IJICTR International Journal of Information & Communication Technology Research

Volume 7- Number 4- Autumn 2015 (65-80)

EMOSS: An Efficient Algorithm to Hide Sequential Patterns

O. Behbahani
Department of Electrical and Computer Engineering, Kharazmi University, Tehran, Iran
behbahani@khu.ac.ir

M.M. Pedram
Department of Electrical and Computer Engineering, Kharazmi University, Tehran, Iran
pedram@khu.ac.ir

K. Badie
Info Society Department, Cyberspace Research Institute, Tehran, Iran
k_badie@csri.ac.ir

B. Rahbarinia
Math and Computer Science Dept., Auburn University Montgomery, Montgomery, AL, USA
brahbari@aum.edu

Received: May 14, 2015- Accepted: July 29, 2015

Abstract—**Nowadays data mining is the way of extracting hidden knowledge from raw data whereas sequence mining aims to find sequential patterns that are frequent in the database, so publishing these data may lead to the disclosure of private information about organizations or individuals. Knowledge hiding is the process of hiding sensitive knowledge extracted previously from the database, to ensure that no abuse will be caused. This paper addresses the problem of sequential pattern hiding and proposes an efficient algorithm which uses a multi-objective approach to overcome the problem of sequence hiding as well as maintaining database fidelity as much as possible. It also shows that the proposed algorithm outperforms existing methods in terms of both speed and memory usage.**

Keywords-data mining, sequence mining, knowledge hiding, sequential pattern.

NOMENCLATURE

SDB: Sequence Database,
bestSolutions: An array associated with a sequence in SDB,which contains best solutions for sanitizing the sequence,
D: The number of distortions,
DBSeqsToCheck: The set of sequences which should be checked in the next iteration of candidate tree generation,
S: A database sequence,
nSP: The number of sensitive patterns,
NSP: The number of non-sensitive patterns,
SP: Sensitive pattern,
SPS: The set of sensitive patterns,
$ros_{sp\rightarrow s}(x:i)$: The related occurrence set of *x*, as the ith item of *SP*,in sequence *S*,
A: An item-set,
minsup(*A*): The percentage of all transactions that contain 1's for all the items in *A*,
maxsup(*A*): The percentage of transactions that contain either 1 or "?" for all the items in *A*,

$f(.,.,.)$: The sanitization objective function which evaluates the generated solutions,
λ: Hiding threshold,
α: Weight of *nSP* in the sanitization objective function,
γ: Weight of *D* in the sanitization objective function,
δ: Weight of *NSP* in the sanitization objective function,
M-pruning: A real value which serves as a threshold to prune the candidate tree,
$gapros_{SP\to S}(x{:}\,i, \min, max)$: The related occurrence set of a gap constraint that enforces the number of elements in *S*, which are between every two elements of *SP*, to be in the interval [*min-1*, *max+1*],
$disros_{SP\to S}(x{:}\,i, min, max)$: The related occurrence set of a distance constraint that enforces the number of elements in *S*, which are between the first and the last elements of *SP*, to be in the interval [*min*, *max*],
R: The complete set of rules mined from a database with respect to a minimum support threshold,
R_{SE}: The set of non-sensitive rules,
R_R: The set of restrictive rules,
R': The set equal to $R - (R_R + R_{SE})$

I. INTRODUCTION

Today, data mining methods help owners of databases extract useful knowledge from their raw data. Privacy preserving data mining is a relatively new research area in the data mining community, having existed for approximately a decade. It investigates the side effects of data mining methods originating from the penetration into the privacy of individuals and organization[7].Privacy-preserving algorithms are divided into two major groups, the first is data hiding which proposes various techniques (perturbation, transformation) for preserving the privacy of raw data and the second is knowledge hiding which involves protecting sensitive data after applying data mining techniques on raw data when the dataset is less distorted. The altered database is also called the sanitized database. So far, most sanitization methods have dealt with classic and simple forms of databases and knowledge namely, frequent item-sets and association rules, while the real-world applications are more structured data, which are named sequential data. In many applications like web usage logs, biomedical patient data, spatio-temporal geo-referenced traces and basket of customer purchasing, sequentiality of data is obvious. While extracting knowledge from these data offers several services to the world, it may be abused by competitors. Consider a medical patient database which contains clinical measurements at different moments in time. By publishing this information, rivals abuse it by sharing several databases to reveal personal information. Another example is the abuse of spatio-temporal data which contains sequences of locations left by mobile phones and other location-aware devices such as vehicular GPS [1]. Traffic management, marketing, fuel management and several other beneficial applications may also use the information extracted from mobility data, which, if published provides competitors with the opportunity to engage in user profiling, unauthorized advertising, terrorist acts, and so on.

This paper proposes a novel algorithm to hide such sensitive patterns before publishing data while maintaining most of the quality of information and data. The benefit of the proposed algorithm is twofold: it maintains the quality of and the fidelity of the data; and can reduce the computational requirements by reducing the computing time and memory usage. The proposed algorithm is based on an efficient tree pruning and shows that can improve the method previously proposed by the authors [10], through reduction of computing time and memory usage.

The rest of this paper is organized as follows. Section2 presents the literature on the sanitization field. The background information and notations are presented in Section 3. In Section 4, the proposed algorithm is introduced and the sequential pattern-hiding problem is described. In addition, the computational burden of the algorithm in terms of the time and memory usage and complexity of it is discussed in the worst case in section 4.2, and a solution to cope with these problems is proposed in the section. Section 5 presents the experimental results of two different datasets. Finally, the conclusion is presented in Section 6.

II. RELATEDWORKS

Most sanitization methods deal with classic and simple forms of databases and information, namely frequent item-sets and association rules. Several algorithms have been proposed to solve the problem of sensitive association rule hiding by manipulating support of or confidence in the rules. In the work done by Saygin et al. (2001), unknown values are introduced which define the support and confidence intervals for an item-set A and for a Rule, i.e. [*minsup*(*A*), *maxsup*(*A*)], where the *minsup*(*A*) is the percentage of all transactions that contain 1's for all the items in *A* and *maxsup*(*A*) is the percentage of transactions that contain either 1 or "?" for all the items in *A*. For rule *R* the interval is [*minconf*(*R*), *maxconf*(*R*)]. The objective is to decline a rule's support or confidence below minimum support or minimum confidence thresholds. In order to decline the minimum confidence of a rule A→B, which is defined as $minconf(A \to B) = \frac{minsup(AB)}{maxsup(A)}$, they decrease *minsup*(*AB*) and/or increase *maxsup*(*A*). Nevertheless, by replacing "?" with items in the A's or B's item-sets, the *minconf*(*A→B*) will be reduced, but it is preferable to alter B's items, because otherwise $maxconf(A \to B) = \frac{maxsup(AB)}{minsup(A)}$ might rise. Also to increase *maxsup*(*A*), "?" marks are substituted for 0's in the transactions. Reducing support of a rule is trivial. They proposed two sets of algorithms for decreasing either support of or confidence in the rules.

In Verykios et al [12], disjoint-sensitive association rules (association rules whose constituent item-sets are disjointed) are hidden one at a time by reducing their support or confidence. Reducing the support is done by reducing a rule's antecedent or its consequent item-sets. In addition, either by increasing the support of a rule's antecedent item-set in transactions that partially support it, or by decreasing the support of the rule's consequent item-set, they decreased confidence in the rule.

Oliveira et al [9], proposed a different idea which concerns sharing association rules rather than the data, and tries to restrict the rules to be published. Let *R* be the complete set of rules mined from a database with respect to a minimum support threshold, and R_R be the restrictive rules, then the goal is to transform *R* to *R'* where sensitive rules in *R* cannot be extracted by analyzing*R'*. Clearly, by merely subtracting R_R from *R*, an adversary could infer restrictive rules. So the algorithm finds a set of non-sensitive rules: R_{SE}, and sets $R' = R - (R_R + R_{SE})$, and as a result, all inference channels are closed. In the frequent item-set graph, either by obliterating at least one subset of each leaf item-set whose corresponding sensitive rules had to be hidden, or by erasing all supersets of each non-terminal node whose rules were considered restrictive, they reached to their objectives.

Aggarwal et al [4] introduced an algorithm for protecting sensitive entries in a database. Some entries in each tuple are considered sensitive by users, and the objective is to guard them from being revealed. The correlation that exists among entries in the database alleviates the contingence of the values of hidden entries by harnessing mined association rules. So some non-sensitive entries should be erased to reduce confidence in the revealing of rules. This is called *Rule Invalidation*. Another method, termed *Rule Marginalization*, precludes guessing the values of entries by blanking out the entries in the sensitive records corresponding to the antecedent of the rules, so the rule will not fire on those records. In this algorithm, first a set of rules, named *Adversarial Rules* which can be utilized by an adversary to predict values of hidden entries, are identified, then a set of non-sensitive entries, with the help of these rules, are removed from the database. Thus, Adversarial Rules become inaccessible.

A new form of practical knowledge and approaches for hiding it were proposed by Abul et al [2]. Co-occurring frequent item-sets are a set of item-sets that appear all together in mining results, and to keep them secret it suffices to impede one of them from being divulged. In another words, one piece of information will not contain secrecy unless it comes into view simultaneously with others. Therefore, if $C = \{C_1, C_2, \ldots, C_n\}$ is the set of sensitive co-occurring frequent item-sets, the two-staged hiding process elects one item-set from each $C_k \in C$ at the first stage, then conceals it as the second stage. Four different heuristic algorithms were proposed for the first stage.

