Massive Catalog Index based Search for e-Catalog Matching

Jae-won Lee, Taehee Lee, Sang-keun Lee, Ok-ran Jeong, and Sang-goo Lee

School of Computer Science and Engineering
Seoul National University
Seoul 151-742, Republic of Korea
{lyonking, thlee, liza183, orjeong, sglee}@europa.snu.ac.kr

Abstract

In e-commerce environment, business partners exchange product information in the form of e-catalogs. Since each business player uses his/her own classification and identification code for describing e-catalogs, it is not easy to match his/her product with the e-catalog requested by his/her partner. To date, these matches have been accomplished through the manual work of domain experts. Therefore, many attempts have been made to create a standard to solve this problem. This research focuses on improving the e-catalog information match process in order to integrate and exchange catalogs among various e-catalog systems. We suggest an index structure, MCI (Massive Catalog Index), which is optimized for searching large heterogeneous e-catalog databases. MCI is capable of searching vast catalog databases quickly as well as recommending precise e-catalog matches.

1. Introduction

In the e-commerce market, multiple companies join together and find business partners by exchanging information on products or services in the form of e-catalogs. E-commerce marketplaces, which function as an intermediate layer of business transactions, help companies to reduce the efforts needed to communicate with their partners by providing mappings of e-catalogs between buyers and suppliers [1]. In order to provide this service, e-commerce marketplaces use e-catalog match processes that integrate and map the customers’ e-catalogs

In almost all the current systems, this match is done manually by domain experts because e-catalogs from different sources have diverse formats and semantics. This paper suggests a method of reducing the efforts of the e-catalog match process. We provide an index structure, called MCI (Massive Catalog Index) and a search algorithm based on MCI, which is optimized for searching large heterogeneous e-catalogs. The algorithm enables us to semi-automate the e-catalog match process. By using our approach, it is possible to recommend e-catalogs similar to the catalog in the query by searching large catalog databases quickly even when their formats are different. Implementation of the MCI index structure enhances search accuracy and response time. Legacy search algorithms have failed to provide desired accuracy and response time because there are multiple properties coupled with inconsistency in e-catalog databases. An e-catalog mapping method requires a fast search algorithm because it has to extract similar product information by searching large catalog database efficiently. However, the search algorithms that have been suggested in other researches are not suitable for large catalog databases. This research proposes MCI for a fast and efficient search.

The proposed MCI structure uses a B+-tree index over a separate keyword index table. In addition, it is capable of calculating the relevance of catalog information and keywords with an optimized ranking function, thus making it possible to improve match accuracy.

The remainder of this paper is organized as follows. Section 2 reviews several related works. Section 3 compares both the legacy e-catalog information match model and the improved model. Section 4 suggests a search algorithm for large e-catalog databases, and describes the MCI structure which is the foundation of the improved process. Section 5 presents performance improvements over legacy search algorithms through experiments. Section 6 concludes the paper.
2. Related Work

A successful e-commerce marketplace has to integrate various hardware and software platforms and provide a common method for exchanging information. However, we are confronted with three levels of heterogeneity problem in the real-world which are content heterogeneity, product catalog structure heterogeneity, and document structure heterogeneity [2]. Content heterogeneity refers to the semantic heterogeneity of the exchanged information. People describe the same product in different ways. To solve this problem, many researchers have pursued the development of ontology merging. The Chimaera ontology-merging environment [3] considers a limited ontology structure in suggesting merging steps. PROMPT [4], which is a tool for semi-automatic ontology merging, identifies candidate pairs of matching terms—terms from different source ontology representing similar concepts. These methods make content integration possible but with a drawback of overheads in every domain as a result of building ontology.

We get product catalog structure heterogeneity because each catalog includes heterogeneous information, such as information on vendors and manufactures, catalog version, date, and identification number. Document structure heterogeneity means that in order to represent each of the catalogs, different languages are used for each case. e.g. XML, RDF, xCBL etc. These two heterogeneities occur because of the mismatch of syntactical structure. Various methods have been applied to solve this heterogeneity: including the XML model [5], RDF data model [6], and enhanced Naïve Bayes Classification [7].

The XML model has to parse and import large XML documents, which is a time consuming task. In addition, errors regarding syntaxes and semantics delay these processes. In the RDF model, document exchange is accomplished first in RDF forms. Then each system converts the document in XML forms. This causes transformation overheads and requires knowledge about the target system. In Naive Bayes Classification, it is essential to properly classify the products. Each categorization must be homogeneous and must be overlapped for proper classification. Furthermore, it does not use classification hierarchical information, which causes inaccuracies.

