SuffixSpan: 순차패턴 마이닝을 위한 형식적 접근방법

조동영†

요 약

GSP와 같은 Apriori-like 순차패턴 마이닝 방법들은 마이닝 과정에서 많은 후보패턴들을 생성하고, 대용량 데이타베이스의 반복적인 탐색을 필요로 하는 문제점이 있다. 그리고 후보패턴들의 탐색공간을 줄이기 위해 단계별로 프레픽스-프로젝티드(prefix-projected) 데이터베이스를 구성하는 PrefixSpan 방법은 탐색공간을 줄이지만 프로젝티드 데이터베이스의 구성비용이 문제가 된다. 효율적인 순차패턴 마이닝을 위해서는 후보패턴의 생성비용과 탐색공간을 모두 줄여야 한다. 본 논문에서는 이를 위한 새로운 순차패턴 마이닝 방법인 SuffixSpan(**Suffix** checked Sequential **Pattern** mining)을 설명하고, 이에 대한 형식적 접근을 보인다.

SuffixSpan: A Formal Approach For Mining Sequential Patterns

Dong-Young Cho†

ABSTRACT

Typical Apriori-like methods for mining sequential patterns have some problems such as generating of many candidate patterns and repetitive searching of a large database. And PrefixSpan constructs the prefix projected databases which are stepwise partitioned in the mining process. It can reduce the searching space to estimate the support of candidate patterns, but the construction cost of projected databases is still high. For efficient sequential pattern mining, we need to reduce the cost to generate candidate patterns and searching space for the generated ones. To solve these problems, we proposed SuffixSpan(**Suffix** checked Sequential **Pattern** mining), a new method for sequential pattern mining, and show a formal approach to our method.

1. Introduction

There have been many researches about sequential pattern mining to search sequential patterns from a large sequence database[1-3,6, 8]. As sequential pattern mining has a large database as a problem domain unlike text-pattern matching[5], it should search all sequential patterns in database. In addition, it requires efficiency and feasibility in amount of memory and time consumed in sequential pattern mining.

After the first introduction[6,7,9], many

† 정회원 : 전주대학교 정보기술컴퓨터공학부 부교수
논문 접수 : 2002년 9월 25일, 심사완료 : 2002년 10월 28일

studies about sequential pattern mining have been done[1,2,3,6,8]. Typical Apriori-like methods such as GSP[8] adopts a multiple-pass, candidate-generation-and-test approach in sequential pattern mining. Although these Apriori-like methods are simple, it has some problems such as generating of lots of candidate patterns, repetitive searching of a large database, and difficulty of finding long sequential patterns[2]. Recently some researches were proposed to solve some problems caused by Apriori-like methods[1,2,4,8]. Particularly PrefixSpan[2] reduces the searching space by using the projected databases based only on frequent prefixes. The efficiency of PrefixSpan depends on the construction cost of the projected databases.

The smaller the minimal support threshold for sequential patterns, the larger the average length and number of sequential patterns in a given sequence database. If the average length of sequential patterns compared to the number of sequential patterns of the sequence database is long, GSP highly increases the cost of generating candidate patterns and searching for them. Most of all, it requires much more time in the process of the mining since the searching space for estimating the support of candidate patterns generated in each step is a whole sequence database. On the other hand, if the average length of sequential patterns compared to the number of sequential patterns is small, it takes lots of cost to construct the projected databases of PrefixSpan. Therefore for efficient sequential pattern mining, we must reduce the cost of generating candidate patterns and their searching space. To solve these problems, we improve the generation method of candidate patterns in GSP and use the concept of the projected database of PrefixSpan to GSP.

In this paper, we propose SuffixSpan as a new sequential pattern mining method which combines the approaches of GSP and PrefixSpan. SuffixSpan generates a small size of candidate patterns using partition property and suffix property at a low cost and also uses 1-prefix projected databases as the searching spaces for checking candidate patterns.