A relatively new and more applicable form of data which has been presented recently, is sequential data [5], so knowledge preserving methods should be applied to this category of data, too. Although it is more likely to represent some real world data by a sequential database, few studies focus on this topic with the possible exception of Abul[3] which introduced the first sequence-pattern hiding algorithm. In this work the authors first define the matching set which is the set of all sets with the size of sensitive patterns, which each sequence supports, then the algorithm finds the occurrences of items of sensitive patterns in each sequence. Next it sorts the database in ascending order according to matching set size, then removes all matches in the top $|SDB| - \lambda$input sequences. [10], introduced a novel method capable of low distortion and infidelity. The algorithm constructs candidate trees for each sequence, which contain all the solutions with respect to the multi-objective sequence-selection framework defined by the user, then it finds the best solution for the database and finally sanitizes the sequence. While the support of sensitive patterns is greater than a defined hiding threshold, this process iterates. The process of knowledge hiding is discussed in the next section.

III. PROBLEMSTATEMENT

In this section, some basic definitions of sequence data mining are presented and a discussion about the problem of sequential pattern hiding is provided.

***Definition1. Sequence*:** A sequence is an ordered list $S = s_1 s_2 \ldots s_l$, where each $s_i (1 \le i \le l)$ is an itemset called an element which is denoted as $(x_1 x_2 \ldots x_m)$ such that each $x_k (1 \le k \le m) \in \Sigma$, and Σ is a finite set of distinct items. A sequence $\alpha = a_1 a_2 \ldots a_n$ is called a subsequence of another sequence $\beta = b_1 b_2 \ldots b_m$ and β a super-sequence is αdenoted as $\alpha \sqsubseteq \beta$, if there exist integers $1 \le j_1 < j_2 < \cdots < j_n \le m$ such that $a_1 \subseteq b_{j1}$, $a_2 \subseteq b_{j2}$, …, $a_n \subseteq b_{jn}$. In addition, a sequence database *SDB* contains a set of sequences.

***Definition2. Support of a Sequence*:** The support of a sequence α in a *SDB* is the number of sequences in *SDB* that are super-sequences of α: $sup_{SDB}(\alpha) = |\{S \in SDB | \alpha \sqsubseteq S\}|$. A sequence α is called a sequential pattern in *SDB* if $sup_{SDB}(\alpha) \geq$*min-sup*.

Given a sequence $S = s_1 \ldots s_n$ and a subsequence $S' = s'_1 \ldots s'_m$, a set of positions $\{i_1, i_2, \ldots, i_m\}$ is called an occurrence of S' in S, if $1 \le i_1 < \cdots < i_m \le n$ and $s'_k = s_{ik}$ for each $1 \le k \le m$.

***Problem definition1. Sequential Pattern Mining*:** Given a sequence database and a minimum support threshold, the sequential pattern mining problem is to find the complete set of sequential patterns in the database.

***Definition3. Related occurrence set of an item in a sequence*:** given a sequence *S*, a sequential pattern *SP*, and an item *x* which is the i^{th} item of*SP*, the related occurrence set of *x* from *SP* in *S* encircles item numbers in *S* which corresponds to item *x* and is denoted as $ros_{sp \to s}(x{:}\, i)$ .As an example consider *s* = *bab*(*cd*)(*abd*)*bb*(*cbd*), and *sp*= (*cd*)*bb*, then the related occurrence set of 3th item from *SP*, i.e. *b*, in *S* is $ros_{(cd)bb \to s}(b{:}\, 3) = \{7, 9, 10\}$.

***Definition4. Sensitive Patterns*:** Experts determine some sequential patterns to be hid. These sequential patterns are called sensitive patterns. In addition, it is referred to items in sensitive patterns as sensitive items.

***Problem definition2. SequentialPattern Hiding*:** Given a sequence database *SDB*, a sensitive pattern set

SPS, and a hiding threshold λ, the goal is to change the *SDB* at least to hide all of the sensitive patterns in it by reducing their support to λ.

There are four important issues in the definition. First, the sequences of database should be changed. This is done by replacing some items of the element by an unknown value "?".The number of items to be changed are called *distortions*. Second, the term "at least" dictates that the distortions should be as few as possible, due to the fact that distortions decrease the quality of the data. Then, the support of sensitive patterns must be reduced exactly to λ, because excessive support diminishes the database's quality. The proposed algorithm is described in the next Section.

IV. THEPROPOSEDALGORITHM

In this section we first review some fundamental implications of our previous work which constitutes the basis of the proposed algorithm here[10].

A. MOSS algorithm

As outlined in the previous section, we need to explain how a sequential pattern vanishes from a sequence; for this purpose, all the occurrences of the sequential pattern must be cleared from the sequence. Consider the sequence *s*=(*cd*)*b*(*cd*)*bebe* and the sequential pattern *sp* = (*cd*)*bb* and *ebe*. The sanitized *s*=(*cd*)*b*(*cd*)?*e*?*e*is gained by two distortion, because $ros_{(cd)bb\to s}(b:3)=\{4,6\}$, $ros_{ebe\to s}(b:2)=\{4\}$. Note that the sequence should be sanitized with as few distortions as possible. It is worth mentioning that an optimal sanitization, namely that of hiding all occurrences of sensitive patterns in a sequence, is NP-Hard [3].

A multi-objective sequence selection framework to surmount the sensitive patterns hiding problem,was introduced in[10]. Then the the algorithm finds the best candidate solution for each sequence to sanitize the dataset and then, by comparing all the best candidates in the database, it selects the best overall candidate. It is then applied to the corresponding sequence, so the support of some sensitive patterns will reduce by one unit. This process will iterate until all sensitive pattern supports descend to the exact value of λ.

The candidate solution selection process considers the following factors [10]:

1. Number of sensitive patterns (*nSP*) to be maximized .
2. Number of distortions (*D*) of the candidate solution that needs to be minimized.
3. Number of non-sensitive patterns (*NSP*) that needs to be minimized.

Thus, the problem of hiding all sequences $s\in SPS$ in a sequential database *SDB* is defined as finding $s'\sqsubseteq s$ to be hidden and changing *SDB* into *SDB′* so that:

$$\begin{aligned} max \quad & nSP(s'), min\, D(s'), min\, NSP(s') \quad (1)\\ s.t. \quad & s' \sqsubset s\,,\ for\ all\ s \in SPS\\ & sup_{SDB'}(s) = \lambda\,, \qquad for\ all\ s \in SPS \end{aligned}$$

To solve the above multi-objective optimization problem, a weighted summation of the above objectives is introduced:

$$F\big(nSP(s'), D(s'), NSP(s')\big) = \alpha * nSP(s') - \big(\gamma * D(s') + \delta * NSP(s')\big) \quad (2)$$

Where α,γ and δ are scaling factors and belong to [0,1]. The sequence analysis process, which results in the best solution, is conducted by constructing a *candidate tree* which again was introduced completely in (Rahbarinia et al., 2010). The candidate tree is composed of all possible solutions to sanitize a sequence in such a way that each of its nodes is a solution. Note that solutions which construct the i^{th} level of the tree must be the combination of *a pair of* items from $(i\text{-}1)^{th}$level. Based on this criterion a huge number of unnecessary and useless solutions will be pruned beforehand.

B. Handling Constraints

Two types of constraints, namely max/min gap and max/min distance (sliding window), could be conveniently enforced to hide the algorithm where no change is required in the algorithm and only the definition of the related occurrence set needs reconsideration. If a sequence S contains a subsequence SP, then a max/min gap constraint demands the number of elements in S that are between every two elements of SP, to be less than (max + 1) and more than (min – 1). In this case, the related occurrence set is denoted as $gapros_{SP\to S}(x:i,min,max)$. Moreover, a Sliding window constraint states that the number of elements in the sequence that are between the first and the last elements of SP, is in the interval [min, max]. In this case the related occurrence set is denoted as $disros_{SP\to S}(x:i,min,max)$. The following definitions about the related occurrence sets are obvious:

$$gapros_{SP\to S}(x:i,min,max) \subseteq ros_{SP\to S}(x:i) \quad (3)$$

$$disros_{SP\to S}(x:i,min,max) \subseteq ros_{SP\to S}(x:i) \quad (4)$$

To generate the new related occurrence sets, the $ros_{SP\to S}(x:i)$ is computed first.Then those item numbers which do not satisfy the constraints are excluded from it. The modified algorithm, i.e., *EMOSS*, will be introduced in the next part.

C. Enhanced Multi-objective Sequence Selection (EMOSS)

The proposed algorithm aimed to reduce the time complexity as well as the memory usage of the previous work. As mentioned before, in order to hide sensitive patterns, the algorithm constructs a candidate tree. The height of the tree depends on the number of sensitive patterns. The enhancement is achieved by pruning the candidate tree. The objective function (2) is computed for each solution and is used as a measure to rank them as sanitizing candidates. It should be noted that a deeper solution in the tree offers more distortions, which leads to a lower objective function. Thus, deeper solutions in the candidate tree are of lower quality. This shows the idea of pruning the

candidate tree, i.e. during the construction of the candidate tree, the sub-tree starting from a solution will be pruned if the objective function value for the solution is lower than the current best objective value for some thresholds.

***Definition5. Measure of Pruning Or M-pruning*:**a real value which serves as a threshold to prune the candidate tree. The current best objective function value is compared with the objective function value of each solution, and the subtree starting from the solution is pruned if the difference is greater than *M-pruning*.

