A Parallelization Framework for Exact Knowledge Hiding in Transactional Databases

Aris Gkoulalas-Divanis and Vassilios S. Verykios

Abstract The hiding of sensitive knowledge, mined from transactional databases, is one of the primary goals of privacy preserving data mining. The increased storage capabilities of modern databases and the necessity for hiding solutions of superior quality, paved the way for parallelization of the hiding process. In this paper, we introduce a novel framework for decomposition and parallel solving of a category of hiding algorithms, known as *exact*. Exact algorithms hide the sensitive knowledge without any critical compromises, such as the blocking of non-sensitive patterns or the appearance of infrequent itemsets, among the frequent ones, in the sanitized outcome. The proposed framework substantially improves the size of the problems that the exact algorithms can efficiently handle, by significantly reducing their runtime. Furthermore, the generality of the framework makes it appropriate for any hiding algorithm that leads to a constraint satisfaction problem involving linear constraints of binary variables. Through experiments, we demonstrate the effectiveness of our solution on handling a large variety of hiding problem instances.

Key words: Exact knowledge hiding, Parallelization, Constraints satisfaction problems, Binary integer programming.

1 Introduction

The hiding of sensitive knowledge has attracted increasing interest over the last decade, particularly due to two main reasons: (i) the protection of the privacy of the

Aris Gkoulalas-Divanis
Department of Computer and Communication Engineering, University of Thessaly, Volos, Greece.
e-mail: arisgd@inf.uth.gr

Vassilios S. Verykios
Department of Computer and Communication Engineering, University of Thessaly, Volos, Greece.
e-mail: verykios@inf.uth.gr

individuals to whom this knowledge may refer, and (ii) the protection of business' secrets that would allow business' competitors to gain advantage over their peers. A motivating example for the latter case involves the sharing of knowledge among competing parties. Consider, for instance, a set of organizations that want to gain knowledge by collectively mining their datasets, involving a set of similar activities that they typically conduct. First, each organization mines its own data and identifies a set of knowledge patterns, some of which are classified by the data owners as sensitive, since they reveal business' secrets. Thus, prior to sharing their data, the data owners want to prohibit the leakage of the sensitive knowledge to the other parties. To accomplish that, a knowledge hiding algorithm has to be employed.

There are several categories of hiding algorithms, depending on (i) the type of data they operate on, (ii) the type of knowledge they hide, and (iii) the hiding process they enforce. Frequent itemset hiding algorithms operate on transactional data, where the sensitive knowledge is depicted in the form of frequent patterns that lead to the production of sensitive association rules. The goal of these methodologies is to create a new - hereon called *sanitized* - dataset which achieves, when mined for frequent patterns using the same (or a higher) threshold of support, to identify all the frequent patterns except from the sensitive ones. When this goal is accomplished, the attained solution is *exact*. Two exact hiding algorithms have been proposed so far [7,8]. In both algorithms, the hiding process constructs a *Constraints Satisfaction Problem* (*CSP*) and solves it by using *Binary Integer Programming* (*BIP*). However, due to the large number of constraints that are typically involved in the *CSP*, even for medium size problems, these algorithms suffer from scalability issues.

In this work, we introduce a novel framework that is suitable for decomposition and parallelization of the approaches in [7,8], and can be applied to substantially improve the scalability of both algorithms. The proposed framework aims at the decomposition of the *CSP* that is produced by the hiding algorithm, into a set of smaller *CSP*s that can be solved in parallel. First, the original *CSP* is structurally decomposed into a set of independent *CSP*s, each of which is assigned to a processor. Second, each independent *CSP* can be further decomposed into a set of dependent *CSP*s. In each step of the framework, a function is applied to question the gain of any further decomposition and allow the algorithm to take the appropriate action. Furthermore, the solutions of the various *CSP*s, produced as part of the decomposition process, can be appropriately combined to provide the solution of the initial *CSP* (prior to the decomposition). The generality of the proposed framework allows it to efficiently handle any *CSP* that consists of linear constraints involving binary variables and whose objective is to maximize (or minimize) the summation of these variables .

The rest of this paper is organized as follows. Section 2 provides the related work. In Section 3 we set out the problem of exact knowledge hiding and provide the necessary background for the understanding of the methodologies that are implemented as part of the proposed framework. The decomposition and parallelization framework is presented in Section 4. Finally, Section 5 contains the experimental evaluation, while Section 6 concludes this paper.

