STUDIA UNIV. BABEŞ–BOLYAI, INFORMATICA, Volume L, Number 1, 2005

LARGE CANDIDATE BRANCH-BASED METHOD FOR MINING CONCURRENT BRANCH PATTERNS

JING LU, OSEI ADJEI, WEIRU CHEN, FIAZ HUSSAIN, CĂLIN ENĂCHESCU, AND DUMITRU RĂDOIU

Abstract. This paper presents a novel data mining technique, known as Post Sequential Patterns Mining. The technique can be used to discover structural patterns that are composed of sequential patterns, branch patterns or iterative patterns. The concurrent branch pattern is one of the main forms of structural patterns and plays an important role in event-based data modelling. To discover concurrent branch patterns efficiently, a concurrent group is defined and this is used roughly to discover candidate branch patterns. Our technique accomplishes this by using an algorithm to determine concurrent branch patterns given a customer database. The computation of the support for such patterns is also discussed.

Keywords: Post Sequential Patterns Mining; Concurrent Branch Patterns; Sequential Patterns Mining

1. Introduction

Sequential patterns mining proposed by Agrawal and Srikant [1] is an important data mining task and with broad applications. Based on the analysis of sequential patterns mining, we proposed a novel framework for sequential patterns called sequential pattern graph (SPG) as a model to represent relations among sequential patterns [2]. SPG can be used to represent sequential patterns encountered in patterns mining. It is not only a minimal representation of Sequential patterns mining result, but it also represents the interrelation among patterns. It establishes further the foundation for mining structural knowledge. Based on SPG and sequential patterns mining, a new mining technique called post sequential patterns mining (PSPM) [3] is presented to discover new kind of structural patterns. A structural pattern [4] is a new pattern, which is composed of sequential patterns, branch patterns or iterative patterns.

In order to perform post sequential patterns mining, the traditional sequential patterns mining should be firstly completed. Post sequential patterns mining can be viewed as a three-phase operation that consists of *pre-processing*, *processing* and *post-processing* phases. In the pre-processing phase, based on the result of sequential patterns mining, the Sequential Patterns Graph (SPG) is constructed. SPG is a bridge between traditional sequential patterns mining and the novel post

Received by the editors: June 1, 2005.

sequential patterns mining. The processing phase corresponds to the execution of the mining algorithm, given the maximal sequences set (MSS) recognized by SPG and customer sequence database DB as input, structural patterns (including concurrent branch patterns, exclusive branch patterns and iterative patterns) are discovered. During post-processing, the mined structural pattern can be represented graphically. In this paper, we focus on concurrent branch pattern and its mining algorithms. We address the question: *Given a set of sequential patterns and customer sequence database DB, how can we efficiently find concurrent branch patterns?*

The first step in the discovery of a concurrent process should be to identify the individual threads and their individual behaviours. Our work demonstrates that since it is part of the post sequential patterns mining, concurrent branch pattern mining discovers patterns on the basis of sequential patterns. In a concurrent process, it is important to also locate the points where the threads interact. Our method solves this crucial problem by taking out a common prefix and/or a common postfix from sequential patterns that is candidate branch patterns. Section 3, discusses the concurrent branch pattern mining algorithm whilst section 4, reviews some related work. Section 5 presents our conclusions.

2. Problem Statement

To formally define the concurrent branch mining algorithm we introduce some basic terminology. In the following definition, let SP represent Sequential Patterns; $x\alpha$, $x\beta$, αy, βy, $x\alpha y$, $x\beta y \in SP$; α, $\beta \in SP$; $x, y \in SP$ or $x, y \in \emptyset$.

Definition 1: CandidateBranch Pattern

Sequential patterns which contain common prefix and/or common postfix can constitute *Candidate Branch Pattern.*

- Sequential patterns $x\alpha$ and $x\beta$ can make up a candidate branch pattern which has a sub-sequence x as a common prefix and denoted by $x[\alpha, \beta]$.
- Sequential patterns αy and βy can make up a candidate branch pattern which has a sub-sequence y as a common postfix and denoted by $[\alpha, \beta]y$.
- Sequential patterns $x\alpha y$ and $x\beta y$ can make up a candidate branch pattern which has sub-sequence x as a common prefix, sub-sequence y as a common postfix and denoted by $x[\alpha, \beta]y$.