The XML model [5], RDF model [6] and Naive Bayes Classification [7] use Fully Match in which each system has the same attribute number in each schema. Furthermore, these methods cause some additional overheads such as parsing, transformation, and classification on every merging.

To solve these problems, MCI uses Partial Match which utilizes the common attributes in two catalog systems. Thus, heterogeneous systems which have common attributes are integrated without any overheads including transformation and classification. MCI enables users to modify attribute values for more accurate integration.

3. e-Catalog Match Process

3.1. Legacy e-Catalog Information Match Process

Figure 1 represents a legacy e-Catalog Information Match Process.

![Figure 1. Legacy e-Catalog Information Match Process](image-url)

**Step1.** A Requester requests a central repository administrator (CRA) to exchange e-Catalog documents.

**Step2.** After confirming the request, the CRA selects a target for exchange.

**Step3.** The CRA selects a candidate category of the requester’s catalog documents from the target’s several catalog categories.

**Step4.** In a selected category the CRA repeatedly searches target catalog documents by hands which are similar to requester’s catalog documents.

**Step5.** If a matching catalog document exists, the CRA modifies the requester’s document form into the target’s. He then reports the modified results and sends a success message to the requester.

**Step6.** If a matching catalog document does not exist, the requester’s catalog document must be new product information. The CRA modifies the requester’s catalog
document into the target’s and then reports the registration request to the target.

The biggest flaw of a legacy process is the inaccuracy and inefficiency of the manual works done by the experts. The inaccuracy occurs often when an expert selects a category from target categories for a requester’s catalog information. In other words, an expert has to select the proper category from a vast amount of categories to which a target belongs. Although this process is done by an expert, it is impossible to select a category with 100% accuracy at all times. Moreover, it is inefficient to select one from so many categories. A CRA (Catalog Repository Administrator) has to determine whether the catalog information similar to the requester’s catalog information exists or not. If it exists, the CRA must modify the attribute’s values. If not, the CRA must input the attribute’s values to register as new information. These processes which are completed manually, proved to be quite problematic when evaluating efficiency.

3.2. An Improved e-Catalog Information Match Process

The e-catalog match process is improved by semi-automating the e-catalog search and classification. Figure 2 depicts the improved process.

**Step1.** A Requester requests a central repository administrator (CRA) to exchange e-Catalog documents.

**Step2.** After the CRA confirms the request, he/she selects a target for exchange.

**Step3.** The CRA inputs attribute’s values which exist both in the requester’s catalog document and target’s catalog documents.

**Step4.** Using Auto-Classifier to select the category of requester’s catalog information, the CRA selects a category among the recommended categories. (The number of recommended categories is fewer than that of the original categories.)

**Step5.** The CRA executes a simple search with proper category information and attribute’s values which exist commonly on both sides.

**Step6.** If a matching catalog document exists, the CRA modifies the requester’s catalog document into the target’s document. The CRA then reports the matched results and sends a success message to the requester.

**Step7.** If a matching catalog document does not exist, the requester’s catalog document is new product information. The CRA modifies the requester’s catalog document into the target’s, then reports the registration request to the target.

The Auto-Classifier makes it possible for users of the improved match process to solve the problems of selecting a wrong category. It recommends a proper category to the user when common product attribute’s values are input, and it is implemented by modifying Naive Bayesian Classifier. However, this topic is beyond the scope of this paper, so it is not further discussed here. The strength of the improved match process is in the automation aspect of mapping e-catalogs by recommending n e-catalogs which are most similar to the e-catalog in query. Therefore, users can enhance the accuracy and efficiency of the match process by checking only the recommended e-catalogs. Furthermore, the semi-automated mapping method makes it possible for a user to refer to the recommended e-catalogs to fill out the unknown values of attributes.

In practice, we applied this process to iMarket Korea, one of the MROs [8]. When the legacy process was applied, it took about 16 hours to exchange 1000 catalog documents. After applying the improved process, the required time decreased to 4 hours. Consequently, by using the improved process, we can expect to cut the exchange time down to about 25% of the legacy process.

4. MCI (Massive Catalog Index)

When we do more general kinds of keyword searches, several problems have to be solved. First, we focus on the case where the keyword in the query matches only a sub-string of an attribute value. In this case SQL will
have clauses with substring predicates such as
WHERE COLUMN LIKE ‘%keyword%’. But traditional B+-tree indexes cannot be used for index search to exploit such predicates. If a user has clauses such as WHERE COLUMN LIKE ‘keyword%’ to use indexes, this produces incorrect result sets. Second, we provide a ranking algorithm in order to provide more precise result sets. Otherwise there is no way to know what users want.

