $getPaid < 4500$] "[1] can be transformed to $False$.

Theorem 3.1 If two conditions in a query are mutually exclusive and the operator between them is "$\wedge$", the two conditions can be transformed to $False$, which can be illustrated as

$$Condition_{M \wedge N}(e) = \textbf{False}$$

$$\textbf{if } Condition_M(e) \wedge Condition_N(e) = \textbf{False}$$[2]

[1] We represent all queries in this paper in XPath.

[2] For more discussion and a proof about each transformation please refer to [6].

	Original query	Generated query
Query for Elimination	document('record.xml')//person[totalPay<5000 and age>35]	Null
Query for Reduction	document('record.xml')//person[totalPay< Val_1 and age< Val_2]	document('record.xml')//person[totalPay< Val_1]
Query for Introduction	document('record.xml')//person[totalPay< Val_1]	document('record.xml')//person[totalPay< Val_1 and AddCon *< Val_2]

*AddCon is the element name appears in the introduced condition

Figure 3: Queries for the experiments

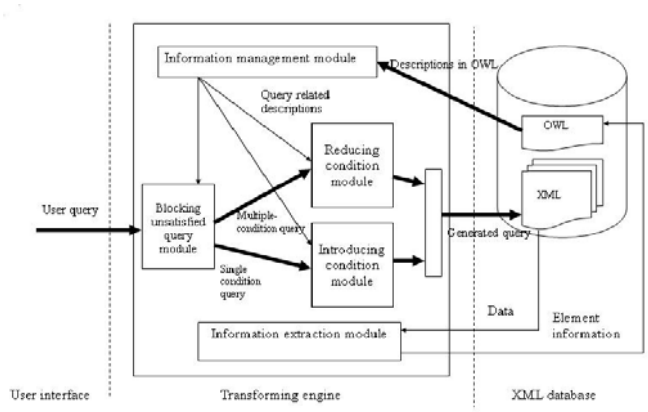

Figure 4: The structure of the transforming engine

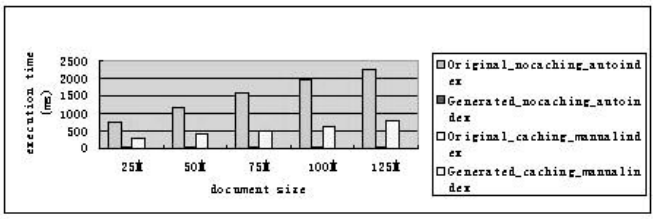

Figure 5: The results of elimination

The elimination transformation is a branch of the reduction transformation. We discuss it separately because this transformation may help us detect and block queries that return no result due to mutually exclusive conditions, as shown in the example above. These queries do not return any result and executing these queries wastes time.

4.2 Reduction Transformation

Reduction is the operation that eliminates redundant query conditions so this kind of transformation can only be applied to multiple-condition queries. An example of this transformation is that the original query "document('record.xml') // person [$position =' manager'$ and $salary > 5000$]" can be simplified to "$document('record.xml')//$ $person[position =' manager']$" if we know that all the managers are paid more than 5000.

Theorem 3.2 If all elements "e" that satisfy query condition "M" always satisfy condition "N" and the operator between the two query conditions is "$\wedge$", then the two query conditions can be simplified to condition "M". This is written below:

$$Condition_{M \wedge N}(e) \Rightarrow Condition_M(e)$$

$$\textbf{if } \forall e(e \in \sigma_{Con(M)}(E) \Rightarrow e \in \sigma_{Con(N)}(E))$$

Theorem 3.3 If all elements "e" that satisfy query condition "M" always satisfy condition "N" and the operator between the two query conditions is "$\vee$", then the two query conditions can be simplified to condition "N". This can be represented as:

$$Condition_{M \vee N}(e) \Rightarrow Condition_N(e)$$

$$\textbf{if } \forall e(e \in \sigma_{Con(M)}(E) \Rightarrow e \in \sigma_{Con(N)}(E))$$

Because the redundant query conditions are removed by the reduction transformation, both the amount of work of "computation" and "join" will be reduced and the performance of the query execution will be improved.

4.3 Introduction Transformation

Introduction is the transformation that adds a new condition, with the aim of reducing the query search space, compared with the original query condition. Consider the scenario where only people, whose rank is lower that 4, get paid less than 3000, the query " document('record.xml')// person [$salary < 3000$] " can be transformed to " document('record.xml')// person[$rank < 4$ and $salary < 3000$] ". The necessary conditions for this kind of transformation are :

- the domain of the original query condition must be a subset of the domain of the additional condition.
- an index is built on the additional condition-related element.
- the selectivity of the additional condition-related element is higher than that of the original condition-related element.