2 Related work

Clifton et al. [5,6] are among the first to discuss the security and privacy implications of data mining and propose data obscuring strategies to prohibit inference and discovery of sensitive knowledge. Since then, several heuristics have been proposed for the hiding of frequent patterns and the related association rules [3,10,14,15].

Menon et al. [13] present an integer programming approach for hiding sensitive itemsets. Their algorithm treats the hiding process as a *Constraints Satisfaction Problem CSP* and identifies the minimum number of transactions to be sanitized. The problem size is reduced to constraints involving only the sensitive itemsets. After solving the *CSP*, a heuristic is used to identify the transactions to be sanitized and perform the sanitization.

Sun and Yu [16] introduce a border based approach for frequent itemset hiding. The approach focuses on preserving the quality of the border constructed by the non-sensitive frequent itemsets in the lattice of itemsets. The authors use the positive border, after the removal of the sensitive itemsets, to keep track of the impact of altering transactions in the database.

Gkoulalas - Divanis and Verykios [7,8] introduce two non-heuristic algorithms for frequent itemset hiding. Both approaches use border revision to identify the candidate itemsets for sanitization. The hiding process is performed by formulating a *CSP* based on the itemsets of the borders and by solving it through *Binary Integer Programming* (*BIP*). The attained solution leads to an exact hiding of the sensitive patterns, while a heuristic approach is used when the constructed *CSP* is infeasible.

In this paper, we propose a framework that is suitable for parallelization of the exact approaches in [7,8], as well as on any hiding algorithm that is based on the construction of a *CSP* involving linear constraints of binary variables. Unlike distributed approaches [17], where the search for the CSP solution is conducted in parallel by multiple agents, our approach takes into consideration the binary nature of the CSPs to achieve a direct decomposition. Together with existing approaches for parallel mining of association rules, as in [2,9,18], our framework can be applied to parallelize the most time consuming steps of the exact hiding algorithms.

3 Notation and problem formulation

This section provides the necessary background regarding sensitive itemset hiding and allows us to proceed to our problem's formulation.

Let $\mathbf{I} = \{i_1, i_2, \ldots, i_M\}$ be a finite set of literals, called *items*, where M denotes the cardinality of the set. Any subset $I \subseteq \mathbf{I}$ is called an *itemset* over $\mathbf{I}$. A transaction T over $\mathbf{I}$ is a pair $T = (tid, I)$, where I is the itemset and tid is a unique identifier, used to distinguish among transactions that correspond to the same itemset. Furthermore, a transaction database $\mathscr{D}$ over $\mathbf{I}$ is a $N \times M$ table consisting of N transactions over $\mathbf{I}$ carrying different identifiers, where entry $T_{nm} = 1$ if and only if the m-th item appears in the n-th transaction. Otherwise, $T_{nm} = 0$. A transaction $T = (tid, J)$ supports an itemset I over $\mathbf{I}$, if $I \subseteq J$. Given a set of items S, let $\wp(S)$ denote the powerset of S, which is the set of all subsets of S.

Given an itemset I over $\mathbf{I}$ in $\mathscr{D}$, we denote by sup$(I, \mathscr{D})$ the number of transactions $T \in \mathscr{D}$ that support I. Moreover, we define the *frequency* of an itemset I in a database $\mathscr{D}$, denoted as freq$(I, \mathscr{D})$, to be the fraction of transactions in $\mathscr{D}$ that support I. An itemset I is *large* or *frequent* in database $\mathscr{D}$, if and only if, its frequency in $\mathscr{D}$ is at least equal to a minimum threshold minf. Equivalently, I is large in $\mathscr{D}$, if and only if $\text{sup}(I, \mathscr{D}) \geq \text{msup}$, where $\text{msup} = \text{minf} \times N$. All itemsets with frequency less than minf are *infrequent*.