The mechanism of pruning is as follows:

a. The first level of tree is constructed and the best objective function value is saved as the current best objective value.
b. For the 2nd level or higher, the process of tree construction continues as described below:
   i. If the difference between the current best objective value and the new solution is less than or equal to *M-pruning*, the solution will be added to the tree.
   ii. If the difference is bigger than *M-prunnig*, prune the subtree starting at the solution.
   iii. If the objective value of the solution is better than the current best objective value, then the current best objective value is updated.

Fig.1 shows the steps of the proposed algorithm. The algorithm *DBSeqsToCheck*, which holds the *s-id* of all sequences, is used to determine which *SDB* sequences should be checked in the next iteration. The algorithm iterates until all sensitive patterns become hidden. In the first step it finds the best candidate solution for each sequence and then finds the best solution for the entire *SDB.* This solution is applied to the corresponding sequence and as a result the supports of sensitive patterns in that sequence will be reduced by one unit. At this point, those sensitive patterns which are successfully hidden will be removed from the *SPS.*

After the first iteration, only those sequences that contain hidden sensitive patterns are rechecked and the sequences are updated taking into account the fact that their candidate tree will not include the hidden sensitive patterns anymore. Other sequences' candidate trees will remain intact. Therefore, a small number of sequences are checked in each iteration.

With the respect of discussion in this section, the complexity of the algorithm in worst case will be computed as below:

In the worst case, if every sequence supports all sensitive patterns (maximum size of the candidate tree for the sequence) and each sensitive pattern has one occurrence in the sequence, then first level of the tree has the complexity of:

NSP*LSP (5)

So with respect to figure 1, the other levels have the complexity of:

(NSP-2)*(NSP^2) (6)

Finally, the complexity of the whole of the **while** loop is:

[(NSP*LSP)+(NSP-2)*(NSP^2)]*DBSize*NSP (7)

The abbreviation of LSP and NSP is described below:

LSP means Length of Sensitive Pattern and NSP points to Number of Sensitive Patterns which the sequence supports.

It is well worth mentiong that the result is for non-pruning tree. The complexity will be much less than the above result when the pruninig process is used, and the complexity is dependent to the depth of the tree which the pruning occurs.

In the proposed approach in[3], the authors sanitize selected sequences by hiding all the occurrences of sensitive patterns in them. In this method, when a sequence to be sanitized is decided upon, all the sensitive patterns are removed from it. The process of hiding all the sensitive patterns from the selected sequence may lead to the loss of the chance to sanitize the database with fewer distortions[10].

In order to illustrate the proposed algorithm, consider sequence *s=bab*(*cd*)(*abd*)*bb*(*bcd*) and its candidate tree in Fig.2 with a sensitive pattern set $SPS = \{(cd)bb, ac(ad)\}$. For the sake of simplicity the effect of *NSP* is ignored, i.e. $\delta = 0$. Other parameters are considered as $\alpha = 1, \gamma = 1$, and *M-Pruning*=1.

In Fig.2, *EMOSS* is applied to the sample sequence and the candidate tree will be described. Each solution is in the form of (*nSP*,*D*),*Obj*, where*nSP*is the number of sensitive patterns,*D* is the distortions, and "*Obj*" is the objective function value for the solution, respectively and *NSP* is ignored for simplicity.

In the first level of the tree, the best objective value is zero, thus it is saved as the current best objective value, then all the solutions that appear in the level two of the tree are evaluated at step 1.2.1 in Fig.1, and the underlined ones are pruned. Then the current best objective value is updated to 1.The final best solution of this candidate tree is solution *cc*(*2*,*1*),*1*, which is shown doubly underlined, having an object value equal to 1.The Result of applying *EMOSS* is the sanitized sequence *s=bab*(*?d*)(*abd*)*bb*(*bcd*), in which the two sensitive patterns are hidden solely by one distortion.

Algorithm: EMOSS
***INPUTS: SDB, SPS, λ, α, γ, δ, M*-pruning**
OUTPUT: sanitized SDB (SDB')

DBSeqsToCheck ← all s-ids
While |SPS| > 0
1. For each s-id in DBSeqsToCheck
   1.1. Generate solutions which consider one sensitive item for deletion, save the best objective value of the first level as the current best objective value,
   1.2. Combine solutions to generate candidate tree:
      1.2.1. If (current best objective value – objective value for the solution) > M-pruning, then prune the subtree starting from the current solution.
      1.2.2. If the objective value of the solution is better than the current best objective value, then update the current best objective value.
   1.3. *bestSolutions ← highest ranked solution in the* candidate tree
2. Find best overall solution
3. Apply best overall solution and update SDB
4. Decrement the support of affected sensitive patterns
5. For each sp in SPS
   5.1. *If sup(sp) = λ then remove sp from SPS*
6. Empty DBSeqsToCheck
7. *DBSeqsToCheck ← updated sequence's s*-id
8. *DBSeqsToCheck* ← DBSeqsToCheck + s-id of sequences which contain removed sensitive patterns
9. For each s-id in DBSeqsToCheck
   9.1. Delete the corresponding bestSolution

Fig.1: *EMOSS* Algorithm

Now suppose the number of sensitive patterns is three or more, then the candidate tree will deepen more than 2 levels and there will be opportunities for pruning, resulting in less computational burdens and memory usage.

Table 1 shows the results of applying the pruning method on two sequences, where the first one is a DNA sequence and the second is a page view of a user during a 24-hour period. In this study, *NSP* is ignored, i.e. $\delta = 0$ andother parameters are considered as $\alpha = 1, \gamma = 1$ and *M-pruning*=1. The sensitive pattern set contains 4 sensitive patterns for both sequences as

Table 1: Results for Appling the Pruning Method

Sec-ID	Sequence	Sensitive patterns (*SP*)	The number of solutions generated by *MOSS*	The number of solutions generated *EMOSS*
1	*tactagcaatacgcttgcgttcggtggttaagtatgtataatgcgcgggcttgtcgt*	*tatcac*, *gga*, *acatg*, *atacat*	30731	21
2	6,2,2,2,15,2,2,1,2,8,5,6,7,3,10,11,14,12,2,2,15,15,1,7,67,6,7,4,7,7,7,7,7,7,6,7, 7,7,7,7,7,7,7,7,7,7,7,7,7,7,7,7,7,77,7,7,7,7,7,7,7,7,7,7,7,7,7,7,7,7,7,7,7,7,7,7,7,7,7,7,7,7, 7,7,7,7,7,7,6,7,7,7,7,7,7,7,7,7,7,7,7,7,7,7,7,7,7,7,7,7,7,7,7,7,7,7,7,7,7,7,7,7,7,7,7,7,7,7,7 ,7,7,7,7,7,7,7,7,7,7,7,7,7,7,7,7,7,7,7,7,7,7,7,7,7,7,7,7,7,7,7,7,7,7,7,7,7,7,7,7,7,7,7,7,7,7, 7,7,7,7,7,7,7,7,7,7,7,7,7,7,7,7,7,7,7,7,7,7,7,7,7,7,7,7,7,7,7,7,7,8,7,6,7,7,7,7,7,11,10,7,14 ,2,12,3,1,7,7,7,6,4,14,2,2,2,2,2,2,14,14,2,14,2,14,16,14,14,14,14,14,14,14,14, 14,14,14,14,14,14,14,14,14,14,14,14,14,14,14,14,14,14,14,14,14,14,14,14,14, 14,14,14,14,16,14,14,14,14,14,14,14,14,14,14,14	(1,14,14), (1,11,1), (7,7,6) ,(1,1,2,2)	6331	33

shown in the third column. The *MOSS* algorithm generated a candidate tree with 30731 solutions, while the *EMOSS* algorithm generated a candidate tree with 21 solutions. *MOSS* and *EMOSS* generated 6331 and 33 solutions, respectively for the second sequence. The results show that the *EMOSS* algorithm pruned the tree considerably, which decreases the time complexity efficiently.

V. EXPERIMENTAL RESULTS

In this section the performance of *EMOSS* is tested on two datasets. The first dataset is the Molecular Biology dataset [6], with 106 DNA sequences of length 57. The MSNBC.com Anonymous Web dataset [8] is the second dataset which incorporates 989818 sequences over integers from 1 to 17 as its items. Each sequence is a page view of a user during a 24-hour period, and the first 5000 sequences have been considered in the experiments. These datasets are denoted as *DNA*, and *WEB* respectively. The proposed algorithm, i.e. *EMOSS*, was implemented in C# and all the experiments were conducted on a system equipped with 2.66GHz Intel core duo processor and 3MB physical memory, running the Windows XP operating system.

The *EMOSS* is compared to *MOSS* in subscection 5.1, and another study between *EMOSS* and *OSH* [3], is performed in subsection 5.2. These comparative studies were performed using the following criteria: the *number of distortions* imposed on the dataset, running-time, and *infidelity*. It is worth while mentioning that *infidelity* is a measure that encompasses those non-sensitive patterns with their support falling below the support threshold after sanitization,[10].

Information regarding the datasets is shown in Table2.The support threshold used to find frequent

patterns for each dataset sequence miner algorithm is shown in Column two of Table2, and the third column of the table shows the number of frequent patterns. Each figure is supplied with legends in the form of "*algorithm-name*,(*α,γ,δ,M-pruning*), *constraint*", where *algorithm-name* refers to the algorithm used in the test. (*α,γ,δ,M-pruning*) are the parameters used in the test of the algorithm, and *constraint* shows the settings for the constraint(s). It should be noted that the parameter *M-pruning* is defined for the *EMOSS* algorithm. Experiments showed that *M-pruning* = 0.7 is a proper value.