2. Related work

As we mentioned in section 1, it is necessary to compare with those of GSP, FreeSpan, and PrefixSpan. Similar to AprioriAll[9], GSP is also based on the fact that supersequences of infrequent sequences are not frequent, and repeats candidate generation and testing in order to search all sequential patterns. GSP contributes to expanding the problem area by considering factors such as time constraints, loosely defined transaction, taxonomies of items[8]. But it has still some problems such as the generation of many candidate patterns, many scans of a large database and difficulty of mining long sequential patterns[4,14].

FreeSpan[4] and PrefixSpan[2] was proposed to solve the problems of Apriori-like methods. They use projected sequence databases to reduce the searching space.

PrefixSpan is an enhanced method of FreeSpan. FreeSpan uses frequent items to recursively project sequence database into a set of smaller projected databases and grow subsequence fragments in each projected database[4]. In PrefixSpan, only frequent prefixes are applied to the projection based on the fact that all frequent sequences can be found by expanding frequent prefixes[2]. The major costs of FreeSpan and PrefixSpan are

the construction of the projected databases.

For efficient sequential pattern mining, we need to reduce the generation cost of candidate patterns and its searching space. Although PrefixSpan improves these problems by using the projected databases based on frequent prefixes, it still has another problems in the construction of the projected databases. Although GSP doesn't require the construction of the projected databases, its performance is limited as the searching space to estimate the support of candidate patterns is a whole sequence database. By using partially the concept of the projected databases of PrefixSpan and improving the generation method of candidate patterns in GSP, our approach make it possible to efficiently mine sequential patterns.

3. SuffixSpan: a new method for mining sequential patterns

In this section, we propose a new method of sequential pattern mining, called SuffixSpan which uses combined approaches of GSP and PrefixSpan. The basic idea of SuffixSpan is to generate small candidate patterns at a low cost by using the suffix property and the partition property of sequential patterns. Also we reduce the searching space by using *1*-prefix projected databases. The partition property means that the set of all *k*-sequential patterns can be partitioned by sets which are extensions of all (*k-1*)-sequential pattern sets. The suffix property means that suffixes of all *k*-sequential patterns are (*k-1*)-sequential patterns. The basic terminologies used in this paper are similer to [1,2,4,7-9]. In section 3.1, we show a formal description for our approach. And in section 3.2, we explain the SuffixSpan with an example.

3.1 SuffixSpan: formal approach and algorithm

Definition 1. Given a sequence $s=<e_1e_2...e_n>$, We define the first item of s, denoted by *first(s)*, as x, if $e_1=(x)$, or x_1, if $e_1=(x_1x_2...x_m)$ and $m\geq 2$. And we define the last item of s, denoted by *last(s)*, as x, if $e_n=(x)$, or x_m, if $e_n=(x_1x_2...x_m)$ and $m\geq 2$.

Let I be the set of items. Given a sequence database S, let F be the set of all sequential patterns in S. Let F_k, where k is a positive integer, be the set of all *k*-sequential patterns in S, and let F_x be the set of all sequential patterns such that its first item is x, where $x \in F_1$. Let *max* be the longest length of sequential patterns in S.

Definition 2. Given $s=<e_1e_2...e_n>$, if $e_1=(x)$ and $x\in I$, let $\beta=<e_2...en>$. If $e_1=(x_1x_2...x_m)$ and $m\geq 2$, where $x_i\in I$ and $1\leq i\leq m$, let $\beta=<e_1'e_2...e_n>$, where $e_1'=e_1-\{x_i\}$ and $1\leq i\leq m$. Then β is called a suffix of s.

Definition 3. Given a sequence $s=<e_1e_2...e_n>$, let $\alpha=<e_1e_2...e_{m-1}e_m'>$, $\beta=<e_m''e_{m+1}....e_n>$, where $1\leq m\leq n$, $e_m'\subseteq e_m$, $e_m''=e_m-e_m'$ and $x<y$ for $\forall x\in e_m'$ and $\forall\ y\in e_m''$, then α is called a prefix of s and β is denoted as $\beta=s|\alpha$.

