Storage Structures for Efficient Query Processing in a Stock Recommendation System

You-Min Ha
Department of Computer Science
Yonsei University, Korea
ymha@cs.yonsei.ac.kr

Sang-Wook Kim
College of Information and Communications
Hanyang University, Korea
wook@hanyang.ac.kr

Sanghyun Park
Department of Computer Science
Yonsei University, Korea
sanghyun@cs.yonsei.ac.kr

Seung-Hwan Lim
College of Information and Communications
Hanyang University, Korea
shlim@hanyang.ac.kr

Abstract

Rule discovery is an operation that uncovers useful rules from a given database. By using the rule discovery process in a stock database, we can recommend buying and selling points to stock investors. In this paper, we discuss storage structures for efficient processing of queries in a system that recommends stock investment types. First, we propose five storage structures for efficient recommending of stock investments. Next, we discuss their characteristics, advantages, and disadvantages. Then, we verify their performances by extensive experiments with real-life stock data. The results show that the histogram-based structure performs best in query processing and improves the performance of other ones in orders of magnitude.

1. Introduction

Time-series data is a list of changing values sampled at fixed time intervals [7]. It reflects the status changes of objects in nature and economy as time passes. In many applications, an element value in time-series data is significantly affected by its preceding values accumulated so far [4]. Thus, by analyzing past element values in time-series data, we can find the regularities and also form their model, thereby predicting the values to appear in the near future.

Stock price sequences are a typical example of time-series data [1, 3, 5]. Since the goal of stock investors is to earn high return, it would help investors achieve successful stock investments to recommend proper buying and selling points via analysis of the stock price sequences. Each stock investor has his/her own conditions for buying or selling stocks. To meet requirements of a variety of stock investors, it would be so useful to develop a system that automatically recommends stock items whose price changing patterns come to satisfy the conditions required by individual investors.

In our previous work [6], we developed a system that recommends investment types to stock investors by discovering useful rules from past changing patterns of stock prices in a database. We proposed a new method that discovers and stores only the rule heads rather than the whole rules in a rule discovery process [6]. This allows investors to impose various conditions on rule bodies flexibly, and also improves the performance of a rule discovery process by reducing the number of rules to be discovered. For efficient discovery and matching of rules, we proposed methods for discovering frequent patterns, constructing a frequent pattern base, and indexing those patterns. We also suggested a method that efficiently finds the rules matched to a query from a frequent pattern base, and proposed a method that recommends an investment type by using the rules.

In our stock investment system, there are a large number of stock investors, who issue queries on multiple stock items of interest. When a query is requested to process, stock prices related to the query are accessed from disk for recommending an investment type. This incurs a number of random disk accesses, thereby causing degradation of system performance. In this paper, we propose a variety of storage structures that reduce disk accesses and CPU computations, and then discuss their advantages and disadvantages. We also evaluate their performances via extensive experiments.

978-1-4244-2624-9/08/$25.00 ©2008 IEEE

2. Stock Investment Recommendation System

In this section, we explain the rule model and the query model proposed in reference [6].

2.1. Rule model

In this paper, we use the following form of a rule to express the trend of changing stock prices. Here, H and B denote a rule head and a rule body, respectively. This rule implies that B happens after time t since H has occurred.

$$H \quad \longrightarrow^{t} \quad B(s,c)$$

Next, we discuss (s,c) in the rule. A changing pattern can be formed as a rule head only when a sufficient number of stock sequences support the pattern. s defined in the following is called a *support*, which means how many times the pattern P corresponding to H appears in past stock sequences.

$$s(H) \quad = \quad \frac{\text{\# occurrences of patterns that match } H}{\substack{\text{\# occurrences of patterns whose length} \\ \text{is same as that of patterns that match } H}} \times 100$$

Also, for being formed as a rule, a set of sequences that satisfy the above support should show a similar tendency in the time range of the rule body. c defined below is called a $confidence$, which represents how many stock sequences matched to H satisfy the condition on B together.

$$c(H,B) = \frac{\substack{\text{\# occurrences of patterns that match } H \\ \text{and satisfy the conditions of } B}}{\text{\# occurrences of patterns that match } H} \times 100$$

Our approach discovers those rules whose support and confidence are both larger than predetermined thresholds during analyzing the past stock sequences. If a recent changing pattern of an investor's stock item of interest is matched to some H, it recommends an investment type by referring to its B. Possible investment types to be recommended are '*BUY*', '*SELL*', '*HOLD*', and '*NO RECOMMENDATION*'. They are decided by conditions for B, which are highly dependent on propensities of investors.