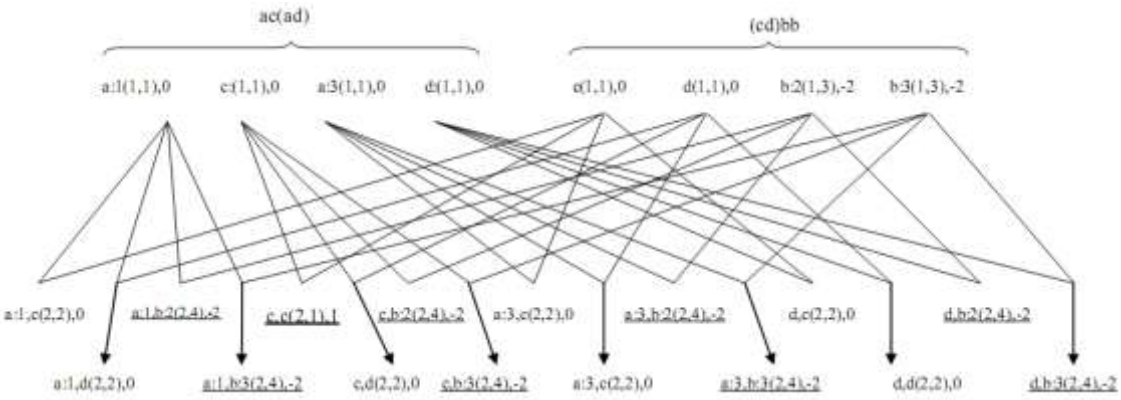


Fig.2: Candidate tree for the sample sequence with pruning

Table 2: Test Datasets[6,8]

Dataset	Support threshold	Patterns
DNA	100	720433
WEB	100	109

A. Comparisson study on EMOSS & MOSS

In this subsection, *MOSS* and *EMOSS* are compared in running-time, distortion, infidelity, and the total number of solutions produced during a run, which indicates the memory usage of each algorithm.

In Fig. 3(a)-3(d), the experiments are performed on a DNA dataset, for sets with sensetive patterns, i.e. 4SPs (*S*ensitive *P*atterns) and no constraint has been included. All experiments were performed for different values of the hiding threshold(λ). Fig.3(a) and 3(b) show that *EMOSS* performs much better than *MOSS* in terms of running-time and memory usage, due to the proposed pruning method in *EMOSS*, while the distortion and infidelity of both algorithms are the same, according to Fig.3(c)-3(d). The same experiments were repeated with two different types of constraints, i.e. min-max distance (sliding window) and min-max gap. Fig.4(a)-4(h) show the results considering the constraint of 18-20 distance, while Fig.5(a)-5(h) shows the results for the constraint of 4-9 *gap*. These Figures confirm the results obtained by Fig.3(a)-3(d). There are sudden drops in Fig. 4(a), 4(b), 4(c), 4(d), 5(b), and 5(d) for *MOSS* and *EMOSS*, and the *EMOSS* algorithm performs like the *MOSS* algorithm after some hiding thresholds. These happen because the support of some sensitive patterns becomes less than the hiding threshold for some hiding thresholds, and these sensitive patterns are ignored in the candidate tree construction. When the number of sensitive patterns reduces to 2, the depth of the candidate tree will be 2 and no further deepening happens, which means no pruning, thus *EMOSS* performs like *MOSS*. Similar experiments were conducted on the Web usage dataset, and the same results are observed in Fig.6(a)-6(h) and Fig.7(a)-7(d). It should be noted that the number of non-sensitive patterns affected by the algorithms was also considered in Fig.7(a)-7(d) by δ=1, which leads to better infidelity.

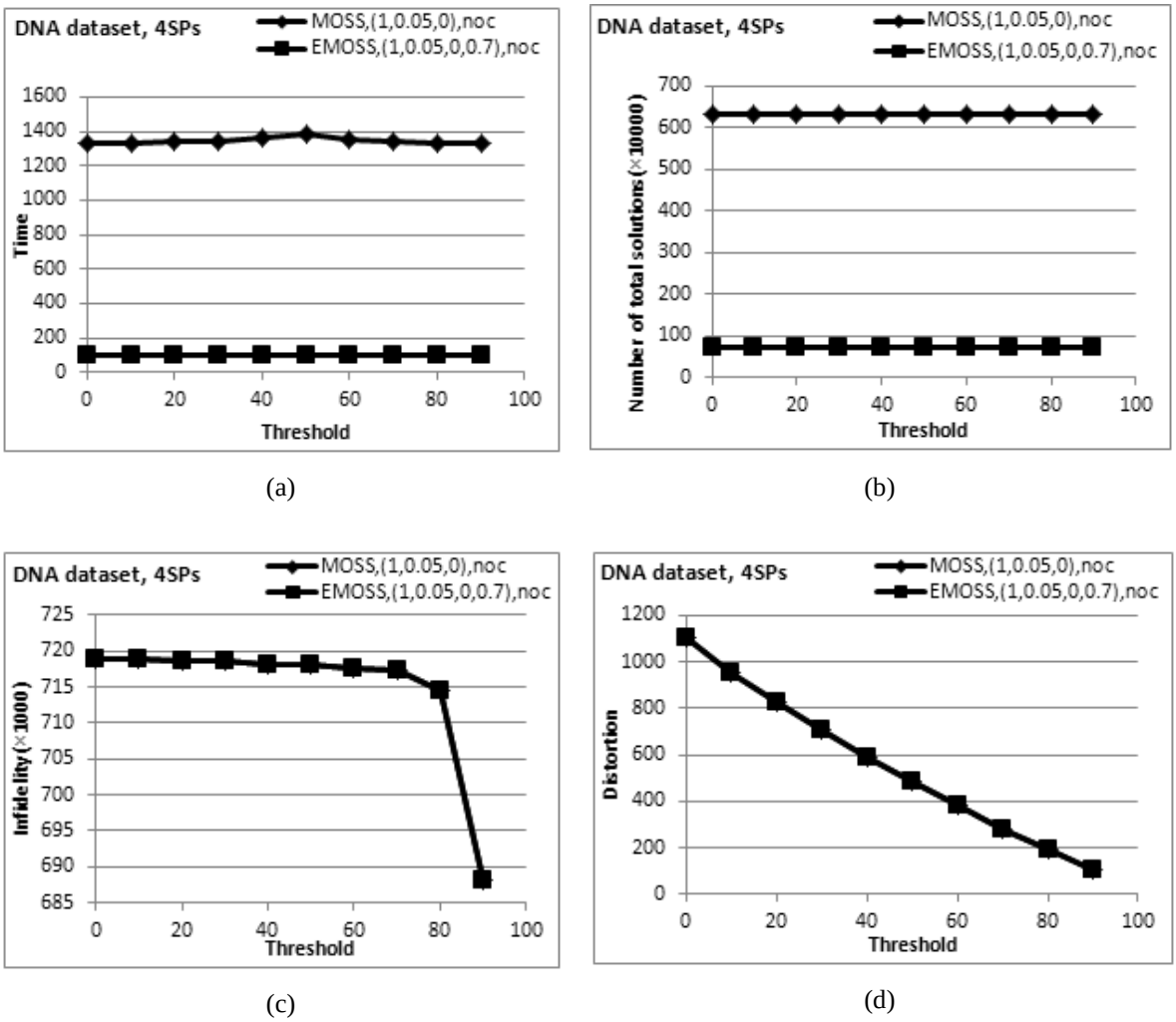
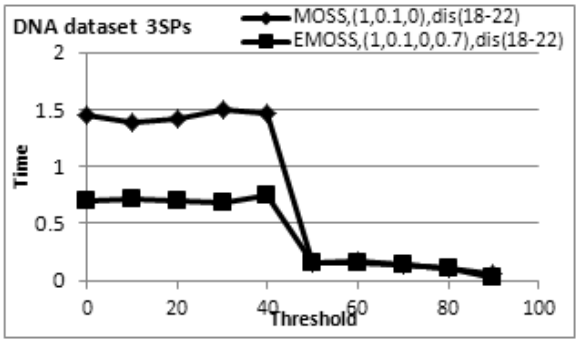
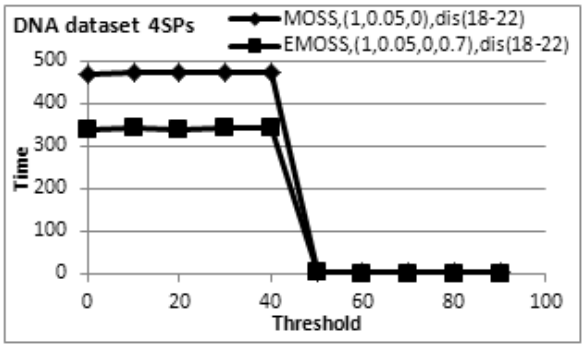


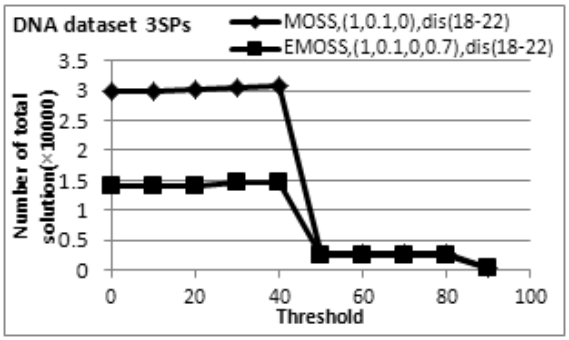
Fig.3: DNA dataset experimental results comparing *EMOSS* and *MOSS* with no constraint: (a) time for 4SPs, (b) number of total solutions for 4SPs, (c) infidelity for 4 SPs, (d) distortion for 4 SPs.