There are three important requirements for MCI to solve the problems stated above. First, it should be possible to search product information regardless of whether product information is divided into many tables or not (e.g. common attribute table, individual attribute table). Second, MCI should provide a method of searching for the most similar catalog documents regardless of whether complete information is provided or not. The last requirement is that MCI should provide a fast search method and ranked results.

This section discusses the definitions of e-Catalog terms and suggests an MCI (Massive Catalog Index) structure.

4.1. MCI Creation Algorithm

MCI is composed of MCI_T and MCI_C table. To create MCI, the user has to pre-compute the values which are used for searching the e-Catalog. Figure 3 shows the schema of each table.

- **MCI_T** = (Prod_id, Attribute, Token, Frequency, Cat_length)
  
  Prod_id: Product identification code
  Attribute: Attribute name of catalog
  Token: Token of parsed value of catalog
  Frequency: Token frequency in the catalog
  Cat_length: The length of the catalog (the length of a row for a product)

- **MCI_C** = (Cat_count, Avg_cat_length, S)
  
  Cat_count: Total count of catalogs whose information stored in MCI tables
  Avg_cat_length: Average catalog length,
  S : empirical constant

**Figure. 3. MCI Table Schema**

When a user wants to construct an MCI_T and MCI_C table, he first inserts all pairs which are an attribute name and the parsed token of the value with the product identification code, frequency and catalog length into the MCI_T. After constructing the MCI_T table, the user computes Cat_count and Avg_cat_length of each attribute and inserts them into the MCI_C. Thus, MCI is a kind of an inverted file and makes it possible to search catalogs fast.

**Table 1. MCI_T and MCI_C Construction**

<table>
<thead>
<tr>
<th>Input : A Catalog Table for Parsing and Updating</th>
<th>Output : MCI_T and MCI_C Tables</th>
</tr>
</thead>
<tbody>
<tr>
<td>// Constructing MCI_T Table</td>
<td></td>
</tr>
<tr>
<td>(1) Set MCI_T to be empty</td>
<td></td>
</tr>
<tr>
<td>(2) for each e-Catalog table {</td>
<td></td>
</tr>
<tr>
<td>(3)   Scan each product id which is Prod_id {</td>
<td></td>
</tr>
<tr>
<td>(4)     Compute Cat_length of each Prod_id {</td>
<td></td>
</tr>
<tr>
<td>(5)      For Attribute and Value pair of each Prod_id {</td>
<td></td>
</tr>
<tr>
<td>(6)          Parse values which make each parsed Token Ti</td>
<td></td>
</tr>
<tr>
<td>(7)      Insert(Prod_id, Attribute, Ti, 0, Cat_length) to</td>
<td></td>
</tr>
<tr>
<td>MCI_T</td>
<td></td>
</tr>
<tr>
<td>(8)    }</td>
<td></td>
</tr>
<tr>
<td>(9)    }</td>
<td></td>
</tr>
<tr>
<td>(10)}</td>
<td></td>
</tr>
<tr>
<td>// Constructing MCI_C Table</td>
<td></td>
</tr>
<tr>
<td>(1) Compute Cat_count which means the number of distinct product</td>
<td></td>
</tr>
<tr>
<td>(2) Compute Avg_cat_length</td>
<td></td>
</tr>
<tr>
<td>(3) Insert(Cat_count, Avg_cat_length, s) where s is constant</td>
<td></td>
</tr>
</tbody>
</table>

**Example**

Tables 2 and 3 are examples of e-Catalog database. Table 2 has the information about common attributes while the values of individual attribute are in Table 3. Table 4 is an example of an MCI_T table constructed from Tables 2 and 3. Each token is obtained from Prod_name, Company, Attribute, and Value by parsing them with the dictionary. Table 5 is an example of an MCI_C table constructed from Table 4. After constructing the MCI tables, the user should create B+tree indexes on every attribute.

