Mining Non-overlapping Repetitive Sequential Patterns by Improving GSP Algorithm

Yongshun Gong[1], Xiangjun Dong [*, 1], Xiqing Han [2] and Ruilian Hou[1]

[1]*School of Information, Qilu University of Technology, Shandong, Jinan, 250353, P.R. China*
[2]*Administration Office, Shandong Institute of Commerce and Technology, Shandong, Jinan, 250103, P.R. China*

Abstract: Repetitive sequential patterns (RSP) mining plays very important roles and has been widely studied in DNA or genome, but there are only a few relevant approaches focusing on mining RSP from sequence database. Taking sequence <*bcbcbcbca*> for example, traditional sequential pattern mining algorithms only consider that <*bc*> appears at one time when calculating the support of <*bc*>, regardless of at least 4 times that <*bc*> appears within this same data sequence. Accordingly, to catch much more interesting sequential patterns, repetitive property needs to be involved during the mining process. However, currently the most relevant RSP methods focus on DNA analysis considering that they cannot be used for recognizing repetitive patterns on events sequences. Therefore, we propose an approach to determine the number of times a sequence repeatedly makes an appearance in a certain data sequence. The support value of a sequence could be more than 100% as this sequence might repeat in one data sequence, therefore we proposed a strategy to ensure the support range of repetitive sequence still within [0,100%]. Finally, we proposed an efficient algorithm, called RptGSP, to discover such repetitive sequential patterns based on improving GSP Algorithm. The experimental results reveal that RptGSP can efficiently discover the repetitive patterns.

Keywords: GSP, Repetitive pattern, Sequence database.

1. INTRODUCTION

Since sequential patterns (SP) mining was first proposed in 1995 [1], some classical SP mining methods, such as PrefixSpan [4], FreeSpan [3], GSP [2], SPAM [6] and SPADE [5] have been utilized to improve SP mining efficiency. However, the above algorithms do not consider the repetitive sequential patterns (RSP) mining problem. RSP can capture repetitions of a pattern in different sequences as well as within a sequence as the same item(s) can occur more than once in a data sequence. For example, suppose a dataset contains two sequences as below: {<*bcbcbcbca*>; <*ca*>} and a given minimum support threshold *min_sup* =2. Traditional SP mining methods can only identify <*ca*> and ignore <*bc*> because the support of <*ca*> and <*bc*> is in the ratio of 2:1, although <*bc*> continuously occurs four times when the first data sequence is scanned. Then sequences with repetitive property, such as <*bc*>, are called repetitive sequential patterns (RSP). RSP mining approaches consider that a sequence/item might repeat many times in a data sequence, which can help analysts to capture more useful information. In fact, mining RSP is helpful for deeply understanding the relation of items in many applications, such as DNA periodic analysis, network attack detection, outlier pattern detection, and other application fields [7-15].

In order to mine RSP reasonably and efficiently, several problems need to be solved: (1) How to define the repetitive times that a sequence occurs in a data sequence. For example, how many times that <*bc*> is repeated in a data sequence <*ba*(*ac*)*bc*>. (2) How to calculate the support of repetitive sequential candidates. (3) Generally, traditional SP mining methods use [0,100%] to express the range of a pattern's support value. But if we take RSP into consideration, this support range would extend above 100% because a sequence may occur repeatedly in a data sequence. Thus, how to restrict a pattern's repetitive support still in [0,100%] is a problem that needs to be solved. (4)The last challenge is how to design an efficient method to discover such repetitive patterns. Most of the existing RSP algorithms focus on DNA analysis [8, 11] that cannot be used to identify repetitive patterns due to typical data characteristic. Although a few algorithms have been proposed for mining frequent episode or periodic patterns in a sequence dataset [7, 9, 10, 12-14, 16, 17], there is not a unified definition of RSP.

Hence, in order to address the above problems, we have designed some good solutions as follows.

(1) We proposed an approach to determine the number of times one sequence repeatedly makes an appearance in a certain data sequence;

(2) The support value of a sequence would be more than 100% because this sequence might repeat in one data sequence. Then we proposed a strategy to ensure the support range of repetitive sequence still within [0,100%].