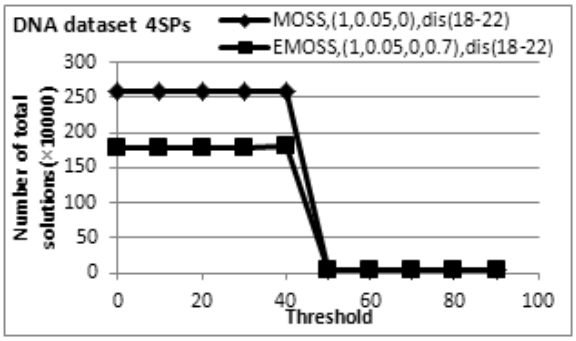
(a)



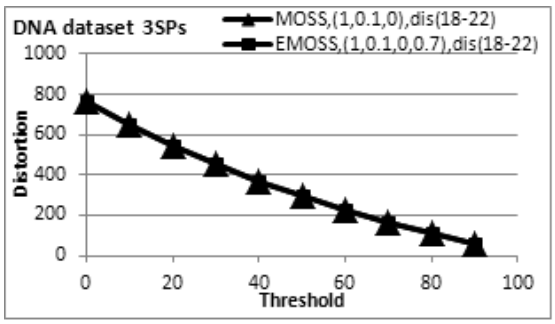
(b)



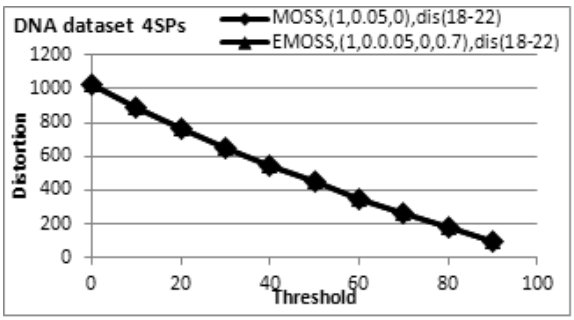
(c)



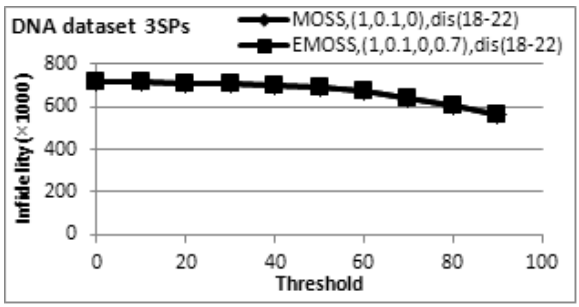
(d)



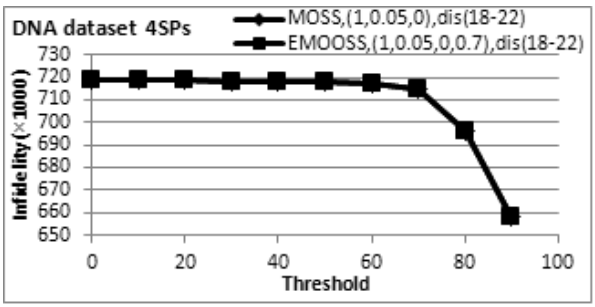
(e)



(f)

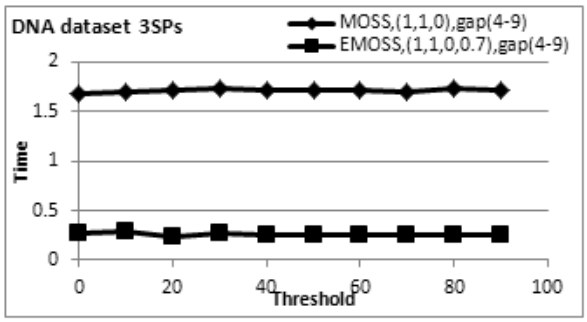


(g)

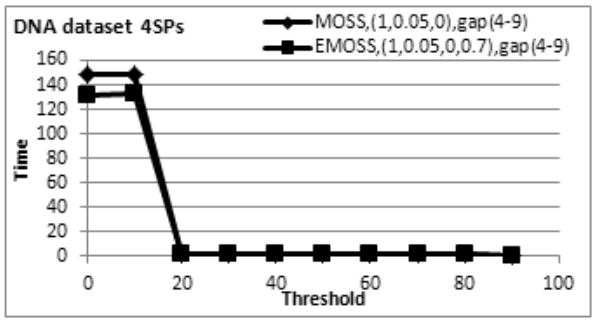


(h)

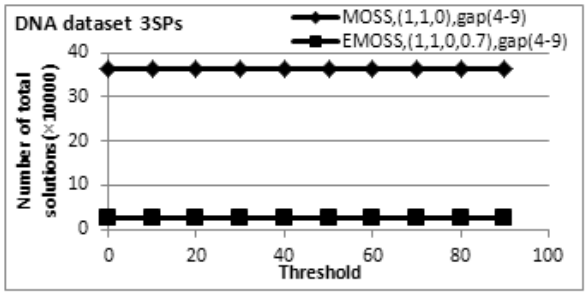
Fig.4: DNA dataset experimental results comparing EMOSS & MOSS with distance constraint:
(a) time for 3SPs,(b) time for 4SPs, (c) number of total solutions for 3SPs,
(d) number of total solutions for 4SPs,(e) distortion for 3 SPs,
(f) distortion for 4 SPs,(g) infidelity for 3 SPs,(h) infidelity for 4 SPs.



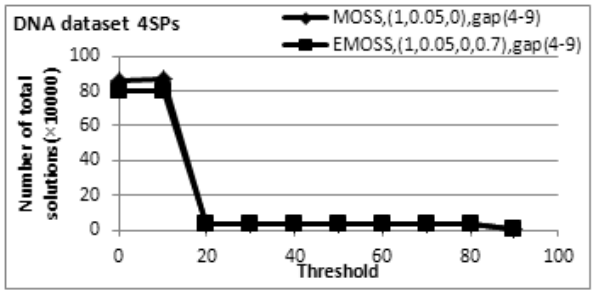
(a)



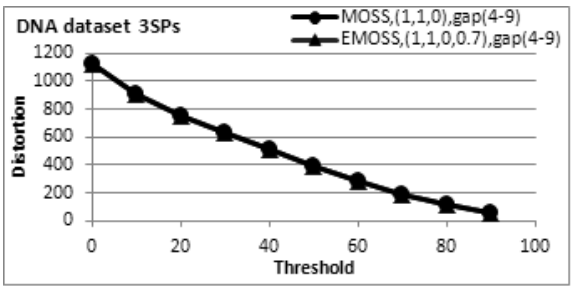
(b)



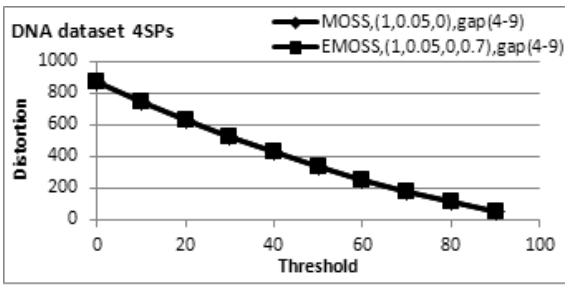
(c)



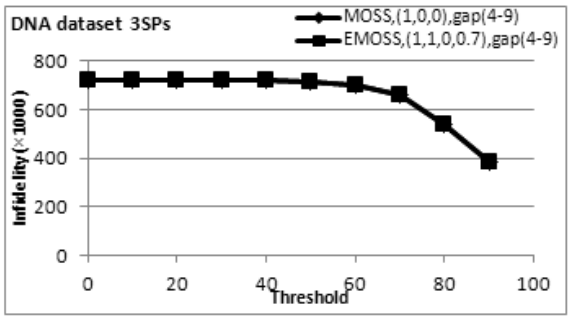
(d)



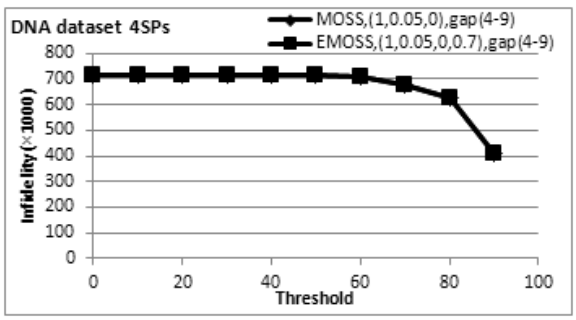
(e)



(f)

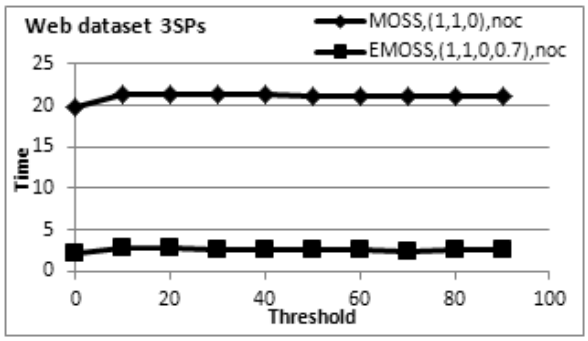


(g)