Theorem 3.4 If all elements "e" that satisfy query condition "M" always satisfy condition "N", then the query condition "M" can be transformed to condition "$M \wedge N$". The transformation can be illustrated as

$$Condition_M(e) \Rightarrow Condition_{M \wedge N}(e)$$

$$\textbf{if } \forall e(e \in \sigma_{Con(M)}(E) \Rightarrow e \in \sigma_{Con(N)}(E))$$

5 Experimentation

In this section, we present the experiments designed for our research. All the experiments are run on an HP workstation XW4200, which is equipped with an Intel Pentium 4 CPU 3.40GHZ and 2 GB memory.

The experimental data sets are built using an XML generator [8] based on the XMark benchmark [16]. We use the generator to build five XML documents with sizes of 20M, 40M, 60M, 80M and 100M. Then we insert four kinds of elements as child nodes to the "person" element. The inserted elements and values are listed in Fig. 2. From the figure, you can see that "person" elements can be classified into different groups based on the values of the inserted elements, such as a grouping of all the persons that are in their thirties and with an importance value of 1.

Three groups of queries are designed for the experiments and the formats are shown in Fig. 3. Because the condition of the original query will be transformed to $False$ using the Elimination Transformation, there is no query generated. For "reduction" and "introduction", we can get a series of queries with different selectivity by changing the values of Val_1 and Val_2. The introduced condition for "introduction" will be generated automatically according the descriptions of element classifications and for different situations the generated conditions are different.

A transforming engine was designed for the experiments, which includes the following modules:

1. **Information management module**, which is designed to manage the elements' classification descriptions used in query transformation;
2. **Information extraction module**, which is designed to extract information for each kind of element from the XML document;
3. **Blocking unsatisfied query module**, which deals with the Elimination transformation;
4. **Reducing condition module**, which deals with the Reduction transformation;
5. **Introducing condition module**, which deals with the Introduction transformation.

The structure of the engine is shown in Fig. 4[3].

We use the eXist XML database [3] in our experiments. Because eXist provides a query caching function to improve the speed of query execution and an index on elements can be built both automatically and manually, both the original queries and the generated queries are executed in four situations:

1. no query caching plus an automatically built index,
2. no query caching plus a manually built index,
3. with query caching plus an automatically built index
4. with query caching plus a manually built index.

Because of the space limitations, we only list the results in two situations: no query caching plus an automatically made index and with query caching plus a manually made index, and with two selectivities: 20% and 80%.

Experiment for Elimination

The results of these experiments are shown in Fig. 5. It can be seen that time spent on the original query execution is much longer than the time spent on the queries, which are generated by the transformation engine. This is because the conditions of the original queries are transformed to "$False$", and no queries are sent to the server and all the computations are finished on the client side. If we contrast this with the time spent on the original query execution, the time spent on query transformation can almost be ignored especially when transformation works together with query caching.

Experiment for Reduction

Results of "reduction" are shown in Fig. 6. From the Fig. it can be seen that because the workload on both computation and join are reduced, the performance of query execution can be improved by about 30% when the query is executed with the automatically built index. When the queries are executed with a manually built index, the performance can be improved around 20%. This phenomena becomes more obvious when the selectivity or the size of the XML document is increased.

Experiment for Introduction

From Fig. 7, it can be seen that the results of "introduction" are different from our prediction and the query execution time has increased instead of reduced. In fact, the bigger the selectivity or the size of the document, the more the execution time increases. After analyzing the query execution, we assume transformation "introduction" may only work on some query engines which execute queries by choosing a sub-query execution strategy. And for query engines, which execute multiple-condition queries by choosing the intersection of the results of each sub-query, the transformation "introduction" will increase the workload and make the query execution worse.

From the results of our experiments we can draw the conclusion that element classification based query transformation can improve the query execution in some situations.

6 Related work

Currently XML query optimization based on the content of the data mainly concentrates on element statistics. In the optimization mechanism in "Lore" [12], the database system abstracts all the information of nodes and edges, and stores them as several indexes in the database management system. All input queries are broken into a series of sub-operations and each sub-operation can be evaluated as part of the query. By creating evaluations for all the sub-operations and joining all the result aggregations together, the system can get the most effective execution order for the sub-operations, resulting in faster execution of the query.

In [17], an element classification method is introduced, which is based on the information of structure of XML documents. By analyzing the DTD [10] of an XML document, elements are classified based on their child nodes. For example: description "$<!ELEMENT\ person(name, email*)>$" will lead to a kind of classification of "person with email" and "person without email". With the classification, the query searching area will be reduced.