*Address correspondence to this author at the School of Information, Qilu University of Technology, Shandong, Jinan, 250353, P.R China; Tel: +86-531-89631251; E-mail: d-xj@163.com

A preliminary version of this article has been published in 2014 International Conference on Machine Tool Technology and Mechatronics Engineering, 2097-2100

(3) We proposed an efficient algorithm, called RptGSP, to discover such repetitive patterns by improving a classic sequential pattern mining method GSP.

The rest of this paper is organized as follows. The related work is discussed in Section 2. Section 3 describes the solutions to these problems and our algorithm RptGSP, is followed by experimental results in Section 4. Section 5 concludes the paper and points out our future work.

2. RELATED WORK

Very limited publications focus on mining repetitive sequential patterns from sequence dataset. We discuss typical existing papers concerning RSP in this section. A study [7] proposed the concept of RSP that would allow items to be overlapped when calculating the repetitive supports. For example, Let *ds*=<*BDABDBAA*> be a data sequence, <*BDB*> occurs twice in *ds* at positions <1, 2, 5> and <5, 6, 7> respectively. [10, 12] The definition of RSP [10,12] is the same as that in given earlier [7]. Another study [10] proposed an efficient algorithm CRGSgrow to solve the problem of how to compress repetitive gapped sequential patterns. One of the studies [12] proposed a navigation pattern clustering approach based on closed repetitive gapped subsequences. But the above RSP definition is different with that in this paper which does not allow the overlap property.

The concept of gap weight in various subsequences was proposed [11], which introduces the gap definition in different events. This paper designed an efficient algorithm EWM to find repetitive subsequences with gap weight. It used different definitions of RSP [7] in which it does not distinguish between non-overlapping subsequences and overlapping ones. Another study [8] introduced a repetitive expansion conception which is applied in DNA replication. Zhang *et al.* discussed DNA sequences also focusing on discovering periodic patterns with gap requirement [9].

There are few approaches which focus on finding RSP just from a single data sequence [13, 16]. Mannila *et al.* proposed an approach of mining episode to catch frequent episodes within a sequence [13]. An episode is defined as a series of events occurring relatively close to one another. If an episode is a sub-sequence of the series of events occurring in the window, then it is also supported by the window. In a study [16], a sequence was separated into windows without overlapping. If a sequence occurs in at least a certain number of windows, then it is recognized as a frequent pattern. A priori property can be utilized with this definition by segmenting a pattern into windows and calculating the number of these windows in which a pattern frequently appears. However, patterns that span multiple windows would not be caught, and in some cases, a reasonable window width is hard to make sure of. Yang *et al.* researched on how to find asynchronous periodic patterns from time series data [17]. In this model, shifts in the appearance of patterns are permitted to filter out random noises. This paper also took a range of periods into consideration that are not same as mentioned in the literature [16], although there is still a restriction on the maximum length of a period.

A method was proposed [14] for identifying iterative patterns, which capture occurrences in the semantics of Message Sequence Chart/Live Sequence Chart, a standard in software modeling. Iterative pattern is known as a series of events which are repeated within and across sequences with different underlying target formalism and semantics. Different search space pruning strategies and mining algorithms are used to efficiently mine recurrent rules. Another study [15] used the definition of iterative patterns similar to one mentioned earlier[14]. It introduced a strategy to find generators of iterative patterns and investigated catching of iterative generators from program execution traces. Generators are the minimal members of an equivalence class, while closed patterns are the maximal members. An equivalence class in turn is a set of frequent patterns with the same support and corresponding pattern instances.

3. RPTGSP

3.1. Basic Definition

Let an *items* set be $I=\{i_1, i_2,..., i_n\}$, and an *itemset* is given as a subset of *I*. A *sequence* represents an ordered list of *itemsets*, which can be described as $< s_1, s_2,..., s_l >$, where $s_j \subseteq I$ $(1\leq j \leq l)$. s_j is also named an *element* of *sequence*, described as $(x_1, x_2,..., x_m)$, where x_k means one item, and $x_k \in I$ $(1\leq k \leq m)$. For simplicity, if an element only contains one item, the bracket could be omitted, *i.e.*, (x_1) can be written as x_1. An item of a sequence can appear at the most once in one element, but it can occur multiple times in various elements. *Length*(*s*) represents the length of sequence *s*, which is the total number of items in all elements in *s*. *Size*(*s*) is the size of *s*, which means the total number of elements in *s*. For example, assume a sequence *s*=<*a*(*bc*)*ae*> is comprised of 4 elements *a*, (*bc*), *a* and *e*; meanwhile, it is also comprised of 5 items *a*, *b*, *c a* and *e*. Thus *s* is a *4-size* and *5-length* sequence.