In the above definitions, notation $[\alpha, \beta]$ represents two *branches* of a candidate branch pattern.

Let us consider some examples. Sequential patterns <*efcb*> and <*ebc*> can make up a candidate branch pattern which has e as a prefix and denoted by $e[fcb,bc]$. This candidate branch pattern has two branches, *fcb* and *bc*. Sequential patterns <*fcb*>, <*dcb*> and <*acb*> can make up a candidate branch pattern which has *cb* as a postfix and denoted by [*f*, *d*, *a*]*cb*. This candidate branch pattern has three branches *f*, *d* and *a*. It should be noted that in a candidate branch pattern such as $a[b,c]d$, the order of b and c is indefinite. Therefore, $a[b,c]d$ can appear in a transaction database in the form of *abcd*, *acbd* or *a(b,c)d*. The purpose of defining

a candidate branch pattern is to discover true branch patterns (concurrent branch patterns or exclusive branch patterns). A candidate branch pattern can also be extended to multiple sequential patterns.

Definition 2: Concurrence

The *concurrence* of sub-sequential patterns α and β is defined as the fraction of customers that contain α and β simultaneously and it is denoted as:

$$concurrence(\alpha \wedge \beta) = \|\{T : \alpha \cup \beta \subseteq T, T \in D\}\|/\|D\|$$

Let *minsup* be user specified minimal support, if *concurrence* $(\alpha \wedge \beta) \geq$ *minsup* is satisfied then α and β are *concurrent*. Similarly, multiple candidate branches $a_1 \ldots a_i$ ($\alpha_i \in SP$; $1 \leq i \leq n$) are concurrent branches if and only if *concurrence* $(a_1 \wedge \ldots \wedge a_i) \geq$ *minsup*.

Definition 3: Concurrent Branch

Two branches α and β of candidate branch pattern $x[\alpha, \beta]y$ are *concurrent branch* if and only if in a transaction database, α and β are concurrent between a common prefix x and/or common postfix y.

Definition 4: Concurrent Branch Pattern

For candidate branch pattern $x[\alpha, \beta]y$, if branches α and β are concurrent branches, then $x[\alpha, \beta]y$ is *concurrent branch pattern*.

The **problem of concurrent branch pattern mining** is to *find the complete set of concurrent branch patterns in a given sequential pattern mining result and customer sequence database DB with respect to given support threshold.*

Example 1: Let us consider a customer sequence database in PrefixSpan [5]:

(1) <a (a,b,c) (a,c) d (c,f)>
(2) <(a,d) c (b,c) (a,c)>
(3) <(e,f) (a,b) (d,f) c b>
(4) <e g (a,f) c b c>

and two branches f and eb of the candidate branch pattern [f, eb]c. Let *minsup*=50%.

Both customer sequence (3) <(e,f) (a,b) (d,f) c b> and (4) <e g (a,f) c b c> contain f and eb. Thus, the concurrence ($f \wedge eb$) is 50%. That is, f and eb are concurrent branches and *sup* ([f,eb]c)=50%. Therefore, the candidate branch pattern [f, eb]c is a concurrent branch pattern.

It can be concluded from definition 4 that, concurrent branch patterns mining problem can be decomposed into the following sub-problems of: how to generate all *candidate branch pattern*; how to determine the *concurrence* of candidate branches; and how to calculate the *support* of candidate branch pattern.

3. Concurrent Branch Pattern Mining

Let us consider the first problem in concurrent branch patterns mining, i.e., how to generate all candidate branch patterns. Since the concurrent branch pattern mining is based on the result of sequential patterns mining, which is the set of sequential pattern, hence the direct way to discover candidate branch pattern should be based on sequential pattern set. All candidate branch patterns can be

generated by taking out a common prefix or/and a common postfix from sequential pattern set. However, the shortcoming of this method is that some non-concurrent branches can be generated.

In order to get rid of non-concurrent items, the concurrent group and the maximal concurrent group are defined first. Then, rough concurrent branch patterns are computed based on the maximal concurrent group to obtain candidate branch patterns.

3.1. Concurrent Group and Rough Concurrent Branch Pattern.

Definition 5: Concurrent Group (*CG*)

Given customer sequences database *DB*, set of items (or itemset) that have transaction support above *minsup* makes up a *concurrent group* and it is denoted by *CG* for brief.