Figure 1 shows a simple example of a stock price change. In this stock data, pattern ⓐ occurs three times. After a time interval ⓑ since then, the price is shown to increase twice and is shown to decrease once. From this fact, if pattern ⓐ appears again, the system recommends '*BUY*' as an investment type for this stock because the price is likely to increase with probability of 0.66. In this example, we regarded a pattern occurring three times as frequent. However, the system regards a pattern frequent only when its support is more than a minimum threshold, thereby considering it to be a rule head [2].

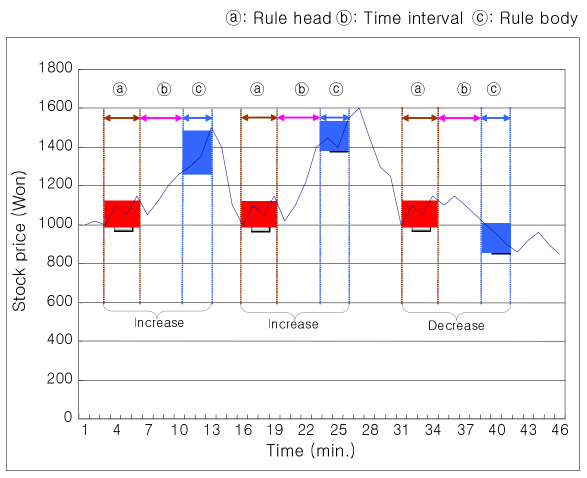

Figure 1. An example showing the rule model.

2.2. Query model

Definition 1 *Q, query form*
A query Q for requiring recommendation types is formulated in the following form.

$$Q = (I, T, BL, [\alpha, \beta], mC)$$

Each variable has the meaning as below.

- *I: the stock item of interest.*
- *T: the time interval between the end of a rule head and the beginning of a rule body.*
- *BL: the length of a rule body.*
- *$[\alpha, \beta]$: the range of average increase ratio for retaining the current stock item. Their actual meanings will be explained in Definition 2.*
- *mC: the minimum confidence, which is set to more than 0.5. The reason for this setting will be also explained in Definition 2.*

The system executes a query Q whenever the price of stock item I is changed. If the pattern obtained so far is matched to some frequent pattern, it recommends an appropriate investment type by referring to the corresponding rule bodies. The result of each query processing, $F(Q)$, has the following value.

Definition 2 *$F(Q)$, the result of processing a query Q*
For every case that supports the rule head, which is matched to Q, we calculate its average increase ratio r by comparing the end price of the rule head to the average price

of the rule body. As a result, $F(Q)$ for query $Q = (I, T, BL, [\alpha, \beta], mC)$, is defined as follows.

$$F(Q) = X, \quad X \in \{SELL, HOLD, BUY, NONE\},$$

where α and β are the minimum and maximum values for selecting 'HOLD', respectively. The recommendation type, X, is determined as follows.

- *$SELL$: If the ratio of cases satisfying ($r \leq \alpha$) is larger than mC.*
- *$HOLD$: If the ratio of cases satisfying ($\alpha < r < \beta$) is larger than mC.*
- *BUY: If the ratio of cases satisfying ($r \geq \beta$) is larger than mC.*
- *$NONE$: Otherwise (All three r's are less than mC)*

As stated in Definition 1, mC is required to set to at least 0.5 in order to avoid more than one investment type from being recommended. When $F(Q)$ has a value of X, we denote it as $F(Q) = X$ and read it as "X is recommended as a result of processing query Q."

The proposed approach enables stock investors to easily adapt the query model to suit their needs or application environments. Thus, it provides a fundamental framework for adaptive recommendation systems for stock investment. According to the experimental results, the system provides more than 70% satisfaction ratio in most cases [6].

3. Proposed Storage Structures

This section proposes storage structures that help process queries efficiently in a stock recommendation system.

3.1. OSM: Offset Storage Method

If a list of recent changing stock prices is matched to a frequent pattern, the system recommends a corresponding investment type for this stock item by analyzing their subsequent stock prices.