Definition 4. Let $s=<e_1e_2...e_n>\in S$ and $x\in I$. Then s is a x-projected sequential pattern in S if and only if there is a prefix α for some $s'\in S$ such that $s'|\alpha=s$, and *last(α)=x*. The set of all x-projected sequential patterns in S, denoted as S|x, is called the x-projected database in S.

Definition 5. Let $s=<e_1e_2...e_n>\in S$ and let $e_1=(\alpha_1\alpha_2...\alpha_{m1})$, $e_2=(\beta_1\beta_2...\beta_{m2})$, ... , $e_n=(\gamma_1\gamma_2...\gamma_{mn})$. The item-sequence of s, denoted by *item-sequence(s)*, is $\alpha_1\alpha_2...\alpha_{m1}\beta_1\beta_2...\beta_{m2}\gamma_1\gamma_2...\gamma_{mn}$.

The length of item-sequence(s) is $\sum_{i=1}^{n} | e_i |$

Example s_1=<abc>, s_2=<a(bc)> and s_3=<(ab)c>. Then item-sequence(s_1) = item-sequence(s_2)=item-sequence(s_3)=abc. The length of item-sequence(s) is 3.

Definition 6. Let α be an item-sequence with length k in S and let x∈I. F[k,α], called *α-partition of* F_k, is the set of all k-sequential patterns such that its item-sequence is α. And the x-extension of F[k, α], denoted by *extension(F[k,α],x)*, is the set of all *(k+1)*-sequential patterns such that its item-sequence is αx.

Definition 7. Let β be an item-sequence with length k-1, and let X⊆F_1. We define the extension of F_{k-1} by X, denoted by *extension(F[k-1,β],X)*, as $\cup_{x\in X}$ extension(F[k,β x]).

Lemma 1. Let α, β be item-sequences with length k in S. If α is not equal to β, then, for any item x, the intersection of *extension(F[k, α])* and *extension(F[k,β])* is also empty.

Lemma 2. F is partitioned by {F_x | x∈F1} and { F_1, F_2, ... , F_{max}}, respectively.

Theorem 1. F is partitioned by {$F_k \cap F_x$ | k=1,2,...,m, and x∈F_1}.

According to Lemma 2 and Theorem 1, we can partition the set of all sequential patterns into disjoint subsets. Also partition sets in each step can be recursively partitioned by partition sets in the previous step. Using this fact, we can generate the set of all candidate patterns in each step.

Theorem 2. For ∀k≥3, ∀x∈F_1, $F_k \cap F_x$ is partitioned by extension($F_{k-1} \cap F_k$,F_1).

Theorem 3. Let α be an item-sequence with length k. For ∀x∈F_1, k≥1, if F[k,α] is empty, then F[k+1,αx] is also empty.

Theorem 4. If F[k,$x_1x_2....x_k$] is empty, where x_i∈F_1 and 1≤i≤k, then F[k+1, $x_1x_2....x_{(k-1)}yx_k$] is also empty, for ∀y∈F_1.

According to Theorem 2, extension(F_{k-1},F_1) is the set of all candidate sequences of F_k. Particularly, Theorem 3 and Theorem 4 provide a method to search sequences which are not frequent without scanning the sequence database. For example, if F[4,bcba] of F_4 is empty, we can guess that F[5,bcbaa], F[5,bcbab], and F[5,bcbac] of F_5 are all empty by theorem 3. And if F[3,bca] of F_3 is empty, we can guess that F[4,bcba] and F[4,bcca] of F_4 is also empty by Theorem 4.

Theorem 5. Let s be sequence with length k in S. If s is a sequential pattern in S, then the suffix of s is a *(k-1)*-sequential pattern in S.