Definition 6: Maximal Concurrent Group (*MCG*)

A concurrent group is called a *maximal concurrent group* if any of its superset is not a concurrent group. The set of maximal concurrent group set is denoted by ***MCGS*** for abbreviation.

Example 2: Consider the customer sequences in example 1 and let *minsup* be 50%. Items (or itemset) sets {a,b,c,d}, {(a,b),c,d,f} and {(a,c),b,d} are all examples of concurrent group since the condition in definition 5 is satisfied. From definition 5 we know that concurrent group is a set and the elements in this set can be an item or an itemset. Consider {(a,b),c,d,f} for example, four elements are contained in this concurrent group, one is an itemset (a,b) and the other three are items c,d, and f. Among these three concurrent groups, {(a,b),c,d,f} is a maximal concurrent group but {a,b,c,d} is not, since its superset {(a,b),c,d,f} is a concurrent group.

If each customer sequence is considered as a transaction, then discovering concurrent group from customer sequence database is identical to the discovery of frequent patterns. The maximal concurrent group of the above example is:

$$MCG = \{\{(a,b),c,d,f\},\{(a,c),(b,c),d\},a,b,c,e,f\}\}$$

Following the definition of the maximal concurrent group, we investigate the relation between the maximal sequence set (MSS) discovered in sequential patterns mining and the maximal concurrent group proposed.

Definition 7: Rough Concurrent Branch Pattern (*RCBP*)

Let C be a maximal concurrent group in *MCG*. *Concurrent sequences* can be obtained by the *sequential intersection* operation of C and each element in *MSS* respectively. These concurrent sequences constitute a rough concurrent branch pattern and denoted by *RCBP* for brief. Sequential intersection operation can be treated as a normal intersection, and the sequence relations among elements after this operation will be consistent with that in the original sequence pattern. The notation for sequential intersection is:

Sequential pattern or Sequential pattern set $\cap$ *Concurrent Group*

Rough Concurrent Branch Pattern is a candidate branch pattern, which has a null common prefix and a null common postfix.

Algorithm 1: Cal_RCBP (Getting a RCBP)
Input: Maximal concurrent group C and maximal sequence set *MSS*.
Output: Rough Concurrent Branch Patterns *RCBP(C)*.
Method: Find the rough concurrent branch patterns in the following steps:

(1) Let rough concurrent branch pattern for *C, RCBP(C)*, be empty.
(2) For each element *ms* in *MSS*
    Add *ms* to *RCBP(C)*;
    For each element (item or itemset) i in *ms*, test if i is an element of C or i is included in one element of C;
    If neither condition is satisfied, then delete i from *ms*.
(3) Delete the element in *RCBP(C)* which contained by another pattern in the *RCBP(C)*.
(4) The result is *RCBP(C)*.

Example 3: Given *MSS*={<eacb>, <efcb>, <a(b,c)a>, <(a,b)dc>, <fbc>, <(a,b)f>, <ebc>, <dcb>, <abc>, <acc>, <(a,c)>} and maximal concurrent group *MCG* = {{(a,b), c, d, f}, {(a,c), (b,c), d}, {a, b, c, e, f}}.

MCG	RCBP
{(a,b),c,d,f}	{<acb>, <fcb>,<aca>,<(a,b)dc>,<fbc>,<(a,b)f>, <dcb>,<abc>,
{(a,c),(b,c),d}	{<acb>, <a(b,c)a>,<aba>,<aca>,<adc>,<bdc>, <dcb>, <abc>,
{a,b,c,e,f}	{<eacb>,<efcb>,<aba>,<aca>,<fbc>,<af>,<bf>,<ebc>,<abc>,

TABLE 1. Rough Concurrent Branch Pattern Example

The rough concurrent branch patterns can be computed using algorithm 1. The final result is shown in table 1.

3.2. **Sub-customer sequence set.** The feature of our method is that the customer sequence database *DB* is not used for counting support after the discovering of candidate branch patterns. Rather, the sub customer sequence set *SubDB* is used for this purpose. The number of entries in *SubDB* may be smaller than the number of transaction in *DB*. In addition, each entry may be smaller than the corresponding transaction because the items (itemset) before the prefix element or after the postfix element are deleted.