Sequence $s_\alpha=<\alpha_1,\alpha_2,\ldots,\alpha_n>$ is named a sub-sequence of sequence $s_\beta=< \beta_1, \beta_2,\ldots, \beta_m >$ and s_β is a super-sequence of s_α, described as $s_\alpha \subseteq s_\beta$, if there exists $1\leq j_1 < j_2<\ldots< j_n \leq m$ such as $\alpha_1 \subseteq \beta_{j1}$, $\alpha_2 \subseteq \beta_{j2},\ldots, \alpha_n \subseteq \beta_{jn}$. It also means that s_β contains s_α . For example, <*c*>, <*ac*> and < (*ab*) *d*> are sub-sequence of < (*ab*) *c d*>.

We use a set of tuples <*sid,ds*> to represent a sequence dataset *D*, where *ds* is the data sequence and *sid* is the id of *ds*. |*D*| is the number of tuples in *D*. The set of tuples containing sequence *s* is described as {<*s*>}. *s_count*(*α*) is the support count of *α*; it is the number of {<*α*>}, *i.e.*, $s\_count(\alpha)= |\{<\alpha>\}|=|\{<sid, ds>| <sid, ds> \in D \wedge (\alpha \subseteq ds)\}|$. The support of *α* is the ratio of *s_count*(*α*) to total number of tuples in *D*, *i.e.*, $s(\alpha)=s\_count(\alpha)/|D|$. If $s(\alpha)\geq min\_sup$, then *α* would be recognized as a frequent sequential pattern; if $s(\alpha)\geq min\_sup$, then we consider *α* as infrequent, where *min_sup* is a minimum support threshold given by users or experts. The task of sequential pattern mining is to discover the set of all sequential patterns.

3.2. Repetitive Containment

As a data sequence (*e.g.*, *ds* =<*bcbcbcbca*>) may contain a sequence (*e.g.*, *s* =<*bc*>) more than once without overlap between the repetitive sequences, the key problem we need to know is the position of *s* from the left side of *ds*. Accordingly, it is important to define the repetitive containment problem.

(1) *Definition 1*□*Left appearance end position*

Assume a data sequence *ds* = $<e_1e_2...e_n>$, and *s* as a sequence, where e_i is an element($1\leq i\leq n$). If $\exists i$ ($1<i\leq n$), s.t. $s\subseteq<e_1...e_i>\wedge s\not\subset<e_1...e_{i-1}>$, then *e* is named the *left appearance end position***,** described as $LAE(s,ds)=i$; if $s\subseteq<e_1>$ then $LAE(s,ds)=1$; if $s\not\subset ds$, then $LAE(s,ds)=0$. In particular, if *s* is a *1-size* sequence, such as <*e*> and < (*ab*)>, *s* is not repetitive; hence its support can be calculated per the traditional way of valuing support.

To compute the times *t* that *s* occurs in $ds=<e_1e_2...e_n>$, first we obtain the *left appearance end position i* of *s* in *ds* by *LAE*(*s*, *ds*). If $i>0$ then $t=t+1$. Secondly, let $ds=<d_{i+1}...d_n>$, repeat the above process until *LAE*(*s*, *ds*)=0. The following algorithm follows this pattern.

Algorithm 1: RptTime(*s*,*ds*)

```
Input: a data sequence $ds=<e_1e_2...e_n>$; a sequence $s$
Output: repetitive times
(1) Set $t$=0;
(2) $i= LAE(s, ds)$;
(3) while ($i\neq 0$) do {
(4) $ds=<e_{i+1}...e_n>$;
(5) $i= LAE(s,ds)$;
(6) t++;
(7) }
(8) return $t$;
```

Example 1. Given *s*=<*ab*>, ds_1=<*aa*(*ab*)*b*>, ds_2<*abababab*c>. *LAE*(*s*, ds_1)=3, *LAE*(*s*, ds_2)=2, *RptTimes* (*s*,ds_1)=1, *RptTimes* (*s*, ds_2)=4, *RptTimes* (<*aba*>,ds_2)=2.