Authors of [4] introduce a heuristic-based algebraic

[3] For more discussion about the transforming engine please refer to [7].

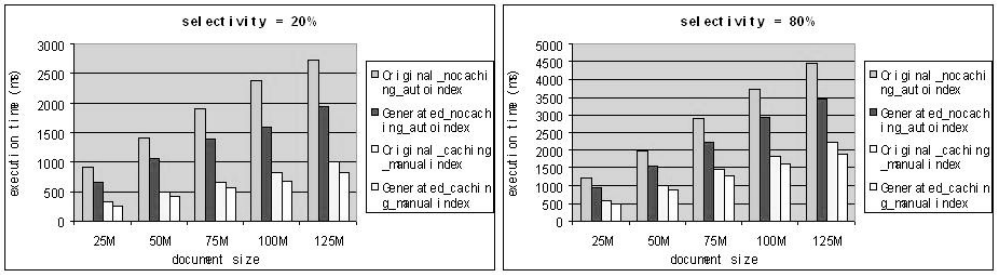

Figure 6: The performance of "reduction"

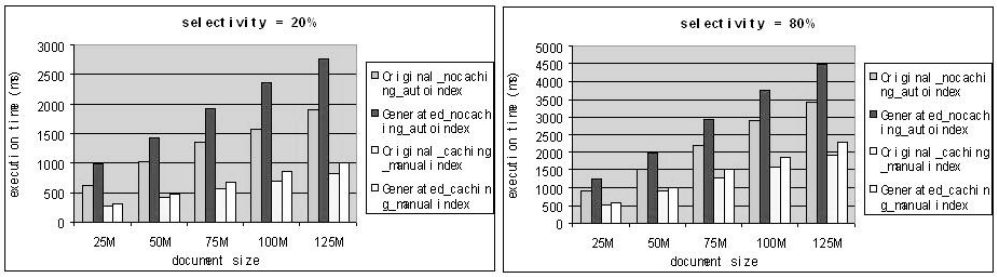

Figure 7: The performance of "introduction"

XML query transformation method. The method is based on a series of equivalences that are represented in PAT algebra expressions [15]. With information of the structure of an XML document, a set of deterministic algebraic transformation rules can be derived based on the PAT equivalences. Then an input query can be transformed to a more effective one with the transformation rules.
The research reported above improves the performance of XML query execution. However, none of this work is based on the distribution of the values of elements.

7 Conclusion and problems to be solved

In this paper, We discuss the possibility of transforming XML queries based on the content of data. We also discuss the possible transformations, which are carried out based on the content of data. A series of experiments are presented and the results are discussed. The experimental results show that transforming queries based on data can improve query execution time in some situations.
Currently the element classifications are edited manually. Now we are working on an algorithm, which analyzes the values of elements and abstracts distribution characteristics. As we discussed previously, XML documents are very flexible making analysis difficult. We have identified the following issues:

1. **the number of classifications** When an XML document is analyzed, it is very difficult for a user or database administrator to specify an accurate or suitable number of classifications based on the data distribution. Because of this, we chose the XMeans [14] algorithm to do the clustering, which is an adapted KMeans algorithm. The advantage of XMeans is that users input maximum and minimum number of classifications instead of specifying the exact number of classifications. XMeans will explore the best classification number automatically. Some experiments have shown that XMeans can satisfy our demands.

2. **missing element or missing values** For this problem, we propose to find an element which has similar characteristics to the element that has a missing subelement or missing value and attach the subelement or value found in the similar element. Consider the example in Fig. 8, with the people elements. One of the people has a missing value for "ID". The similar element belongs to department "computer" and has "ID" 01030, so the two elements will be classified into the same group.

3. **multiple same name elements under one parent element** For this situation, as a first step we will replicate and divide the element. For example: $< people >< /hobby >< /hobby >< /name >< /people >$ will be divided into two elements that has structure $< people >< /hobby >< /name >< /people >$. This solution will enable us to classify people based on individual hobbies but not on multiple hobbies.

4. **datatype handling** Values of elements/attributes in XML documents may belong to various datatypes. The data can be number, string or other types. The data can be very long, such as a paragraph of a book, or very short, such as an English character. Also the data can be classified as continuous or discrete data. Usually the data needs preprocessing before analysis. Currently our system can analyse number, short string or character. More functionality of datatype handling will be added to the whole system in the future.