Definition 8: Sub-customer sequence set

Given a candidate branch pattern $x[\alpha, \beta]y$ and a customer sequence database *DB*, the *sub customer sequence set* of *DB* is obtained by deleting the *minimal pre-sub sequence* contains prefix x or/and the *minimal post-sub sequence* contains a postfix y of each customer sequence in *DB*. This is denoted by *SubDB*(x,y).

The support of the sub-customer sequence set *SubDB*(x,y) is:

$$sup(\text{SubDB}(x,y))=|\ \text{SubDB}(x,y)|/|\text{DB}|$$

Explanation. Minimal pre-sub sequence contains x: Suppose x=*cd* and the customer sequence is *acbdefdg*. The result for deleting the minimal pre-sub sequence is *efdg*.

Minimal post-sub sequence contains y: Suppose y=*bg* and customer sequence is *acbdefdg*. The result for deleting the minimal post-sub sequence is *ac*.

The purpose of finding the sub-customer sequence set is to calculate the *support* of the candidate branch pattern $x[\alpha, \beta]y$ and to determine the *concurrence* of branches $[\alpha, \beta]$. For a candidate branch pattern $x[\alpha, \beta]y$, (i) if *sup*(SubDB(x,y))< *minsup*, then the candidate branch pattern $x[\alpha, \beta]y$ cannot be a concurrent branch pattern; (ii) if *sup*(SubDB(x,y))≥ *minsup*, then only the *concurrence* checking of branches x and y is needed. That is, it is only necessary to check if x and y occurs simultaneously in each sub customer sequence of *SubDB*(x,y).

Algorithm 2: Gen_SubDB (Computes Sub-customer sequence set)
Input Common prefix x and/or common postfix y of candidate branch pattern $x[\alpha, \beta]y$; Customer sequence database *DB*.
Output Sub customer sequence set *SubDB*.
Method:

SubDB(x,y)=∅;
For each customer sequence *cs*∈*DB* **Do**
{ Scan *cs* from left to right, find the sub customer sequence which contains prefix x completely, record the position p of the last matched element in *cs*. If not found, set p be the length of *cs*;
Scan *cs* from right to left, find the sub customer sequence which contains prefix y completely, record the position q of the first matched element in *cs*. If not found, set p be 0;
If p≥q //There is no sub sequence have prefix x and postfix y in *cs*
then *DB*=*DB*-{*cs*} //Delete *cs* from *DB*
else
Delete sub customer sequence before the p^{th} (contains p^{th}) element and after the q^{th} (contains q^{th}), obtained $cs(xy)$
SubDB(x,y)= *SubDB(x,y)*∪ $cs(xy)$
End if
}
return *SubDB(x,y)*.

Theorem 1: Given a candidate branch pattern $x[\alpha, \beta]y$ and sub customer sequence set *SubDB(x,y)*, if α and β are *concurrent* in *SubDB(x,y)* i.e., if the number of occurrence of α and β simultaneously in *SubDB(x,y)* is greater than or equal to a user specified minimal support *minsup*, then the candidate branch pattern $x[\alpha, \beta]y$ is a concurrent branch pattern. (Proof is omitted for brevity)

Thus, the problems of how to determine the concurrence of candidate branch pattern and how to calculate the support of a candidate branch pattern reduces to

how to find a sub-customer sequence set *SubDB* and how to check the concurrence of a candidate branch pattern in *SubDB*.

3.3. **Concurrent Branch Pattern Mining Method.** Steps taken to mine concurrent branch patterns based on candidate branch pattern are given as follows.

(1) Find the maximal sequence set (*MSS*) from customer sequences using traditional sequential patterns mining algorithm;
(2) Find the maximal concurrent group set (*MCGS*) from customer sequences in *DB* using traditional frequent patterns mining algorithm;
(3) Generate the rough concurrent branch pattern (*RCBP*) using *Cal_RCBP* algorithm;
(4) Calculate the sub-customer sequence set (*SubDB*) using *Gen_SubDB* algorithm;
(5) Determine the support of the candidate branch in the sub-customer sequence set to generate concurrent branch pattern.