According to Theorem 5, if a sequence s is a *k*-sequential pattern, its *suffix(s)* must be a *(k-1)*-sequential pattern. In addition, in order to find out whether *suffix(s)* is a *(k-1)*-sequential pattern or not, we will search not the whole F_k but $F_{k-1} \cap F_{first(suffix(s))}$ from theorem 1. Based on above several theorems, our SuffixSpan can be summarized as follows.

```
Algorithm SuffixSpan Mining
// S is a sequential database
// Fk is the set of all k-sequential patterns with
   length k
```

```
// Fx is the set of all sequential patterns s such that
   first(s)=x , for each x∈F1
 1. Input S, min_sup;
 2. By Scanning S, Construct F1, Fx for each x∈F1;
 3. By scanning S,
    for each x∈F1, Construct x-projected databases;
    Construct F2 using the same method of PrefixSpan;
    For each s∈F2, Ffirst(s) ←Ffirst(s) ∪ {s};
 4. // construction of F3(k≥3)
    k←2;
    while (Fk≠)
       Call SuffixSpan(min_sup, Fk, F1, Fx for each x∈F1);
       k←k+1;
 5. Output F1∪F2∪... ∪F(k-1);

Subroutine SuffixSpan(min_sup, Fk, F1, Fx for each x F1)
 1. F(k+1) ←;
 2. By theorem 2, 3, 4 construct C(k+1)-candidate;
 3. For each s∈Ck-candidate
    if (s∈Ffirst(suffix(s)) ∩Fk)
    compute support(s) by scanning first(s)-projected DB;
    if (support(s) ≥ min_sup)
        F(k+1)←F(k+1) ∪ {s};
    Ffirst(s)←Ffirst(s) ∪ {s};
 4. Output F(k+1), Fx for each x∈F1
```

3.2 An Example of SuffixSpan

In this section, we explain the SuffixSpan with an example in <Table 1>. First, through scanning a given sequence database, SuffixSpan generates the set F_1 of all *1*-sequential patterns and the set F_2 of all *2*-sequential patterns. Then it generates the set F_k of all k-sequential patterns by using $F_{(k-1)}$(k ≥ 3), and it repeats this process until it generates the set F_{max} of all sequential patterns with the longest length *max*. Based on Theorem 3, 4, candidate patterns of F_k are constructed. And scanning *1*-prefix projected databases, the supports of candidate patterns are computed. For the example in <Table 1>, SuffixSpan performs as follows.

<Table 1> A sample sequence adatbase

S_{id}	sequence
10	<a(abc)(ac)d(cf)>
20	<ad)c(bc)(ae)>
30	<(ef)(ab)(df)cb
40	<eg(af)cbc)

<Table 2> 1-prefix projected databases

<x>	<x>-projected databases
<a>	<(abc)(ac)d(cf)>, <(_d)c(bc)(ae)>, <(_b)(df)cb>, <(_f)cbc>
<b>	<(_c(ac)d(cf)>, <(_c)(ae)>, <(df)cb>, <c>
<c>	<(ac)d(cf)>, <(bc)(ae)>, <b>, <c>
<d>	<(cf)>, <c(bc)(ae)>, <(_f)cb>
<e>	<(_f)(ab)(df)cb>, <g(af)cbc>
<f>	<(ab)(df)cb>, <cbc>

Step-1 : By scanning a given sequence database, we construct F_1, and generate F_x= {<x>} for each x∈F_1. For a given sequence database in <Table 1>, we can acquire F_1={<a>,<b>,<c>,<d>,<e>,<f>} and F_a={<a>}, F_b={<b>}, F_c={<c>}, F_d={<d>}, F_e={<e>} and F_f={<f>}, supposing that minimal support is 2.

Step-2 : By scanning a given sequence database, we construct the *x*-projected database for each x∈F_1. According to Theorem 1, those projected databases are used the searching space of SuffixSpan. For a sequence database in <Table 1>, the projected databases are constructed as in <Table 2>.