Let $\mathscr{F}_{\mathscr{D}} = \{I \subseteq \mathbf{I} : \text{freq}(I, \mathscr{D}) \geq \text{minf}\}$ be the frequent itemsets in $\mathscr{D}$ and $\mathbf{P} = \wp(\mathbf{I})$ be the set of all patterns in the lattice of $\mathscr{D}$. The *positive* and the *negative* borders of $\mathscr{F}_{\mathscr{D}}$ are defined as $\mathscr{B}^+(\mathscr{F}_{\mathscr{D}}) = \{I \in \mathscr{F}_{\mathscr{D}} |$ for all $J \in \mathbf{P}$ with $I \subset J$ we have that $J \notin \mathscr{F}_{\mathscr{D}}\}$ and $\mathscr{B}^-(\mathscr{F}_{\mathscr{D}}) = \{I \in \mathbf{P} - \mathscr{F}_{\mathscr{D}} |$ for all $J \subset I$ we have that $J \in \mathscr{F}_{\mathscr{D}}\}$.

Based on the notation presented above, we proceed to the problem statement for the exact hiding algorithms. In what follows, we assume that we are provided with a database $\mathscr{D}_{\mathscr{O}}$, consisting of N transactions, and a threshold minf set by the owner of the data. After performing frequent itemset mining in $\mathscr{D}_{\mathscr{O}}$ with minf, we yield a set of frequent patterns, denoted as $\mathscr{F}_{\mathscr{D}_{\mathscr{O}}}$, among which a subset $\mathbf{S}$ contains patterns which are considered to be *sensitive* from the owner's perspective.

$$\textbf{maximize}\left(\sum_{u_{nm} \in U} u_{nm}\right)$$

$$\textit{subject to} \begin{cases} \sum_{T_n \in D_{\{X\}}} \prod_{I_m \in X} u_{nm} < sup(I, \mathscr{D}), \forall X \in \mathbf{S} \\ \sum_{T_n \in D_{\{R\}}} \prod_{I_m \in R} u_{nm} \geq sup(I, \mathscr{D}), \forall R \in \mathbf{V} \end{cases}$$

Fig. 1 The **C**onstraints **S**atisfaction **P**roblem for the inline approach of [7].

Given the set of sensitive itemsets $\mathbf{S}$, we define the set $\mathbf{S}_{min} = \{I \in \mathbf{S} |$ for all $J \subset I, \quad J \notin \mathbf{S}\}$ that contains all the *minimal* sensitive itemsets from $\mathbf{S}$, and the set $\mathbf{S}_{max} = \{I \in \mathscr{F}_{\mathscr{D}_{\mathscr{O}}} | \exists J \in \mathbf{S}_{min}, J \subseteq I\}$ that contains all the itemsets of $\mathbf{S}_{min}$ along with their frequent supersets. The goal of an exact hiding algorithm is to construct a new, sanitized database $\mathscr{D}$, which achieves to protect the sensitive itemsets from disclosure, while leaving intact the non-sensitive itemsets existing in $\mathscr{F}_{\mathscr{D}_{\mathscr{O}}}$. Thus, when the sanitized dataset $\mathscr{D}$ is mined, the frequent patterns that are discovered are *exactly* those in $\mathscr{F}'_{\mathscr{D}} = \mathscr{F}_{\mathscr{D}_{\mathscr{O}}} - \mathbf{S}_{max}$. This set is called *ideal*, as it pertains to an optimal hiding solution. When constructed, database $\mathscr{D}$ can be safely released since it protects sensitive knowledge. In the case of the inline approach, an exact solution is attained when the status (frequent vs infrequent) of the itemsets in $\mathbf{C} = \{I \in \mathscr{B}^+(\mathscr{F}'_{\mathscr{D}}) : I \cap \mathbf{I}^{\mathbf{S}} \neq \varnothing\} \cup \mathbf{S}$ is properly controlled. This, is achieved by solving the *CSP* of Fig. 1, where $\mathbf{V} = \{I \in \mathscr{B}^+(\mathscr{F}'_{\mathscr{D}}) : I \cap \mathbf{I}^{\mathbf{S}} \neq \varnothing\}$, $D_{\{I\}}$ denotes the set of supporting transactions for an itemset I and u_{nm} corresponds to the m-th item of the n-th transaction, while in the sanitization process.

4 A parallelization framework for exact knowledge hiding

Performing knowledge hiding by using the inline or the hybrid approach allows for the identification of exact solutions, whenever such solutions exist. However, the cost of identifying an exact solution is high due to the solving of the involved *CSP*s.

In this section, we propose a framework for decomposition and parallel solving that can be applied as part of the sanitization process of exact hiding algorithms. Our proposed framework operates in three phases, namely (i) the *structural decomposition* phase, (ii) the *decomposition of large individual components* phase, and (iii) the *parallel solving* of the produced *CSP*s. In what follows, we present the details that involve each phase of the framework.