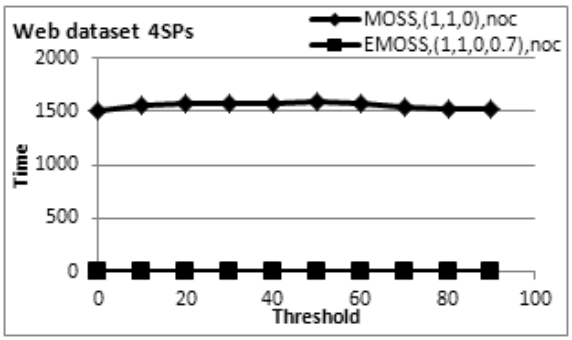


(h)

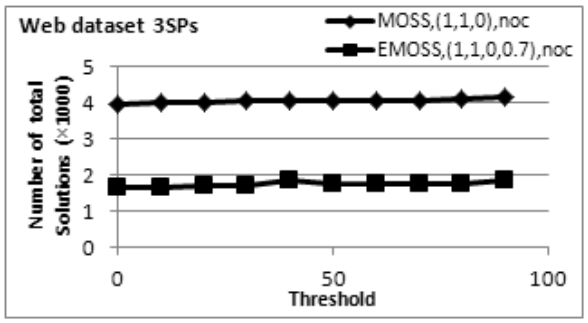
Fig.5: DNA dataset experimental results comparing EMOSS & MOSS with gap constraint:
(a) time for 3SPs,(b) time for 4SPs,(c) number of total solutions for 3SPs,
(d) number of total solutions for 4SPs,(e) distortion for 3 SPs,(f) distortion for 4 SPs,
(g) infidelity for 3 SPs,(h) infidelity for 4 SPs.



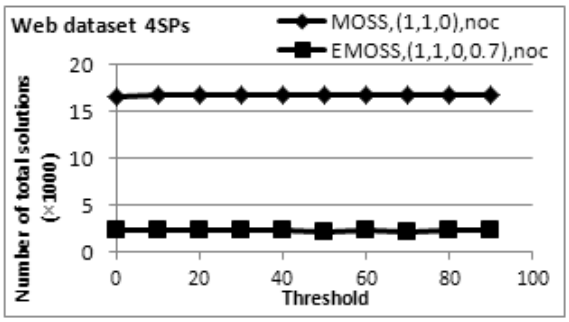
(a)



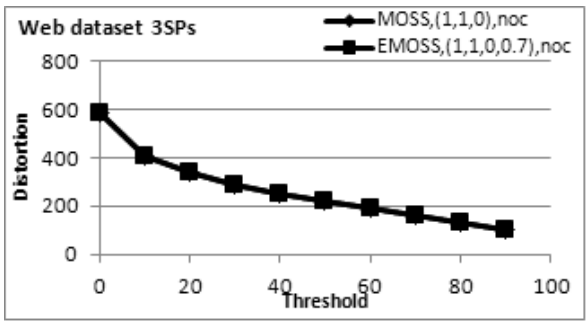
(b)



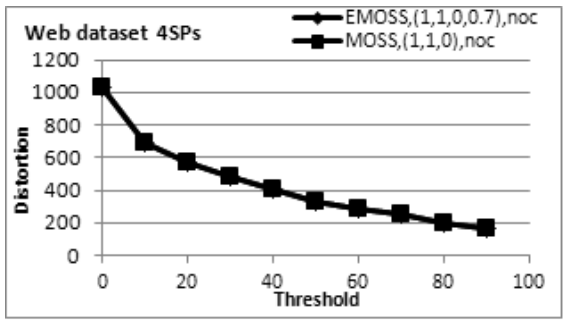
(c)



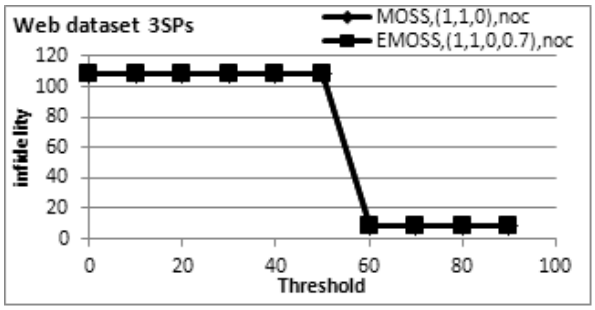
(d)



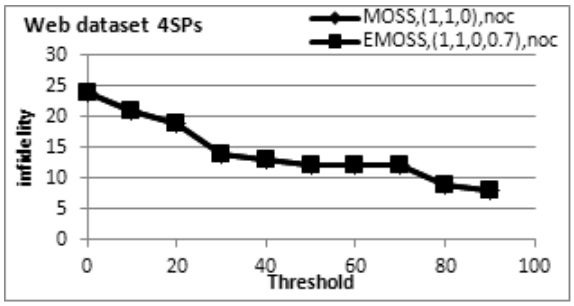
(e)



(f)



(g)



(h)

Fig.6: Web dataset experimental results, comparing MOSS & EMOSS: (a) time for 3SPs,(b) time for 4SPs, (c) number of total solutions for 3SPs,(d) number of total solutions for 4SPs,(e) distortion for 3 SPs, (f) distortion for 4 SPs,(g) infidelity for 3 SPs,(h) infidelity for 4 SPs

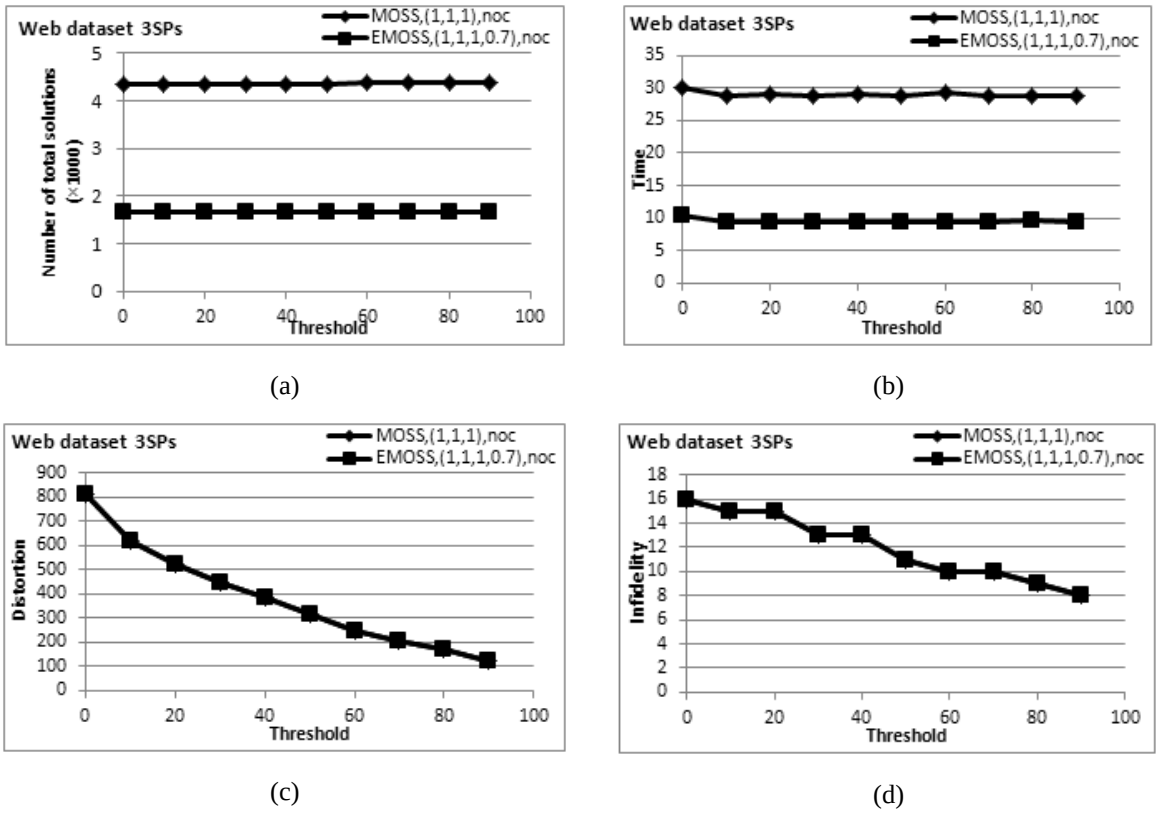


Fig.7: Web dataset experimental results, comparing MOSS & EMOSS with interfering non-sensitive patterns: (a) time for 3SPs,(b) number of total solution for 3 SPs,(c) distortion for 3SPs,(d) infidelity for 3SPs.