5. **value distribution** When the classification is derived, the value distribution of each group will be

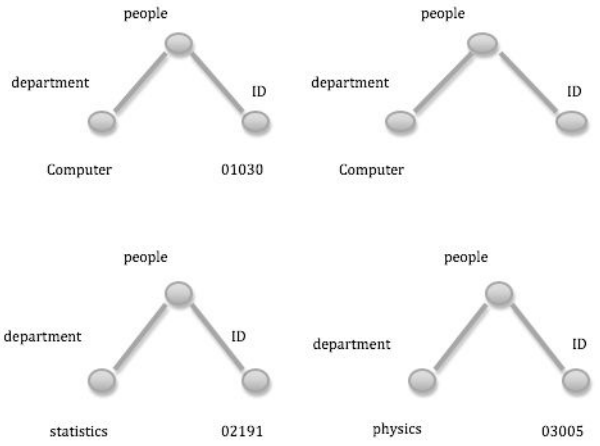

Figure 8: The XML document with missing value

used to optimize the input queries. The distributions can be classified into three types: isolated, overlapping and including. In our paper, we only demonstrate one situation where the value of each group is isolated. In our research, we found that the value distribution is very complex and the distributions influence the possibility of query optimization greatly. An example for overlapping is all lecturers have salary from 4000 to 6000 and all professors have salary from 5500 to 8000. An example for including is that the carparks for professors are distributed on level 2 and level 3 while the carparks for lectures are distributed on level 1, 2 and 3. How to handle the complex situation of value distribution and how to carry out query optimization with the complex data distribution will be studied in the future.

6. **complex elements** Even when we can specify several elements to analyze, some elements with complex structure may still be involved in the analysis.How to deal with complex objects is still an open question for us.

Our contributions to date have been a data generator to build data sets with specific characteristics to carry out accurate and targeted experiments, the classification of possible transformations and experiments that prove how well the transformations work. From here, we plan to build a tool that classifies the data. In order to do so, we will have to address the issues described above.

References

[1] R. Ahme, A. Lee, A. Witkowski, D. Das, H. Su, M. Zait, and Cruanes T. Cost-based query tranformation in oracle. Seoul, Korea, 2006. VLDB.

[2] A. Barta, M. P. Consens, and A. O. Mendelzon. Xml query optimization using path indexes. Paris, France, 2004. XIME-P.

[3] A. B. Chaudhri, A. Rashid, and R. Zicari. *XML Data Management: Native XML and XML-Enabled Database Systems*. Addison Wesley, 2003.

[4] D. Che, K. Aberer, and T. Özsu. Query optimization in xml structured-document databases. *the International Journal on Very Large Data Base*, 2006.

[5] M. Fernández and D. Suciu. Optimizing regular path expressions using graph schemas. Orlando, Florida, 1998. ICDE.

[6] K. Geng and G. Dobbie. Element classification based transformation of xml queries. Jakarta, Indonesia, 2007. IIWAS.

[7] K. Geng and G. Dobbie. Using data in the transformation of xml queries. Christchurch, New Zealand, 2008. NZCSRSC.

[8] K. Geng and Gillian Dobbie. An xml document generator for semantic query optimization experimentation. Yogyakarta, Indonesia, 2006. IIWAS.

[9] S. Groppe and S. Böttcher. Schema-based query optimization for xquery queries. Tallinn, Estonia, 2005. ADBIS.

[10] D. Hunter, K. Cagle, C. Dix, R. Kovack, J. Pinnock, and J. Rafter. *Beginning XML*. Wrox Press, Indianapolis, Indiana, second edition, 2003.

[11] D. X. T. Le, S. Bressan, D. Taniar, and W. Rahayu. Semantic xpath query transformation: Opportunities and performance. Bangkok Thailand, 2007. DASFAA.

[12] J. McHugh and J. Widom. Query optimization for xml. Edinburgh, Scotland, 1999. VLDB.

[13] H. H. Pang, H. J. Lu, and B. C. Ooi. An efficient semantic query optimization algorithm. Kobe, Japan, 1991. ICDE.

[14] D. Pelleg and A. Moore. X-means: Extending k-means with efficient estimation of number of clusters. San Francisco, 2000. ICML.

[15] A. Salminen and F. W. Tompa. Pat expressions: an algebra for text search. *Acta Linguistica Hungarica*, 1992.

[16] A. Schmidt, F. Waas, M. Kersten, M. J. Carey, I. Manolescu, and R. Busse. Xmark: A benchmark for xml data management. Hong Kong, China, 2002. VLDB.

[17] W. Sun, D. Liu, and W. Zhang. An efficient method for xml queries optimization based dtd abstraction and classification. Hangzhou, China, 2004. WCICA.