4.1 Structural decomposition of the original CSP

The number of constraints in a *CSP* can be very large depending on the database properties, the minimum support threshold used, and the number of sensitive itemsets. Moreover, the fact that various initial constraints may incorporate products of u_{nm} variables, thus have a need to be replaced by numerous linear inequalities (using the *CDR* approach of [7]), makes the whole *BIP* problem tougher to solve. There is, however, a nice property in the *CSP*s that we can use to our benefit. That is, decomposition.

Based on the divide and conquer paradigm, a decomposition approach allows us to divide a large problem into numerous smaller ones, solve these new subproblems independently, and combine the partial solutions to attain the exact same overall solution. The property of the *CSP*s which allows us to consider such a strategy lies behind the optimization criterion that is used. Indeed, one can easily notice that the criterion of maximizing (equiv. minimizing) the summation of the binary u_{nm} variables is satisfied when as many u_{nm} variables as possible are set to one (equiv. zero). This, can be established independently, provided that the constraints that participate in the *CSP* allow for an appropriate decomposition. The approach we follow for the initial decomposition of the *CSP* is similar to the decomposition structure identification algorithm presented in [13], although applied in a "constraints" rather than a "transactions" level. As demonstrated on Fig. 2, the output of structural decomposition, when applied on the original *CSP*, is a set of smaller *CSP*s that can be solved independently. An example will allow us to better demonstrate how this process works.

Consider database $\mathscr{D}_{\mathscr{O}}$ of Fig. 3(a). Performing frequent itemset mining in $\mathscr{D}_{\mathscr{O}}$ using frequency threshold minf = 0.3, we compute the following set of large itemsets: $\mathscr{F}_{\mathscr{D}_{\mathscr{O}}} = \{A, B, C, D, AB, CD\}$. Suppose that we want to hide the sensitive item-

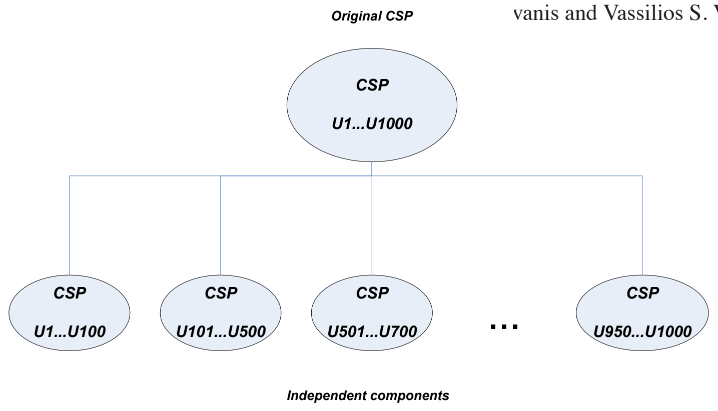

Fig. 2 Decomposing large *CSP*s to numerous independent components.

A	B	C	D
1	1	0	0
1	1	0	0
1	0	0	0
1	1	0	0
0	1	0	1
1	0	1	1
0	0	1	1
0	0	1	1
1	0	0	0
0	0	0	1

(a) Original database $\mathscr{D}_{\mathscr{O}}$.

A	B	C	D
1	u_{12}	0	0
1	u_{22}	0	0
1	0	0	0
1	u_{42}	0	0
0	u_{52}	0	1
1	0	u_{63}	u_{64}
0	0	u_{73}	u_{74}
0	0	u_{83}	u_{84}
1	0	0	0
0	0	0	1

(b) Intermediate form of $\mathscr{D}_{\mathscr{O}}$.

Fig. 3 The original and the intermediate form of database $\mathscr{D}_{\mathscr{O}}$ used in the example.

sets in $\mathbf{S} = \{B,CD\}$ using, for instance, the inline approach [1]. Then, we have that: $\mathbf{S}_{max} = \{B,AB,CD\}, \mathscr{B}^{+}(\mathscr{F}'_{\mathscr{D}}) = \{A,C,D\}$, and $\mathbf{V} = \{C,D\} \subset \mathscr{B}^{+}(\mathscr{F}'_{\mathscr{D}})$.