For each stock item, this method maintains a data file storing all the original changing prices and an index file storing pairs of <fPattern, listOfPositions>, where fPattern is a frequent pattern discovered and listOfPositions is a list of positions, each of which points to the location of an occurrence of the frequent pattern in the data file(see Figure 2). If frequent pattern 'ACD' appears in stock item I2, from its index file, the method finds all the positions at which the pattern occurred. Then, the method computes a final recommendation type by reading prices occurring after every pattern 'ACD' in original stock data files. We call this method OSM(offset storage method).

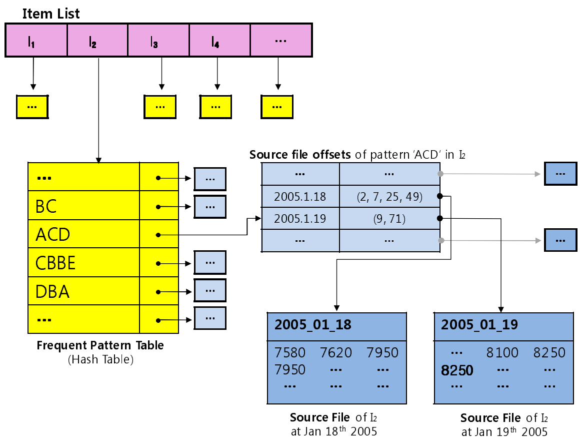

Figure 2. OSM: Offset Storage Method.

OSM is very simple and requires small extra storage space. However, for stock price prediction, this method has to read stock prices following all the occurrences of the frequent pattern that are scattered over an entire database. Thus, this method incurs a large number of random disk accesses during query processing.

3.2. VSM: Value Storage Method

A solution to the problem with OSM is that, if some values are highly likely to be accessed together, we store them in physically adjacent locations within disk. To the end, for each frequent pattern, we store a predetermined number of stock prices following every occurrence of the frequent pattern. This method sequentially accesses those prices stored together in disk. This method is expected to achieve the performance improvement in comparison with OSM.

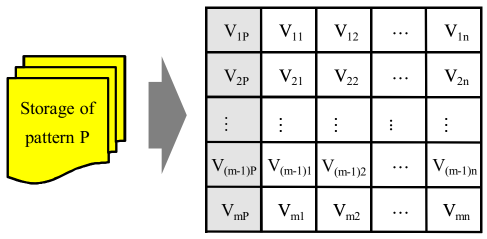
Storage of pattern P
V_{1P} V_{11} V_{12} $\cdots$ V_{1n}
V_{2P} V_{21} V_{22} $\cdots$ V_{2n}
$V_{(m-1)P}$ $V_{(m-1)1}$ $V_{(m-1)2}$ $\cdots$ $V_{(m-1)n}$
V_{mP} V_{m1} V_{m2} $\cdots$ V_{mn}

Figure 3. VSM: Value Storage Method.

Suppose that stock prices following those occurrences of pattern P are stored as in Figure 3. Here, m is the number of occurrences of pattern P, and n is the maximum of T+BL to be used in an application. V_{xy} is the y-th price from the last price of the x-th occurrence of pattern P, and V_{xp} is the last price of the x-the occurrence of pattern P. With this storage method, we compute the stock price increase rate r_x for the x-th occurrence of pattern P as follows.

$$r_x = \frac{\frac{\sum_{i=T+1}^{T+BL+1} V_{xi}}{bl} - V_{xP}}{V_{xP}}$$

We call this method VSM(value storage method). VSM can effectively avoid random disk accesses during query processing.

3.3. ADSM: Accumulated Difference Storage Method

For every occurrence supporting a frequent pattern, the average stock price in a rule body is computed during query processing. If we store accumulated differences of adjacent stock prices instead of original stock prices, we reduce computations of average stock prices in rule bodies.

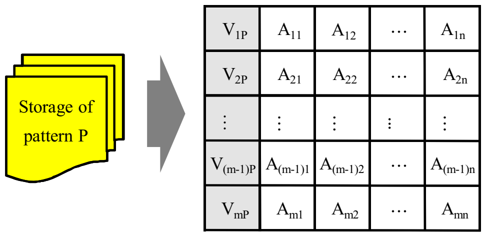
Storage of pattern P
V_{1P} A_{11} A_{12} ⋯ A_{1n}
V_{2P} A_{21} A_{22} ⋯ A_{2n}
⋮ ⋮ ⋮ ⋮ ⋮
$V_{(m-1)P}$ $A_{(m-1)1}$ $A_{(m-1)2}$ ⋯ $A_{(m-1)n}$
V_{mP} A_{m1} A_{m2} ⋯ A_{mn}

Figure 4. ADSM: Accumulated Difference Storage Method.