3.3. RSP Support Calculation

According to the definition of support in traditional sequential pattern mining, if a data sequence *ds* contains a sequence *s*, then 1 will be added to the support count of *s* no matter how many times that *s* occurs in *ds*; as a result, the range of *s*(*s*) would be [0,100%]. But now, if we still calculate supports in the traditional way, the maximum of *s*(*s*) would be higher than 100% when taking repetitive property into consideration. This does not accord with the traditional support-confidence framework and would make users confused when setting the *min_sup*. Therefore, to ensure the repetitive support of *s*, *i.e.* the range of *s*(*s*) is still within [0,100%], we have designed a novel strategy to address this problem.

(1) *Definition 2*□*repetitive support count.*

The *repetitive support count* of sequence *s*, denoted as *rps_c* (*s*) describes the total number of times that one sequence *s* has made repeated appearances in a certain dataset *D*.

$$rps\_c(s) = \sum t(t = RptTimes(s, ds), \forall ds \in D) \qquad (1)$$

(2) *Definition 3*□*repetitive support.*

Assuming a sequence has the maximum support count in database *D*, then its support count will be denoted as *max_s_c*; assuming a sequence owns the maximum repetitive support count in *D*, its repetitive support count will be denoted as *max_rps_c*; and the *repetitive support* of a sequence *s***,** denoted as *rps*(*s*), is

$$rps(s) = \frac{rps\_c(s)}{|D|} * \frac{max\_s\_c}{max\_rps\_c} \qquad (2)$$

We restrict the support range of repetitive sequence still within [0,100%] by equation 2 that users or experts could conveniently set up the repetitive support threshold in the traditional method.

3.4. RptGSP Algorithm

We design an RSP mining algorithm by improving GSP, called RptGSP. GSP only gets each item's support count during its first data scanning, RptGSP can also obtain each item's repetitive support count so as to obtain *max_s_c* and *max_rps_c*. Subsequently, each item's repetitive support could be calculated by utilizing equation 2. For any sequence candidate *c*, RptGSP uses equation 1 to get *c*'s repetitive support count, *rps_c* (*c*), and then its repetitive support value, *rps*(*c*), would be obtained by equation 2. RptGSP mining process is shown in the following pseudocode.

Algorithm 2 : **RptGSP**

```
Input: Dataset $D$, $min\_sup$;
Output: $RSP$;
1 $C_1\leftarrow$First_scan($D$); // store all items and their information into $C_1$, including $s\_count$ and $rps\_c$.
2 Find $max\_s\_c$ and $max\_rps\_c$ in all items;
3 for each item $s$, calculate $rps(s)$ by equation 2;
4 $F_1\leftarrow\{<\{f\}>|f\in C_1, rps(f)\geq min\_sup\}$;
5 for ($k$=2; $F_{k-1}\neq\emptyset$; $k$++) do
6 $C_k\leftarrow$generation ($F_{k-1}$);
7 for each candidate $c\in C_k$
8 for each data sequence $ds\in D$
9 if ($ds$ contains $c$)
10 $c.rps\_c$=+RptTimes ($c$,$ds$);
11 endfor
12 calculate $rps(c)$ by equation 2;
14 endfor
15 $F_k\leftarrow\{c\in C_k| rps(c)\geq min\_sup\}$;
16 endfor
17 $RSP=\cup_k F_k$;
18 return $RSP$;
```

Line 1 scans sequence database *D* at first pass and gets *s_count*(*s*), *rps_c* (*s*) of every item *s* and stores them in C_1. Lines 2-3 get *max_s_c*, *max_rps_c* and calculate *rps*(*s*) by using equation 2; Line 4 gets 1-*length* repetitive sequential patterns. Note that the repetitive support threshold *min_sup* here is a percentage within [0,100%]. Lines 7-14 are to calculate the repetitive support of candidate *c*. The other lines are almost the same as GSP. Because repetitive sequences still meet Apriori properties, the candidate generation function *generation*(F_{k-1}) (Line 6) in GSP still works well here, and it is the same as the original GSP and is omitted here.

4. EXPERIMENTS

We compared our algorithm with GSP from two aspects: the number of patterns and their runtime. Four datasets generated by IBM data generator are as follows:

Dataset1(DS1) is C8_T4_S6_I6_DB10k_N100;

Dataset2(*DS2*) is C12_T4_S6_I6_DB10k_N100;

Dataset3(*DS3*) is C10_T8_S20_I10_DB10k_N200;

Dataset3(*DS4*) is C16_T4_S6_I6_DB10k_N100.