The intermediate form of this *CSP* is shown in Fig. 3(b) and its *CSP* formulation in Fig. 4 (left). Table 1 presents the various constraints c_r along with the variables that they control. As we can observe, we can cluster the various constraints into disjoint sets based on the variables that they involve. In our example, we can identify two such clusters of constraints, namely $\mathbf{M_1} = \{c_1\}$, and $\mathbf{M_2} = \{c_2, c_3, c_4\}$. Notice that none of the variables in each cluster of constraints is contained in any other cluster. Thus, instead of solving the entire problem, we can solve the two sub-problems shown in Fig. 4 (right), yielding, when combined, the same solution as the one of the initial *CSP*: $u_{12} = u_{22} = u_{42} = u_{63} = u_{64} = u_{73} = u_{74} = u_{83} = 1$ and $u_{52} = u_{84} = 0$.

[1] We need to mention that it is of no importance which methodology will be used to produce the *CSP*, apart from the obvious fact that some methodologies may produce *CSP*s that are better decomposable than those constructed by other approaches. However, the structure of the *CSP* also depends on the problem instance and thus it is difficult to know in advance which algorithm is bound to produce a better decomposable *CSP*.

$$\textbf{maximize}\ (u_{12}+u_{22}+u_{42}+u_{52}+u_{63}+u_{64}+u_{73}+u_{74}+u_{83}+u_{84})$$

$$\textit{subject to}\begin{cases} u_{12}+u_{22}+u_{42}+u_{52}<3 \\ u_{63}u_{64}+u_{73}u_{74}+u_{83}u_{84}<3 \\ u_{63}+u_{73}+u_{83}\geq 3 \\ u_{64}+u_{74}+u_{84}\geq 1 \end{cases}$$

$$\textbf{maximize}(u_{12}+u_{22}+u_{42}+u_{52})$$

$$\textit{subject to } u_{12}+u_{22}+u_{42}+u_{52}<3$$

and

$$\textbf{maximize}(u_{63}+u_{64}+u_{73}+u_{74}+u_{83}+u_{84})$$

$$\textit{subject to}\begin{cases} u_{63}u_{64}+u_{73}u_{74}+u_{83}u_{84}<3 \\ u_{63}+u_{73}+u_{83}\geq 3 \\ u_{64}+u_{74}+u_{84}\geq 1 \end{cases}$$

Fig. 4 The original *CSP* (left) and its structural decomposition (right).

Table 1 The constraints matrix for the produced *CSP*.

	c_1	c_2	c_3	c_4
u_{12}	X			
u_{22}	X			
u_{42}	X			
u_{52}	X			
u_{63}		X	X	
u_{64}		X		X
u_{73}		X	X	
u_{74}		X		X
u_{83}		X	X	
u_{84}		X		X

4.2 Decomposition of large independent components

The structural decomposition of the original *CSP* allows one to divide the original large problem into a number of smaller subproblems which can be solved independently, thus highly reduce the runtime needed to attain the overall solution. However, as it can be noticed, both (i) the number of subproblems, and (ii) the size of each subproblem, are totally dependent on the underlying *CSP* and the structure of the constraints matrix. This fact means that there exist problem instances which are not decomposable and other instances which experience a notable imbalance in the size of the produced components. Thus, in what follows, we present two methodologies which allow us to decompose large individual components that are non-separable through the structural decomposition approach. In both schemes, our goal is to minimize the number of variables that are shared among the newly produced components, which are now *dependent*. What allows us to proceed in such a decomposition is the binary nature of the variables involved in the *CSP*s, a fact that we can use to our benefit to minimize the different problem instances that need to be solved to produce the overall solution of the initial problem.

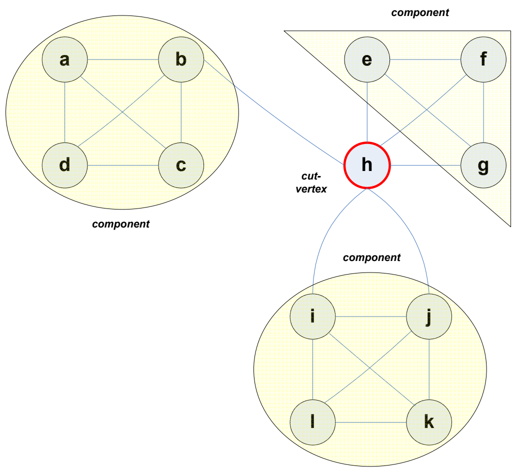

Fig. 5 An example of decomposition using articulation points.