Suppose that the summations of stock prices are stored as in Figure 4. Here, m is the number of occurrences of pattern P, and n is the maximum of T+BL. Also, A_{xy} is computed as below.

$$A_{xy} = \sum_{i=1}^{y} V_{xi}$$

We compute the stock price increase rate r_x for the x-th occurrence of pattern P as follows.

$$r_x = \frac{\frac{A_{x(T+BL+1)} - A_{x(T+1)}}{bl} - V_{xP}}{V_{xP}}$$

We call this method ADSM(accumulated difference storage method). ADSM efficiently computes the summation of the (T+1)-th to the (T+BL+1)-th stock prices after each occurrence of pattern P. This makes the computation of the stock price increase rate reduced from O(m×BL) to O(x), and also makes disk accesses reduced because it accesses only the 3-rd column in the storage.

3.4. RSM: Ratio Storage Method

When value T is fixed in some applications, the stock price increase rate can be pre-computed and stored. With this method, the computation time is significantly reduced in query processing. We call this method RSM(ratio storage method).

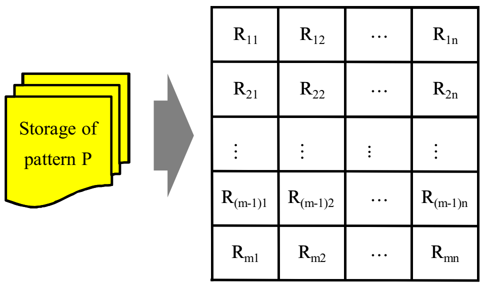
Storage of pattern P
R_{11} R_{12} ⋯ R_{1n}
R_{21} R_{22} ⋯ R_{2n}
⋮ ⋮ ⋮ ⋮
$R_{(m-1)1}$ $R_{(m-1)2}$ ⋯ $R_{(m-1)n}$
R_{m1} R_{m2} ⋯ R_{mn}

Figure 5. RSM: Ratio Storage Method.

Suppose that the stock price increase rate after the occurrences of pattern P is stored as in Figure 5. Here, m is the number of occurrences, and n is the maximum of (T+BL). V_{xy} is the stock price increase rate from the 1-st price to the y-th price compared to the last price of the x-th occurrence of pattern P. The stock price increase rate R_{xy} is computed as follows and then stored in a database.

$$R_{xy} = \frac{\frac{\sum_{i=1}^{y} V_{xi}}{y} - V_{xP}}{V_{xP}}$$

In case T=0, we can obtain the stock price increase rate easily by finding R_{xy} where y=BL. In query processing, the CPU overhead is considerably reduced owing to the pre-computation, and the disk access overhaed is also reduced because only one column is accessed from disk. RSM, however, is applicable only when value t is fixed.

3.5. HSM: Histogram Storage Method

The pre-computation of the stock price increase rate is possible if T and BL are fixed in applications. In the recommendation step, this stock price increase rate is compared to the range $[\alpha, \beta]$ in order to determine the recommendation type. If we store ratios of three cases: (1) the increase rate is smaller than α, (2) the increase rate is larger than β, (3) the increase rate is in between α and β, we can provide the recommendation type with this information. Thus, as in RSM if histograms are constructed by computing the increase rates in advance, query processing is performed with this information. Figure 6 shows this method.

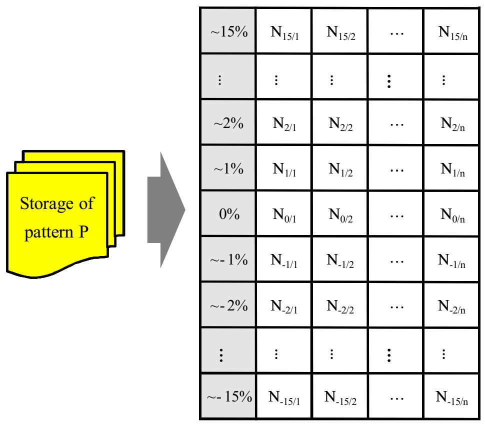

Figure 6. HSM: Histogram Storage Method.