**Table 2. Example of Common Attribute Relation**

<table>
<thead>
<tr>
<th>Prod_id</th>
<th>Class code</th>
<th>Prod_name</th>
<th>Company</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>100</td>
<td>SyncMaster</td>
<td>Samsung</td>
</tr>
<tr>
<td>2</td>
<td>101</td>
<td>VAIOX505</td>
<td>Sony</td>
</tr>
<tr>
<td>3</td>
<td>102</td>
<td>XNOTE</td>
<td>LG</td>
</tr>
</tbody>
</table>
Table 3. Example of Individual Attribute Relation

<table>
<thead>
<tr>
<th>Prod_id</th>
<th>Class_code</th>
<th>Attribute</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>100</td>
<td>Volt</td>
<td>220V</td>
</tr>
<tr>
<td>2</td>
<td>101</td>
<td>Memory</td>
<td>256MB</td>
</tr>
<tr>
<td>3</td>
<td>103</td>
<td>HDD</td>
<td>505MB</td>
</tr>
</tbody>
</table>

Table 4. Example of MCI_T Table

<table>
<thead>
<tr>
<th>Prod_id</th>
<th>Attribute</th>
<th>Token (Ti)</th>
<th>Frequency</th>
<th>Cat length</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Prod_name</td>
<td>Sync</td>
<td>1</td>
<td>33</td>
</tr>
<tr>
<td>1</td>
<td>Prod_name</td>
<td>Master</td>
<td>1</td>
<td>33</td>
</tr>
<tr>
<td>1</td>
<td>Company</td>
<td>Samsung</td>
<td>1</td>
<td>33</td>
</tr>
<tr>
<td>1</td>
<td>Volt</td>
<td>220</td>
<td>1</td>
<td>33</td>
</tr>
<tr>
<td>1</td>
<td>Volt</td>
<td>V</td>
<td>1</td>
<td>33</td>
</tr>
<tr>
<td>1</td>
<td>Class_code</td>
<td>100</td>
<td>1</td>
<td>33</td>
</tr>
<tr>
<td>2</td>
<td>Prod_name</td>
<td>VAIO</td>
<td>1</td>
<td>33</td>
</tr>
<tr>
<td>2</td>
<td>Prod_name</td>
<td>505</td>
<td>1</td>
<td>31</td>
</tr>
<tr>
<td>2</td>
<td>Company</td>
<td>Sony</td>
<td>1</td>
<td>31</td>
</tr>
<tr>
<td>2</td>
<td>Memory</td>
<td>256</td>
<td>1</td>
<td>31</td>
</tr>
<tr>
<td>2</td>
<td>Memory</td>
<td>MB</td>
<td>1</td>
<td>31</td>
</tr>
<tr>
<td>2</td>
<td>HDD</td>
<td>20</td>
<td>1</td>
<td>31</td>
</tr>
<tr>
<td>2</td>
<td>HDD</td>
<td>Gb</td>
<td>1</td>
<td>31</td>
</tr>
<tr>
<td>2</td>
<td>Prod_name</td>
<td>X</td>
<td>1</td>
<td>31</td>
</tr>
<tr>
<td>2</td>
<td>Class_code</td>
<td>101</td>
<td>1</td>
<td>31</td>
</tr>
<tr>
<td>3</td>
<td>HDD</td>
<td>505</td>
<td>1</td>
<td>23</td>
</tr>
<tr>
<td>3</td>
<td>HDD</td>
<td>MB</td>
<td>1</td>
<td>23</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Cat_count</th>
<th>Avg_cat_length</th>
<th>S</th>
</tr>
</thead>
<tbody>
<tr>
<td>3</td>
<td>29</td>
<td>0.7</td>
</tr>
</tbody>
</table>

4.2. Search using MCI and Ranking Function

The purpose of search using MCI is to get top-k similar product identification codes matched to the keywords. E-Catalog documents are composed of attribute-value pairs. The parser divides the input values into many tokens and then selects the tuples from the MCI_T table where the values of the token attributes are matched to these tokens. Thus the MCI provides a sub-string match search method by parsing the keywords.

Then, the user can choose the top-k products which are ordered by the summation of the relevance scores of tokens.

Example

We assume that a user inputs a keyword such as ‘X505’. In this case, SQL will have a clause with substring predicate such as WHERE COLUMN LIKE ‘%X505%’. This clause can not use the traditional B+tree index and has to know the name of column. In MCI, a keyword (X505) is parsed into ‘X’ and ‘505’. Then tuples are selected from the MCI_T table. These tuples are matched to the given tokens.

The SQL Query for ‘X505’ is as follows:

```
SELECT DISTINCT Prod_id FROM MCI_T WHERE Token(Ti) = 'X' OR Token(Ti) = '505'
```

Finally, these results are ranked with the MCI_C table values using a ranking function. Then the user chooses the top-k products which are ordered by the ranking score.