4.2.1 Decomposition using articulation points

To further decompose an independent component we need to identify the least amount of u_{nm} variables which, when discarded from the various inequalities of this *CSP*, produce a *CSP* that is structurally decomposable. To find these u_{nm}s we proceed as follows. First, we create an undirected graph $\mathscr{G}(V,E)$ in which each vertex $v \in V$ corresponds to a u_{nm} variable, and each edge $e \in E$ connects vertexes that participate in the same constraint. Graph $\mathscr{G}$ can be built in linear time and provides us with an easy way to model the network of constraints and involved variables in our input *CSP*. Since we assume that our input *CSP* is not structurally decomposable, graph $\mathscr{G}$ will be connected.

After creating the constraints graph $\mathscr{G}$, we identify all its articulation points (a.k.a. cut-vertexes). The rationale here is that removal of a cut-vertex will disconnect graph $\mathscr{G}$ and the best cut-vertex u_{nm} will be the one that leads to the largest number of connected components in $\mathscr{G}$. Each of these components will then itself constitute a new subproblem to be solved independently from the others. To identify the best articulation point we proceed as follows. As is already known, a fast way to compute the articulation points of a graph is to traverse it by using DFS. By adding a counter to a table of vertexes each time we visit a node, we can easily keep track of the number of components that were identified so far. In the end of the algorithm, along with the identified articulation points we can have knowledge of the number of components that each of these articulation points decomposes the initial graph. This operation can proceed in linear time O($V+E$).

After identifying the best articulation point, our next step is to remove the corresponding u_{nm} variable from graph $\mathscr{G}$. Then, each component of the resulting graph corresponds to a new subproblem (i.e. a new *CSP*) that can be derived in linear time and be solved independently. To provide the same solution as the original *CSP*, the solutions of the various created subproblems need to be cross-examined, a procedure that is further explained in Section 4.3.

A final step to be addressed involves the situation in which no single cut-vertex can be identified in the graph. If such a case appears, we choose to proceed heuristically in order to experience low runtime of the algorithm. Our empirical approach is based on the premises that nodes having high degrees in graph $\mathscr{G}$ are more likely than others to correspond to cut-vertexes. For this reason, we choose to compute the degree of each vertex $u \in V$ in graph $\mathscr{G}(V,E)$ and identify the one having the maximum degree. Let $v = max_{u \in V}(degree(u))$ be the vertex whose degree is the maximum among all other vertexes in the graph. Then, among all neighbors of v we identify the one having the maximum degree and proceed to remove both vertexes from the graph. As a final step we use DFS to traverse the resultant graph to examine if it is disconnected. The runtime of this approach is linear in the number of vertexes and edges of graph $\mathscr{G}$. If the resultant graph remains connected, we choose to leave the original *CSP* as-is and make no further attempt to decompose it. Figure 5 demonstrates an example of decomposition using articulation points. In this graph, we denote as “cut-vertex”, the vertex which, when removed, leads to a disconnected graph having the maximum number of connected components (here 3).

4.2.2 Decomposition using weighted graph partitioning

One of the primary disadvantages of decomposition using articulation points is the fact that we have limited control over (i) the number of components in which our initial *CSP* will eventually split, and (ii) the size of each of these components. This fact may lead to low CPUs utilization in a parallel solving environment. For this reason, we present an alternative decomposition strategy which can break the original problem into as many subproblems as we can concurrently solve, based on our underlying system architecture. The problem formulation is once more tightly related to the graph modeling paradigm but instead of using articulation points, we rely on *graph partitioning* algorithms to provide us with the optimal split.

By assigning each u_{nm} variable of the initial *CSP* to a vertex in our undirected graph, and each constraint c to a number of edges e_c formulating a clique in the graph (while denoting the dependence of the u_{nm} variables involved), we proceed to construct a *weighted* version of the graph $\mathscr{G}$ presented in the previous section. This weighted graph, hereon denoted as $\mathscr{G}^W$, has two types of weights: one associated with each vertex $u \in V^W$, and one associated with each edge $e \in E^W$. The weight of each vertex corresponds to the number of constraints in which it participates in the *CSP* formulation. On the other hand, the weight of each edge in the graph denotes the number of constraints in which the two vertexes (it connects) appear together. Using a *weighted graph partitioning algorithm*, such as the one provided by METIS [11], one can decompose the graph into as many parts as the number of available processors that can be used to concurrently solve them. The rationale behind the applied weighted scheme is to ensure that the connectivity between ver-