The number of occurrences in each interval of the stock price increase rate is stored in a form of a histogram. In our case, the entire range(-15%∼15%) of the histogram was divided into 30 intervals in step of 1%. n is the maximum of T+BL. $N_{r/i}$ is the number of occurrences that have stock price increase rate between [r-1%, r%]. We call this method HSM(histogram storage method). With HSM, the size of a histogram does not increase even though the size of the original data size increases. Also, the size of an entire histogram is fixed, the computation is reduced from O(n) to O(1). As the same as RSM, it is applicable to situations where value T is fixed. However, even when value T varies, it can be applied by building more than one histogram for multiple values of T since its space overhead is very small.

4. Performance Evaluation

For performance evaluation, we first extracted three-month data from the Korean stock database KOSPI [8]. We set the maximum length of the stocks after the occurrence of frequent patterns as 20, and created 108 queries for each one of 905 stock items by combining the following parameters: (1) α: choose one among -0.003, -0.002, and -0.001, (2) β: choose one among 0.001, 0.002, and 0.003, (3) T: fix at 0, (4) bodyLen: choose one among 1, 3, and 5, (5) minimum confidence: choose one among 50%, 60%, 70% and 80%. All experiments were performed on the PC equipped with Intel Pentium 2.4GHz CPU, 1GB memory, and Window 2003 Server operating system.

We performed three kinds of experiments to evaluate the performance of the proposed storage structures. In the first experiment, we measured the disk space requirement of each storage structure. In the second experiment, we compared the storage structures in terms of the elapsed time for processing queries. In the final experiment, to evaluate the scalability of the storage structures, we measured their query processing performance while increasing the size of a data set.

Figure 7 shows the result of the first experiment. The X-axis represents each one of the five storage structures and the Y-axis does its disk space requirement in the unit of gigabyte(GB). The result reveals that OSM, VSM, ADSM, and RSM require the disk space of 0.11 GB, 1.2 GB, 1.2 GB, and 1.1 GB, respectively. That is, compared to OSM which stores the original data without any modification, VSM, ADSM, and RSM require about 12 times larger disk space. On the contrary, HSM requires less storage space than OSM. Therefore, by building separate HSMs for various starting points for prediction, we can easily overcome the major shortcoming of HSM that it can be used only for a specific starting point for prediction.

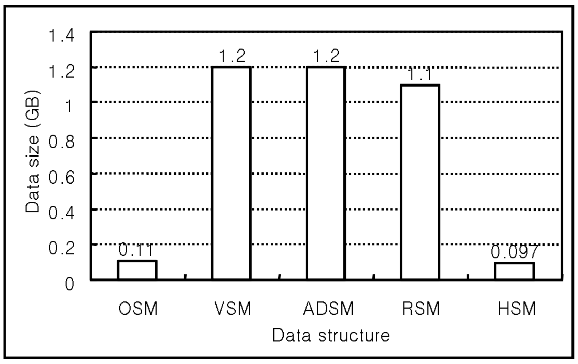

Figure 7. Disk space requirement of five storage structures.

Figure 8 shows the result of the second experiment. The X-axis represents each one of the five storage structures and the Y-axis does its query processing time in the unit of second. The result reveals that OSM, VSM, ADSM, RSM, and HSM required 3553.01 seconds, 3775.97 seconds, 3391.83 seconds, 2763.12 seconds, and 20.89 seconds, respectively, for processing all the queries. Note that, compared to OSM which stores the original data without any modification, VSM requires 1.6 times longer. This can be interpreted that, with the stock data stored sequentially, we have to read different files for different frequent patterns, which lowers the hit ratio of disk cache and subsequently increases the query processing time. The time of ADSM for query processing is about 1.05 times and about 1.11 times less than those of OSM and VSM, respectively. The time of RSM for query processing is about 1.29 times, about 1.37 times, and about 1.23 times less than those of OSM, VSM, and ADSM, respectively. In case of HSM, its query processing time is about 170.08 times, about 180.75 times, about 162.37 times, and 132.27 times less than those of OSM, VSM, ADSM, and RSM, respectively.