Step-3 : By scanning a given sequence database, we construct the set F_2. Each sequential pattern of F_2 is added to $F_{first(s)}$ according to their first item *first(s)*. In fact, Step-2 and Step-3 can be simultaneously performed by a scanning of a given sequence database. The result of Step-3 is as follows.

$F_2 \cap F_a$ = {<aa>,<ab>,<(ab)>,<ac>,<ad>,<af>}
$F_2 \cap F_b$ = {<ba>,<bc>,<(bc)>,<bd>,<bf>}

$F_2 \cap F_c$ = {<ca>,<cb>,<cc>}
$F_2 \cap F_d$ = {<db>,<dc>}
$F_2 \cap F_e$ = {<ea>,<eb>,<ec>,<ef>}
$F_2 \cap F_f$ = {<fb>,<fc>}

Step-4 : In SuffixSpan, the method to construct F_1, F_2, and F_3 is similar to that of PrefixSpan. However, the method to obtain $F_k(k \geq 4)$ is different from that of PrefixSpan. The method to generate F_k from $F_{k-1}(k \geq 3)$ can be explained as follows.

Most of all, the candidate sequence set $C_{k\text{-candidate}}$ of F_k can be generated by $F_{k-1} \cap F_{first(suffix(s))}$, s is ignored. Otherwise, it should be added to F_k and $F_{first(s)}$, if it satisfies minimal support as the result of scanning *first(s)*-projected database which is constructed in Step-2. For example, considering s=<a(bc)> $\in F_3$ and <a>$\in F_1$ in Step-4 of <Table 3>, *a*-extension of <a(bc)> is s_1=<a(bc)a> and s_2=<a(bca)>. suffix(s_1)=<(bc)a>$\in F_3$, however, suffix(s_2)=<(bca)> F_3. Therefore, s_2 is ignored according to Theorem 5. And as a result of scanning *a*-projected database, support(s_1)=2$\geq$ min_sup. Therefore, s_1 belongs to F_4. Considering s=<a(bc)>$\in F_3$ and <b>$\in F_1$, since

<Table 3> SuffixSpan's stepwise Results for <Table 1>

	Step-1 (Genetation of F_1)	Step-2 (Genetation of F_2)	Step 3 (Genetation of F_3)	Step-4 (Genetation of F_4)	Step-5 (Genetation of F_5)
F_a	<a>	<aa>, <ab>,<(ab)>, <ac>, <ad>, <af>	<aba>, <a(bc)>,<abc>,<(ab)c>, <(ab)d>,<(ab)f>, <aca>,<acb>,<acc>, <adc>	<a(bc)a>, <(ab)dc>	-
F_b	<b>	<ba>, <bc>,<(bc)>, <bd>,<bf>	<(bc)a>,<bdc>	-	
F_c	<c>	<ca>,<cb>,<cc>,	-		
F_d	<d>	<db>,<dc>	<dcb>	-	
F_e	<e>	<ea>,<eb>, <ec>,<ef>,	<eab>,<eac>,<ebc>, <ecb>, <efb>,<efc>	<efcb>	
F_f	<f>	<fb>,<fc>	<fbc>,<fcb>	-	

constructing *extension*($F_{(k-1)}$, F_1) and then removing sequences that cannot be sequential patterns by Theorem 3 and Theorem 4. For example, in step-3 of <Table 3>, the candidate sequence set $C_{3\text{-candidate}}$ is constructed by removing the sequences that belong to *extension*($F_2 \cap F_a$,{<e>}),*extension*($F_2 \cap F_b$,{<b>, <e>}), *extension*($F_2 \cap F_c$, { <d>, <e>, <f> }), *extension*($F_2 \cap F_d$, { <a>, <d>, <e>, <f> }), *extension*($F_2 \cap F_e$, { <d>, <e> }), and *extension*($F_2 \cap F_f$, { <a>, <d>, <e>, <f> }) in extension(F_2, F_1). According to Theorem 5, if suffix(s), where s$\in C_{k\ candidate}$, do not exist in F[3,abb] is empty, sequence <a(bc)b> cannot be a sequential pattern by Theorem 4. In the same way we repeat the process to obtain F_k from F_{k-1} and F_1 until F_k is empty.