The ranking function of the search using MCI is based on Document Scoring with Pivoted Normalization Weighting [9].
where
\( t_f \): the token's frequency in catalog
\( q_{tf} \): the token's frequency in keyword
\( N \): the total number of catalogs in the e-Catalog Database
\( d_f \): the number of catalog that contain the term
\( d_l \): the catalog length (in bytes), and
\( a_v d_l \): the average catalog length
\( s \): empirical constant (0.0 ~ 1.0)

Figure 6. Document Scoring based on Pivoted Normalization Weighting

We evaluate the effect of constant \( s \) in an experiment by changing the value from 0.0 to 1.0.

5. Experiment and Result Analysis
5.1. Experiment Design

This section experimentally compares the search execution time and proper match accuracy of MCI, DBXplorer[10] and Oracle Text 9i [11][12]. For evaluation, we use the 500,000 e-Catalog dataset from the Korean public procurement service system. We conducted our experiments using Oracle 9i RDBMS on Athlon(tm) 64 Processor 3000+ with 1.50G of RAM. The Catalog match system using each method was implemented.

5.2. The Comparison of Execution Time

Five sets out of 495686 match requests were used to compare the search execution time of each method and each set was composed of 1000 match requests. This experiment consisted of two parts. The execution time of each method was measured to search the e-catalog documents using parsed keywords. Then, the execution time required for using only the keywords that are contained in the vocabulary dictionary was compared to that required for using the keywords that also include UOM and stop words (articles and prepositions etc.).

Figure 7 shows the result of the experiment using parsed keywords in the dictionary. It shows that MCI and DBXplorer performed better than Oracle Full-Text. It also shows that although MCI is a little slower than DBXplorer which is optimized to the keyword based search, the difference is very small.

Figure 8 shows the execution time of each method using all parsed keywords. In this case, DBXplorer is much slower than Oracle Text 9i and MCI, because it has a number of tuples which include the same keywords such as ‘mm’ which is a unit of measurement. If DBXplorer creates a query using a ‘mm’, the created query becomes long and inefficient. The query has a disjunction form of prefix and keyword (e.g. WHERE COL1 LIKE ‘th%mm%’ or COL1 LIKE ‘an%mm%’ or … ). MCI has the best performance and it provides 121% better performance than Oracle Text 9i and 2.5 times better than DBXplorer.

5.3. The Comparison of Match Accuracy
The proper match accuracy of MCI was compared with that of Oracle Text 9i. Since DBxplorer does not provide a ranking algorithm, it was exempted. Five sets were used, and each test set had 1000 match requests. We compared the match results of the system to the result of the hand-work of experts for accuracy comparison, assuming that the latter has 100% accuracy.

Figure 9. Match Accuracy using Vocabulary Dictionary

Figure 9 shows that the match by MCI has a better appropriate match accuracy than Oracle Text 9i. However, the accuracy of MCI is not sufficient to be used in real situations.

Figure 10. Match Accuracy using all Keywords

Figure 10 indicates that MCI has 1.47 times better match accuracy than Oracle Text 9i. The property of MCI ranking algorithm considers more components of e-catalog documents such as keyword frequency and catalog length than the Salton ranking algorithm [7] used in Oracle Text 9i. This allows the MCI methods to have better match accuracy. It is very important to consider the words which are not in the dictionary such as model number and UOM in searching a similar e-catalog with better performance. This experiment indicated that MCI is more appropriate than Oracle Text 9i in searches for similar e-catalog documents with high match accuracy.

5.4. The Effect of Empirical Constant S

The constant s within the ranking function is an empirical constant. The value is usually 0.2 [9]. The set for the experiments was composed of 1000 match requests. We compared the accuracy to evaluate the effect of constant s.

Figure 11. The effect of Constant

Figure 11 shows that the match accuracy is the highest when s is 0.4. The accuracy was found to increase according to the empirical constant s until it is less than 0.4. However, the accuracy decreases as the empirical constant S increases further. It is interesting to note that the curve of rank 3 is flatter than that of rank 1. This means that the empirical constant become more sensitive as rank n increases.

6. Conclusion

In e-commerce market places, efficient exchanges of e-Catalog documents must be accomplished to activate e-transactions. To date, matches between e-catalogs have been accomplished by manual works of domain experts. To solve these problems, we have presented two improvements. First, we improved the process of large catalog information matching by automatic recommendation procedures of similar e-catalogs. Second, we improved the search accuracy and response time by developing the MCI index structure. Furthermore, we increased match probabilities by using a customized ranking function.

We believe that e-transactions between business partners from heterogeneous industry sectors will be
facilitated with added effectiveness and efficiency in product information mapping and integration.

7. References


Acknowledgement
This work was supported by the Ministry of Information & Communications, Korea, under the Information Technology Research Center (ITRC) Support Program.