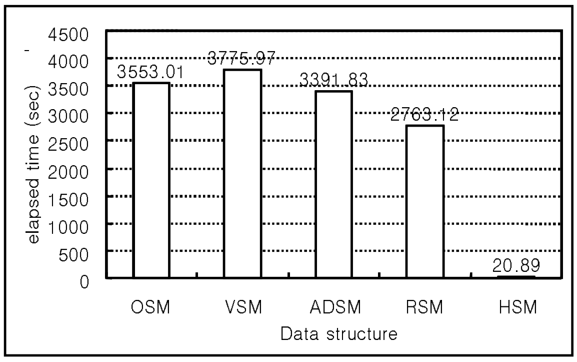

Figure 8. Elapsed time of five storage structures for query processing.

In the final experiment, to evaluate the scalability of the storage structures, we measured their query processing time while duplicating the original data set to increase its size two times, three times, and four times. Figure 9 shows the result of the final experiment. The X-axis represents the ratio of the size of the data set being used to that of the original data set, and the Y-axis does the query processing time in the unit of second. From the figure, we can see that the query processing time of the four storage structures other than HSM increases linearly to the size of data set. This is because the increase of the size of data set causes the increase of disk space requirement of the four storage structures which subsequently incurs the increase of their query processing time. On the contrary, the query processing time of HSM is almost constant regardless of the increase of the size of data sets. This is because the size of HSM is not affected by the size of data set.

In summary of the experimental results, HSM utilizing histograms shows the best performance in terms of disk space requirement, query processing time, and scalability.

5. Conclusions

This paper addresses storage structures that make queries efficiently processed in a realtime stock recommendation system. The proposed storage structures were devised to improve the performance of query processing by reducing the number of disk accesses as well as CPU computations. They have different characteristics in the accuracy, space, and processing time. Performance evaluation with experiments was performed for comparing those proposed storage structures. The results reveal that HSM shows best performance and also maintains performance not that changeable

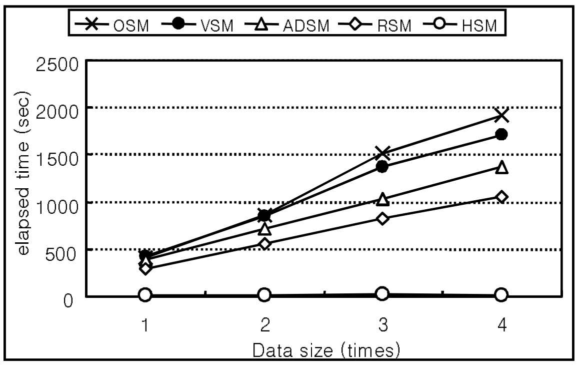

Figure 9. Query processing time with increasing data set sizes.

with the size of the original data file.

6. Acknowledgment

This work was partially supported by the Korea Research Foundation Grant funded by the Korean Government (KRF-2005-206-D00015), the ITRC support program supervised by the IITA (IITA-2008-C1090-0801-0040) and the Small and Medium Business Administration (SMBA) of South Korea (Project No. S5107A13101).

References

[1] R. Agrawal et al., “Fast Similarity Search in the Presence of Noise, Scaling, and Translation in Time-Series Databases,” In *Proc. VLDB*, pp. 490-501, 1995.

[2] R. Agrawal and R. Srikant, “Fast Algorithms for Mining Association Rules,” In *Proc. VLDB*, pp. 487-499, 1994.

[3] T. Anderson, The Statistical Analysis of Time Series, *Wiley,* 1971.

[4] P. Bloomfield, Fourier Analysis of Time Series, *Wiley,* 2000.

[5] C. Faloutsos, M. Ranganathan and Y. Manolopoulos, “Fast Subsequence Matching in Time-Series Databases,” In *Proc. ACM SIGMOD*, pp. 419-429, 1994.

[6] Y. M. Ha, S. W. Kim, and S. Park, “Rule Discovery and Matching in Stock Databases,” In *Proc. IEEE COMPSAC*, 2008.

[7] S. W. Kim, S. Park and W. W. Chu, “An Index-Based Approach for Similarity Search Supporting Time Warping in Large Sequence Databases,” In *Proc. IEEE ICDE*, pp. 607-614, 2001.

[8] Koscom Data Mall, *http://datamall.koscom.co.kr,* 2005.