C, *T*, *S*, *I*, *DB* and *N* describe characteristics of sequence data. *C* means the average number of elements in each sequence; *T* reveals the average number of items in each element; *I* shows the average size of items per element in potentially maximal large sequences; *DB* is the number of data sequences; and N is the number of items.

The experimental results are shown in Figs. (**1**) and (**2**). From Fig. (**1**), we can clearly find that RptGSP has the ability to mine more sequential patterns than GSP at the same *min_sup*, because it caters for the repetitive property when calculating the candidate support.

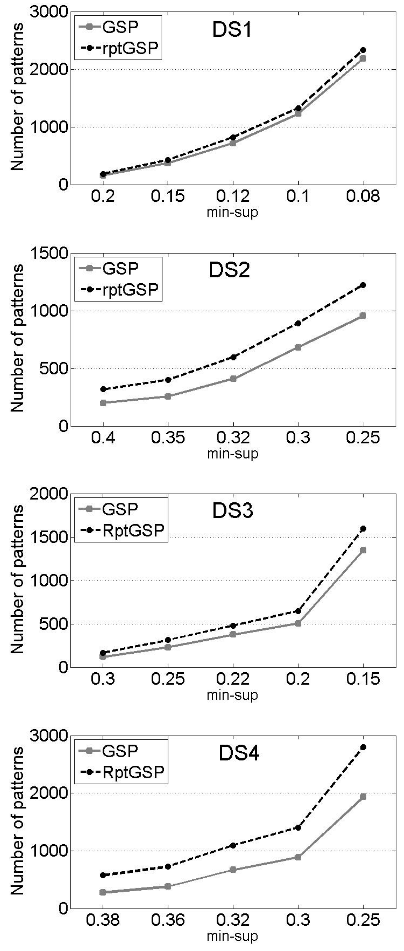

Fig. (1). The number of patterns on the four datasets.

The running time of RptGSP is also higher than GSP, especially with the *min_sup* decreasing, which is shown in Fig. (**2**). The reason is that in order to calculate the repetitive times that a subsequence *s* appears in a data sequence, the rest of the data sequence needs to be scanned after scanning the left appearance end position, whereas GSP does not need to do this step.

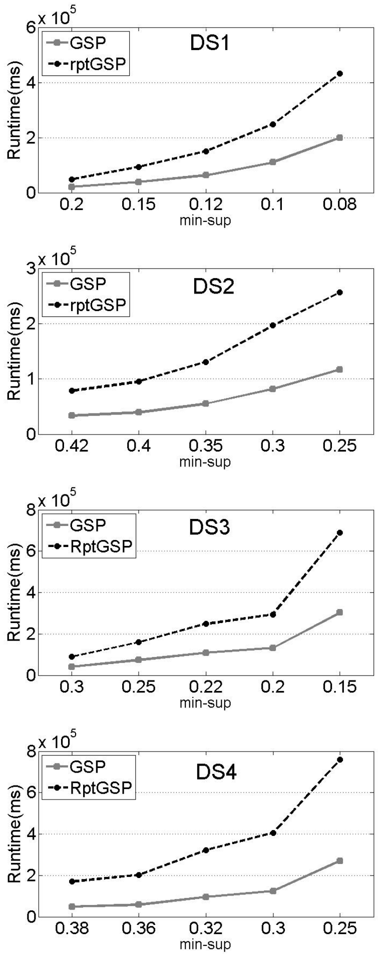

Fig. (2). Runtime on the four datasets

5. CONCLUSION AND FUTURE WORK

Repetitive sequential patterns (RSP) are usually used to understand those special behaviors of repetitive sequences and thus have attracted increasing attention in the recent years. In fact, mining RSP is helpful for deeply understanding the relations of items in many applications, such as DNA periodic analysis, network attack detection, and outlier pattern detection. Accordingly, this paper has defined the repetitive containment problem and proposed an approach to determine the times one sequence makes repeated appearances in a certain data sequence; secondly, we have also proposed a strategy to restrict any pattern's repetitive support still in [0,100%]; finally, we have also proposed an efficient algorithm, called RptGSP, to discover such RSP based on improving GSP Algorithm. RptGSP has been compared with GSP on four datasets, and experiments and comparisons clearly demonstrated that RptGSP has the ability to efficiently capture interesting RSP.