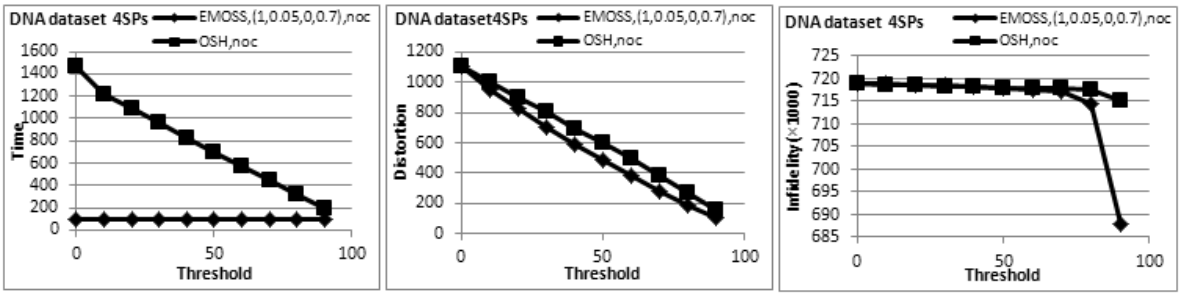


Fig. 8: DNA dataset experimental results comparing EMOSS & OSH with no constraint: (a) time for 4SPs, (b) distortion for 4SPs, (c) infidelity for 4 SPs

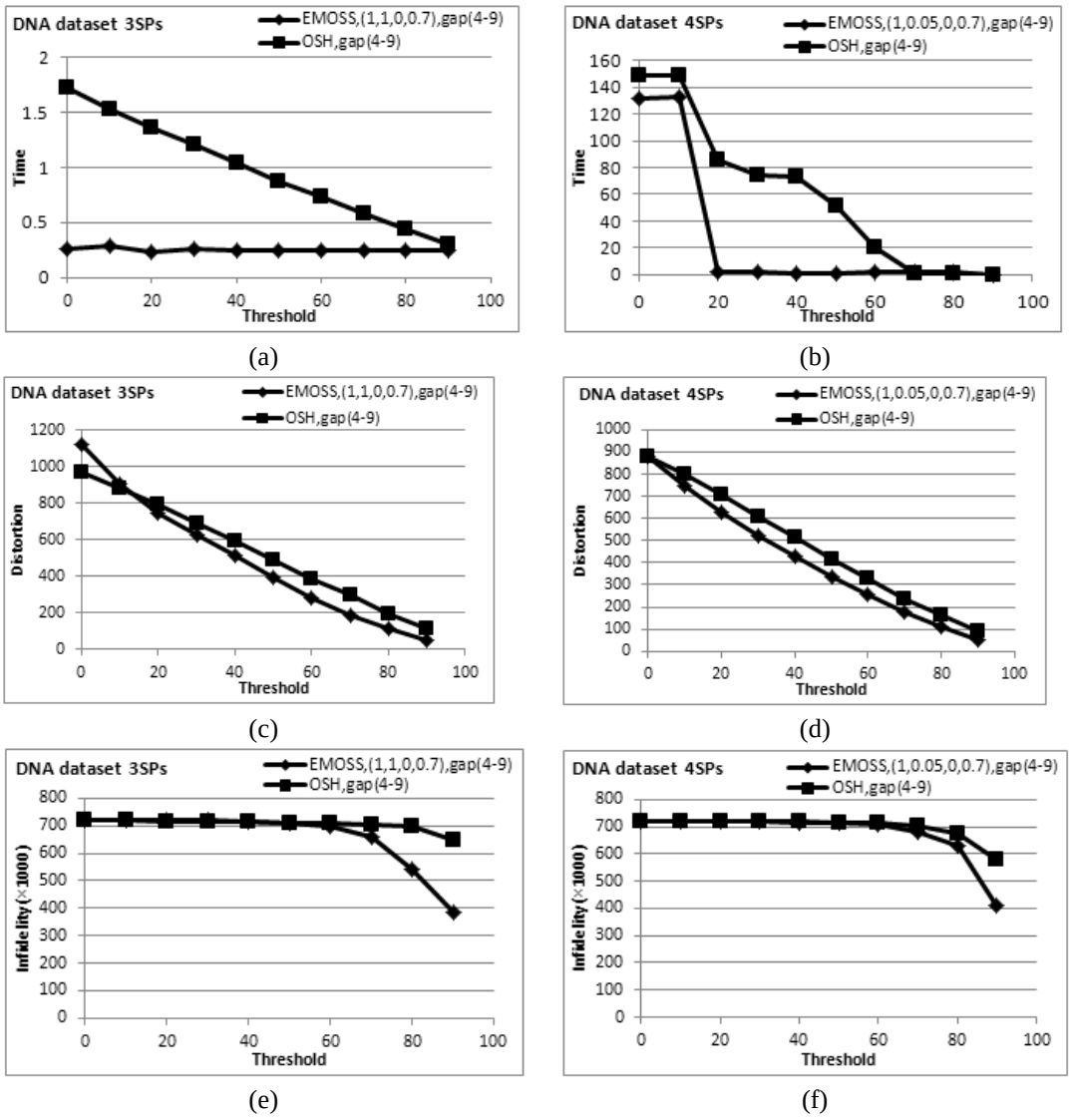


Fig. 10: DNA dataset experimental results comparing EMOSS & OSH with distance constraint: (a) time for 3SPs, (b) time for 4SPs, (c) distortion for 3SPs,(d) distortion for 4SPs, (e) infidelity for 3 SPs, (f) infidelity for 4 SPs

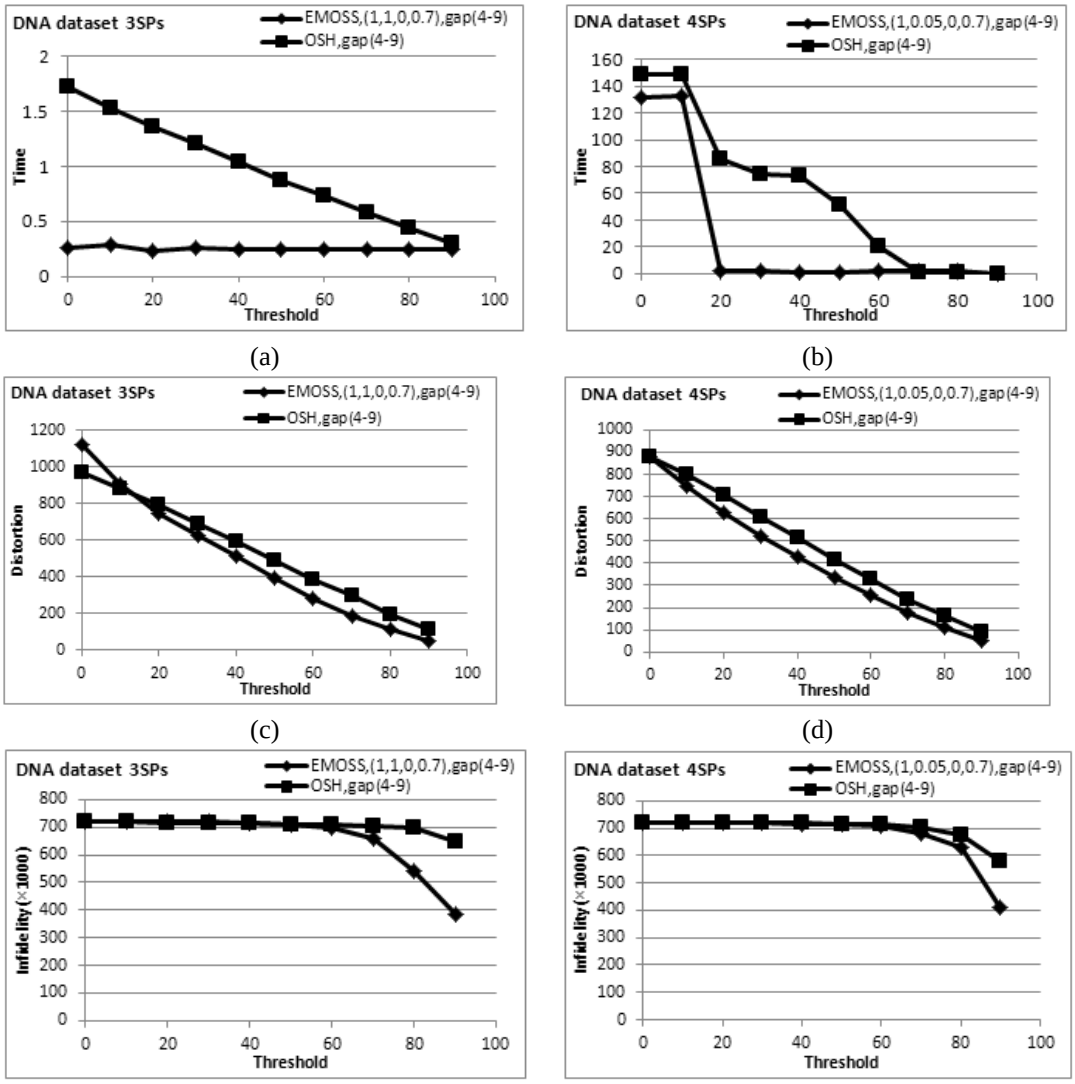


Fig. 11: DNA dataset experimental results, comparing EMOSS & OSH with gap constraint: (a) time for 3SPs, (b) time for 4SPs, (c) distortion for 3SPs, (d) distortion for 4SPs, (e)infidelity for 3 SPs, (f) infidelity for 4SPs