4. Comparison of GSP, PrefixSpan, and SuffixSpan

The candidate-generation-and-test approach of SuffixSpan is similar to that of GSP, but it is different from that of GSP in the method to generate candidate patterns and in that it uses *1*-prefix projected databases instead of given

sequence database as a searching space. In addition, SuffixSpan constructs projected databases only once instead of recursively constructing projected database in each step as in PrefixSpan.

Like almost all sequential pattern mining methods, algorithms of GSP, PrefixSpan, and SuffixSpan have different results in their performances according to methods of their implementation, since they demand a great deal of block I/O and in-memory for dealing with a large database. Therefore, the comparison of these algorithms by their implementations leads to unreliable results. In order to obtain reliable results, it is necessary of us to perform the comparison by an analytic method rather than implementation. In the performance evaluation of GSP, PrefixSpan, and SuffixSpan, the major factors are the number of scanning of a given sequence database, the generation cost of candidate patterns in each step, and the testing cost of candidate patterns. The total costs of each algorithms are represented as follows.

$$(1)\ C_{GSP} = \alpha + \sum_{i\geq 2} (\beta_{i\text{-}GSP} + \gamma_{i\text{-}GSP} * \alpha)$$

$$(2)\ C_{prefix} = 2\alpha + C + \sum_{i\geq 3} (\beta(i,(i-1)) + \gamma_{i\text{-}prefix} * p_i \cdot \alpha), \quad \text{where } 0 \leq p_i < 1$$

$$(3)\ C_{suffix} = 2\alpha + C + \beta(2) + \sum_{i\geq 3} (\beta_{i\text{-}Suffix} + \gamma_{i\text{-}Suffix} * q \cdot \alpha), \quad \text{where } 0 \leq q < 1$$

Let S be a given sequence database S. In above formula, α is the one time scanning cost of S and C is a additional cost required for the generation of F_2 in PrefixSpan and SuffixSpan. $\beta_{i\ GSP}$, $\beta(i,(i-1))$, and $\beta_{i\ Suffix}$ are generation costs of i-candidate patterns in GSP, PrefixSpan, and SuffixSpan, respectively. $\gamma_{i\text{-}GSP}$, $\gamma_{i\text{-}prefix}$, and $\gamma_{i\text{-}Suffix}$ are numbers of i-candidate patterns in GSP, PrefixSpan, and SuffixSpan, respectively. $p_i \cdot \alpha$ and $q \cdot \alpha$ are scanning costs for its searching space in PrefixSpan and SuffixSpan respectively. p_i is a decreasing ratio of projected databases in PrefixSpan and q is a decreasing ratio of 1-prefix projected databases for S in SuffixSpan, and p_3 is equal to q. $\beta(i,(i-1))$ is the construction cost of projected databases which are used searching spaces for F_i. In general, is a monotonously decreasing function, and $\beta(max+1,\beta(max))=0$, where *max* is the length of the longest sequential patterns in S.

If $(\gamma_{suffix} * q \cdot \alpha)$ is less than $(\gamma_{i\text{-}GSP} * \alpha)$, the cost of SuffixSpan is lower than that of GSP, although SuffixSpan has additional costs such as C and $\beta(2)$. And if $p_i \cdot \alpha$ is not sufficiently less than $q \cdot \alpha$, the cost of SuffixSpan is lower than that of PrefixSpan because $\sum_{i\geq 3} \beta(i, \beta(i-1))$ is big. The values of p_i and q are depend on a given sequence database. In our observation, $(\beta_{2\text{-}GSP} + \gamma_{2\text{-}GSP} * \alpha)$ is greater than $(\alpha + C)$, and $(\gamma_{suffix} * q \cdot \alpha)$ is less than $(\gamma_{i\text{-}GSP} * \alpha)$ because $q \cdot \alpha$ is sufficiently less than α and $\gamma_{i\text{-}suffix}$ is less than α or equal to $\gamma_{i\text{-}GSP}$. Also $p_i \cdot \alpha$ is not sufficiently less than $q \cdot \alpha$. Therefore SuffixSpan is more efficient than GSP and PrefixSpan, although exact values of p_i and q are depend on a given sequence database.