At present, there is no work to consider the repetitive property during the negative sequential patterns mining process. So we are further working on mining repetitive negative sequential patterns.

ACKNOWLEDGEMENTS

This work was supported partly by National Natural Science Foundation of China (71271125), Natural Science Foundation of Shandong Province, China (ZR2011FM028) and Scientific Research Development Plan Project of Shandong Provincial Education Department (J12LN10).

REFERENCES

[1] R. Agrawal and R. Srikant. "Mining sequential patterns," In ICDE'95, 1995, pp.3-14.

[2] R. Srikant and R. Agrawal. "Mining sequential patterns: Generalizations and performance improvements," In EDBT '96: Proc. of the 5th International Conference on Extending Database Technology, London, UK, 1996, pp. 3–17.

[3] J. Han, J. Pei, B. Mortazavi-Asl, Q. Chen, U. Dayal, and M.-C. Hsu. "Freespan: frequent pattern-projected sequential pattern mining," In KDD '00: Proc. Of the 6th ACM SIGKDD international conference on Knowledge discovery and data mining. New York, NY, USA. 2000, pp.355–359.

[4] J. Pei, J. Han, B. Mortazavi-Asl, H. Pinto, Q. Chen, U. Dayal, and M.-C. Hsu. "Prefixspan: Mining sequential patterns efficiently by prefix-projected pattern growth," In ICDE '01: Proc. of the 17th International Conference on Data Engineering, Washington, DC, USA, 2001. pp.215-226.

[5] M. J. Zaki. "Spade: An efficient algorithm for mining frequent sequences,"Machine Learning, 2001, 42(1-2), pp.31–60.

[6] J. Ayres, J. Flannick, J. Gehrke, and T. Yiu. "Sequential pattern mining using a bitmap representation," In KDD'02: Proc. of the 8th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, New York, NY, USA, 2002. ACM. pp. 429–435.

[7] B. Ding, D. Lo and J. Han. Efficient Mining of Closed Repetitive Gapped Subsequences from a Sequence Database. IEEE Computer Society. ICDE '09: Proceedings of the 2009 IEEE International Conference on Data Engineering, 2009, pp. 1024-1035.

[8] L .Brooke. Heidenfelder, D.Michael . Topal. Effects of sequence on repeat expansion during DNA replication. Nucleic Acids Research, Vol. 31, NO. 24, 2003, pp. 7159-7164.

[9] M. Zhang, B. Kao, D. Cheung and K. Yip. "Mining periodic patterns with gap requirement from sequences," *SIGMOD,* 2005.

[10] Y. Tong, L. Zhao, D. Yu and *et al*. "Mining Compressed Repetitive Gapped Sequential Patterns Efficiently," ADMA 2009, pp. 652–660.

[11] E. Lee, W. Kim, J. Ryu and U. Kim. "Efficient Weighted Mining of Repetitive Subsequences," SWS '09, Web Society 2009, pp. 66 –70.

[12] C. Ma and W. Shen. "Clustering Navigation Patterns using Closed Repetitive Gapped Subsequence.," Logistics Systems and Intelligent Management, 2010, pp. 1660 – 1663.

[13] H. Mannila, H. Toivonen, and A.I. Verkamo. "Discovery of frequent episodes in event sequences," DMKD, 1997, pp.259-289.

[14] D. Lo, S.-C. Khoo, and C. Liu. "Efficient mining of iterative patterns for software specification discovery," in KDD, 2007.

[15] D.Lo, Jinyan.Li, Limsoon.Wong, S.-C.Khoo. "Mining Iterative Generators and Representative Rules for Software Specification Discovery," In IEEE 2011, pp. 282-296

[16] Jiawei Han, Guozhu Dong, and YiWen Yin. "Efficient mining of partial periodic patterns in time series database," In Proc. of 15th International Conference on Data Engineering, ICDE 99, pp. 106-115.

[17] Jiong Yang, Wei Wang, and Philip S. Yu. "Mining asynchronous periodic patterns in time series data," In Proceedings of the sixth ACM SIGKDD international conference on Knowledge discovery and data mining, Boston, MA USA, 2000. pp.275-279.

Received: May 26, 2015 Revised: July 14, 2015 Accepted: August 10, 2015