Example 4: Given a customer sequence database *DB* (refer to example 1) and its rough concurrent branch pattern *RCBP* (shown in table 1), steps taken to find all concurrent branch patterns are as follows:

(1) Generate the candidate branch pattern *CanBP* based on *RCBP*. With respect to Table 1:
RCBP(3) = {<eacb>, <efcb>, <aba>, <aca>, <fbc>, <af>, <bf>, <ebc>, <abc>, <acc>}.
Since <*eacb*> and <*efcb*> have a common prefix *e* and a common postfix *cb*, these two sequential patterns can constitute candidate branch pattern *e*[*a,f*]*cb*; <*aba*> and <*aca*> have a common prefix *a* and common postfix *a*, and make up a candidate branch pattern *a*[*b,c*]*a*; Similarly, <*af*> and <*bf*> make up [*a,b*]*f*.
The candidate branch pattern set of RCBP(3) is CanBP_3 = {e[a,f]cb; a[b,c]a; a[b,c]c; ab[a,c]; ac[a,c]; [f,e,a]bc}. In the same way, CanBP_1 = {ac[a,b,c]; [a,f,d]cb; (a,b)[dc,f]; a[b,c]c; [a,f]bc; f[cb,bc]}; CanBP_2 = {ac[a,b,c]; ab[a,c]; [a,d]cb; a[b,c]a; a[b,d,c]c; [a,b]dc}
(2) Generate the Sub-Customer Sequence Set *SubDB*
For the common prefix and/or common postfix in the above candidate branch patterns, the sub-customer sequence set of example 1 can be calculated by using algorithm 2. The result is shown in table 2.
(3) Counting the support to find Concurrent Branch Pattern *CBP*
Calculate the support of the candidate branch in sub-customer sequence set *SubDB* to generate concurrent branch patterns. Here, we only consider: CanBP_1={ac[a,b,c];[a,f,d]cb;(a,b)[dc,f];a[b,c]c;[a,f]bc;a[b,c]a }. Table 3 is an example of the processes involved in the calculation of the support.

Next, the candidate branch which is not concurrent and the number of which is at least 2 is decomposed. The concurrence of its decomposition is determined

Prefix	Postfix	SubDB			
e	cb	(a,b) (d,f)	G (a,f)		
a	c	(a,b,c) (a,c) d	C (b,c)	(d,f)	c b
ab	-	(a,c) d (c,f)	(a,c)	c	
ac	-	(a,c) d (c,f)	(b,c) (a,c)	b	b c
(a,b)	-	(a,c) d (c,f)	(d,f) c b		
-	cb	(a,d)	(e,f) (a,b) (d,f)	e g (a,f)	
-	bc	a	(a,d) c	(e,f)	e g (a,f) c

TABLE 2. Sub Customer Sequence Set of Example 1

Prefix	Branches	Postfix	support in Sub customer sequence set	Concurrence
ac	a,b,c	-	1	No
a	b,c	c	3	Yes
-	f,a	bc	1	No
-	a,f,d	cb	1	No
(a,b)	dc,f	-	2	Yes
a	b,c	a	2	Yes

TABLE 3. Example for counting support for $CanBP_1$

continuously. Since $ac[a,b,c]$ is not concurrent, it is decomposed into [a,b], [a,c], [b,c]. Also [a,f,d]cb is not concurrent and it is decomposed into [a,f], [a,d], [f,d]. The process is shown in table 4.

Finally, the concurrent branch pattern CBP_1 derived from $CanBP_1$ is computed as CBP_1={a[b,c]c, (a,b)[dc,f], ac[a,c], ac[b,c], [a,d]cb, [a,f]cb, a[b,c]a}.

4. Related Work

Concurrency is particularly a difficult aspect of some systems' behaviours. Cook *et al.* [6] presented a technique to discover patterns of concurrent behaviour from traces of system events. The technique uses statistical and probabilistic analyses to determine when a concurrent behaviour occurs, and what dependent relationships might exist among events. The technique is useful in a wide variety of software engineering tasks that includes, re-engineering, user interaction modelling, and software process improvement.

Prefix	Branches	Postfix	support in Sub customer sequence set	Concurrence
ac	a,b	-	1	No
ac	a,c	-	2	Yes
ac	b,c	-	2	Yes
-	a,f	cb	2	Yes
-	a,d	cb	2	Yes
-	f,d	cb	1	No

TABLE 4. Example for counting support for the decomposition of *CanBP*

Other related work can also be found in the area of workflow data analysis, since many workflows exhibit concurrent behaviour. Herbst [7-9] investigated the discovery of both sequential and concurrent workflow models from logged executions. Agrawal *et al.* [10] investigated production activity dependency graphs from event-based workflow logs that had already identified the partial ordering of concurrent activities.