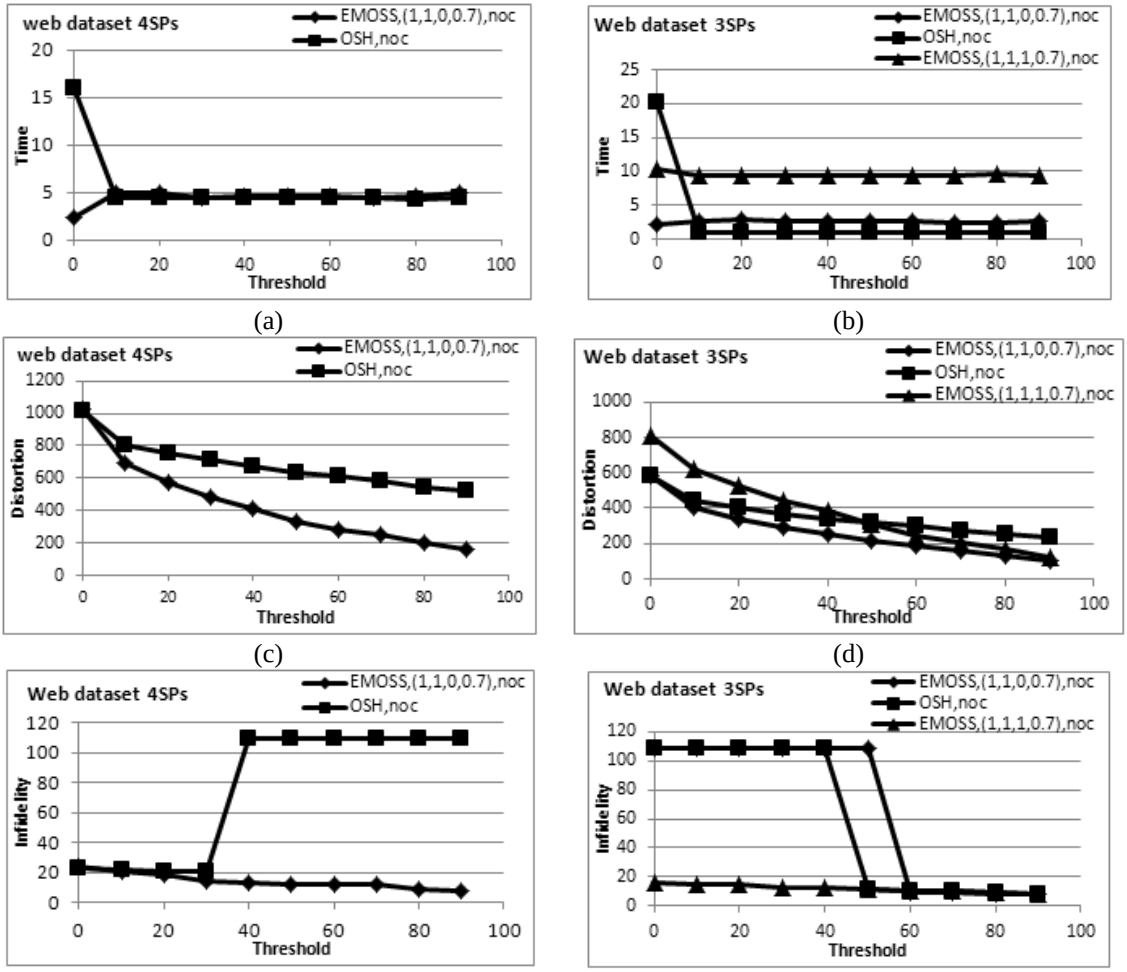


Fig. 12: Web dataset experimental results, comparing EMOSS & OSH: (a) time for 3SPs,(b) time for 4SPs,(c) distortion for 3SPs,(d) distortion for 4SPs,(e) infidelity for 4 SPs,(h) infidelity for 3 SPs

B. Comparative Study on EMOSS & OSH

In this subsection, *EMOSS* and *OSH* are compared in terms of running-time, distortion, infidelity. Fig.8(a)-8(c) show experimental results for DNA dataset with no constraint. Fig.8(b) compares the number of distortions for both algorithms and shows that *EMOSS* performs better, while *EMOSS* outperforms OSH in running-time in Fig.8(a).The running-time figure is a descending ramp for OSH, due to the removal of some sequences during the sanitization process.

Fig.8(c) shows that *EMOSS* and *OSH* are the same in terms of fidelity for most values of the hiding threshold, and *EMOSS* performs better in the higher hiding threshold.Fig.9(a)-9(f) show experimental results for the DNA dataset with distance constraint. In Fig.9(c) and 9(d), the previous result is seen again. In Fig.9(a) and 9(b), the curve of running-timeslopes down for *EMOSS* is just like that of Fig.4(b) which was mentioned earlier. Fig.9(e) and (f) show a similar performance to that seen in Fig.8(e) and 8(f).

Fig.10(a)-10(f) show experimental results for the DNA dataset with a gap constraint and similar performances are seen. Fig. 11(a)-11(f) show experimental results for the Web dataset which compares EMOSS and OSH. Fig.11(a) and 11(b) indicate that EMOSS performs better than OSH in running-time, but in Fig. 11(c) and 11(d) as well as 11(e) and 11(f), EMOSS(1,1,1,1) performs better where infidelity has been taken into account. The major reason for the drop in the running-time for OSH is that it removes some sequences from its processing list with respect to the hiding threshold.

VI. CONCLUSION

In this paper the problem of hiding sequential patterns has been addressed. The main contributions are that it proposes a more efficient algorithm with fewer distortions and lower infidelity. Furthermore, it considers a highly flexible weighted objective function to find the best solution among all candidate solutions to sanitize sequences. Handling constraints is the other advantage of the proposed method. Experimental studies based on two datasets proved the advantages of the algorithm mentioned. The results demonstrate that the proposed algorithm outperforms the other existing algorithm in terms of computing time and memory

usage. Further studies include investigating other algorithmic solutions taking into account other types of data, like multi-dimensional data which the authors are working on.

ACKNOWLEDGMENT

This research was partially supported by The Research Institute for Information and Communication Technology of Iran.

REFERENCES

[1] O. Abul, F. Bonchi, and F. Giannotti, "Hiding Sequential and Spatiotemporal Patterns," IEEE Transactions on Knowledge and Data Engineering, 22, 2010 ,pp.1709-1723.

[2] O. Abul, "Hiding co-occurring frequent itemsets," 2nd Intenational Workshop on Privacy and Anonymity in the Information Society(PAIS'09), ACM, 2009.

[3] O. Abul, M. Atzori, F. Bonchi, and F. Giannotti, "Hiding sequences," IEEE 23rd International Conference on Data Engineering Workshop(ICDEW 2007), 2007, pp. 147-156.

[4] C. C. Aggarwal, J. Pei, and B. Zhang, "On privacy preservation against adversarial data mining," 12th ACM SIGKDD international conference on Knowledge discovery and data mining, ACM, 2006.

[5] R. Aggrawal, R. Srikant, "Mining sequential patterns," 11th International Conference on Data Engineering (ICDE95), IEEE, 1994.

[6] C. Harley, R. R. Molecular Biology (Promoter Gene Sequences) Data Set UCI Machine Learning Repository, 1987.

[7] A. Gkoulalas-Divanis, V. S. Verykios, "Association Rule Hiding for Data Mining," New York,USA, Springer Verlag, 2010, pp. 45 - 52.

[8] D. Heckerman, MSNBC.com Anonymous Web Data UCI Machine Learning Repository, 1999.

[9] S. R. M. Oliveira, O. R. Zaïane, Y. Saygin, "Secure association rule sharing," 8th Pacific-Asia Coference, Advances in Knowledge Discovery and Data Mining (PAKDD'04), 2004, Springer, pp.74-85.

[10] B. Rahbarinia, M. M. Pedram, H. R. Arabnia, Z. Alavi, "A multi-objective scheme to hide sequential patterns," 2nd International Conference on Computer and Automation Engineering(ICCAE), 2010, IEEE, pp.153-158.

[11] Y. Saygin, V. S. Verykios, C. Clifton, "Using unknowns to prevent discovery of association rules," ACM SIGMOD Record, vol. 30, no. 4, 2001, pp. 45-54.

[12] V. S. Verykios, A. K. Elmagarmid, E. Bertino,Y. Saygin, and E. Dasseni, "Association rule hiding," IEEE Transaction on Knowledge and Data Engineering, 16, 2004, pp. 434-447.

Olya Sadat Behbahani received her M.Sc. degree in Artificial Intelligent (Computer Sceince) from the Kharazmi University, Tehran, Iran, 2011, and B.Sc. degree in Computer Hardware Engineering from Sanati Babol University, Mazandaran, Babol, Iran, 2004. She is currently a lecturer in the department of Computer Engineering at Islamic Azad University-North Tehran Branch. Her Research areas are Expert Systems, Machine Learning, Data Mining and Operating Systems problems.

Mir Mohsen Pedram received his Ph.D. degree in Electrical Engineering from the Tarbiat Modarres University, Tehran, Iran, 2003,his M.Sc. degree in Electrical Engineering from Tarbiat Modarres University, Tehran, Iran, 1994 and his B.Sc. degree in Electrical Engineering from Isfahan University of Technology, Isfahan, Iran, 1990. He is currently an Assistant Professor in the Department of Electrical and Computer Engineering at Kharazmi University. He is also the head of the Data Mining and Cognitive Science research laboratories at Kharazmi University. His main areas of research are Intelligent Systems, Machine Learning, Data Mining and Cognitive Science.

Babak Rahbarinia is currently an Assistant Professor in the Mathematics and Computer Science Department at Auburn University at Montgomery (AUM). He obtained his Ph.D. degree in Computer Science in May 2015 from University of Georgia. He has a M.Sc. degree in Computer Science from Azad University, Iran (2010), and a B.Sc. degree in Software Engineering from University of Science and Culture, Iran (2007). His research focuses on Computer Networks, Cyber Security, and Machine Learning.

Kambiz Badie received all his degrees from Tokyo Institute of Technology, Japan, majoring in pattern recognition. Within the past years, he has been actively involved in cognitive modeling & systemic knowledge processing in general and analogical knowledge processing, and modeling interpretation process in particular, with emphasis on creating new ideas, techniques and contents. Dr. Badie is an active researcher, in the areas of interdisciplinary and transdisciplinary studies in Iran. At present, he is a member of scientific board of IT Research Faculty, and in the meantime, Deputy Director for Research Affairs in ICT Research Institute.