5. Conclusions

In this paper, we proposed SuffixSpan, a new method for sequential pattern mining and show a formal approach to our method. Our basic idea is to generate small candidate patterns at a low cost by using partition and suffix property of sequential patterns and to reduce the searching space by constructing

1-prefix projected databases.

In general, for efficient sequential pattern mining, the generation cost of candidate patterns and its searching space should be reduced at the same time. With SuffixSpan we can achieve this goal, by applying the idea of projected database of PrefixSpan to GSP and by improving the generation method for the candidate patterns of GSP. The formal approach to SuffixSpan shown in this paper provides a formal model for the problems of sequential pattern mining.

We should make a further study on the extension of SuffixSpan and the development of an analytic cost model and applied systems of SuffixSpan, considering time constraints, loosely defined transaction, and taxonomies among items which are defined in [11].

References

[1] M. J. Zaki. SPADE: An Efficient Algorithm for Mining Frequent Sequences. In Proc. of Machine Learning Journal, special issue on Unsupervised Learning (Doug Fisher, ed.), Vol. 42 Nos. 1/2, pages 31-60, Jan/Feb 2001.

[2] 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. Proc. 2001 Int. Conf. Data Engineering (ICDE'01), pages 215-224, Heidelberg, Germany, April 2001.

[3] J. Han, J. Pei, B. Mortazavi-Asi, Q. Chen, U. Dayal, and M.-C. Hsu. FreeSpan: Frequent Pattern-Projected Sequential Pattern Mining. In Proceedings of the Association for Computing Machinery Sixth International Conference on Knowledge Discovery and Data Mining, pages 355-359, 2000.

[4] J. Han, J. Pei, Y. Yin, Mining Frequent Patterns without Candidate Generation, Proc. 2000, ACM-SIGMOD Int. Conf. on Management of Data(SIGMOD'2000), pp. 1-12, Dallas TX May 2000

[5] M.Garofalakis, R. Rastogi, and K. Shim. Spirit: Sequential pattern mining with regular expression constraints. In Proc. 1999 Int. Conf. Very Large Data Bases(VLdatabase99), pages 223-234, Edinburgh, UK, Sept. 1999.

[6] R.Agrawal and R.Srikant. Mining Sequential Patterns. In Proc. Of the 11th Intl Conference on Data Engineering, Taipei, Tiwan, March 1995.

[7] R.Srikant and R.Agrawal. Mining Generalized Association Rules. In Proc. Of the 21st Intl Conference on Very Database, Zurich, Switzerland, September 1995.

[8] R.Srikant and R.Agrawal. Mining Sequential Patterns: Generalizations and Performance Improvements. Research Reports RJ 9994, IBM Almaden Research Center, San Jose, California, December 1995.

[9] R.Agrawal, S. Srikant, "Fast Algorithms for Mining Association Rules", Proc. 1994 Int. Conf. on Very Large Data Base, pp. 487-499, Santiago, Chile, Sept. 1994.

조 동 영

1986 고려대학교 수학교육과 (이학사)
1988 고려대학교 이학석사 (전산학 전공)
1992 고려대학교 이학박사(전산학 전공)
1993~현재 전주대 정보기술컴퓨터공학부 부교수
관심분야: 데이터공학, 컴퓨터교육, 이동컴퓨팅
E-Mail: chody@jeonju.ac.kr