5. Conclusions

In this paper, we developed candidate branches based method to detect concurrent behaviour in customer sequence database and to infer a model that describes the concurrent behaviour. The problem of finding concurrent branch pattern was first introduced in this paper. This problem is concerned with finding concurrent branch pattern in a given sequential pattern mining result and a customer database. The main purpose of Post Sequential Patterns Mining is to discover the hidden structural patterns in event-based data. Concurrent branch pattern is an important pattern, which occurs in many event-based data. Thus, we concentrated on concurrent branch pattern mining in this paper.

An important phase for our work is to perform more experiments to support our theories. In our previous work [2], we implemented the algorithm for constructing SPG and analysed the efficiency of that approach. In our existing research work, we anticipate that more experiments are needed to demonstrate the affective nature and efficiency of concurrent branch patterns mining algorithms. This paper has been theoretical; experimentation is on going to establish the validity of our algorithms. In addition to the above, we intend to extend the method to cover concurrent branch patterns to exclusive branch patterns mining or iterative patterns mining. This, we envisage will be our ultimate goal.

References

[1] Agrawal, R. & Srikant, R. (1995). Mining sequential patterns. *Proceedings. of the Eleventh Internal Conference on Data Engineering*(pp.3-14). Taipei, Taiwan. IEEE Computer Society Press.

[2] Lu,J., Wang, X.F., Adjei, O., & Hussain, F.(2004a) *Sequential Patterns Graph and its Construction Algorithm*. Chinese Journal of Computers. *6*, 782-788.

[3] Lu, J., Adjei, O., Wang, X.F., & Hussain, F. (2004b) Sequential Patterns Modeling and Graph Pattern Mining. *Proceedings of the Tenth International Conference IPMU* (pp.755-761). Perugia: Italy.

[4] Lu,J., Adjei, O., Chen, W.R., & Liu, J. (2004c) Post Sequential Pattern Mining: A new Method for discovering Structural Patterns. *Proceedings of International Conference on Intelligent Information Process* (pp.239-250). Beijing: China.

[5] Pei,J., Han, J.W., Behzad M.A, & Pinto, H.(2001). PrefixSpan: Mining sequential patterns efficiently by prefix-projected pattern growth. *Proceedings. of 17th International Conference on Data Engineering* (pp.215-226). Heidelberg: Germany.

[6] Cook ,J.E., & Wolf ,A.L. (1998). Event-Based Detection of Concurrency. *Proceedings of the Sixth International Symposium on the Foundations of Software Engineering* (pp.35-45). Orlando: FL.

[7] Herbst, J.(1999). Inducing Workflow Models from Workflow Instances. *Proceedings of the Sixth European Concurrent Engineering Conference*, *Society for Computer Simulation* (pp.175-182).

[8] Herbst, J.(2000a). A Machine Learning Approach to Workflow Management. *Proceedings of European Conference on Machine Learning*(pp.183-194). Lecture Notes in Artificial Intelligence Nr. 1810.

[9] Herbst, J.(2000b). Dealing with Concurrency in Workflow Induction. *Proceedings of the Seventh European Concurrent Engineering Conference, Society for Computer Simulation* (pp.169- 174).

[10] Agrawal, R., Gunopulos, D., & Leymann, F. (1998). Mining Process Models from Workflow Logs. *Proceedings of the Sixth International Conference on Extending Database Technology* (EDBT).

Department of Computing and Information Systems, University of Luton, Park Sq. Luton, LU1 3JU UK, School of Computer Science & Technology, Shenyang Institute of Chemical Technology, Shenyang 110142 China
E-mail address: jing.lu@luton.ac.uk

Department of Computing and Information Systems, University of Luton, Park Sq. Luton, LU1 3JU UK
E-mail address: osei.adjei@luton.ac.uk

School of Computer Science & Technology, Shenyang Institute of Chemical Technology, Shenyang 110142 China
E-mail address: willc@mail.china.com

Department of Computing and Information Systems, University of Luton, Park Sq. Luton, LU1 3JU UK
E-mail address: fiaz.hussain@luton.ac.uk

School of Computer Science, Petru Maior University, Târgu Mureş, Romania
E-mail address: ecalin@upm.ro

School of Computer Science, Petru Maior University, Târgu Mureş, Romania
E-mail address: dumitru.radoiu@Infopulse.ro