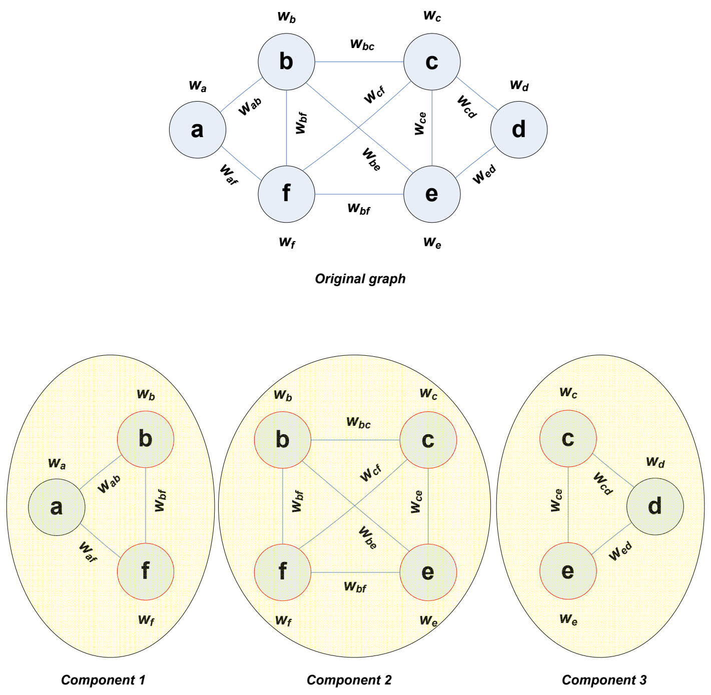

Fig. 6 An example of a three-way decomposition using weighted graph partitioning.

texes belonging in different parts is minimal. Figure 6 demonstrates a three-way decomposition of the original *CSP*, using weighted graph partitioning.

4.3 Parallel solving of the produced CSP*s*

Breaking a dependent *CSP* into a number of components (using one of the strategies mentioned earlier) is a procedure that should incur only if the *CSP* is large enough to worth the cost of decomposition. For this reason, it is necessary to define a function $\mathbf{F_S}$ to calculate the size of a *CSP* and a threshold, above which the *CSP* should be decomposed. We choose function $\mathbf{F_S}$ to be a weighted sum of the number of binary variables involved in the *CSP* and the associated constraints $\mathbf{C}$. The weights are problem-dependent. Thus, $\mathbf{F_S} = w_1 \times |u_{nm}| + w_2 \times |\mathbf{C}|$.

Our problem solving strategy proceeds as follows. First, we apply structural decomposition on the original *CSP* and we distribute each component to an available processor. These components can be solved independently of each other. The final solution (i.e. the value of the objective for the original *CSP*) will equal the sum of the values of the individual objectives; thus, the master node that holds the original *CSP* should wait to accumulate the solutions returned by the servicing nodes.

Each servicing node applies the function $\mathbf{F_S}$ to its assigned *CSP* and decides whether to further decompose it. To decompose the *CSP*, it uses one of schemes presented earlier and then assigns each of the newly created *CSP*s to an available processor. A mechanism that keeps track of the jobs distribution to processors and their status (i.e. idle vs occupied) is essential to allow for the best possible CPUs utilization. The same process continues until all *CSP*s are below the size threshold and therefore do not need further decomposition.

The handling of dependent *CSP*s by the servicing nodes, is complex. Let *border* u_{nm} be a variable that appears in two or more dependent *CSP*s. This means that this variable was either the best articulation point selected by the first strategy, or a vertex that was at the boundary of two different components, identified by using the graph partitioning algorithm. Border variables need to be checked for all possible values they can attain in order to provide us with the exact same solution as the one of solving the independent *CSP*. Suppose that p such variables exist. Then, we have 2^p possible value assignments. For each possible assignment, we solve the corresponding *CSP*s. In the objective functions of these *CSP*s, apart from the u_{nm} variables for the non-border cases, we include the values of the currently tested assignment for the p variables. After solving the *CSP*s for each assignment, we sum up the resulting objective values. The final solution will correspond to the maximum value among the different summations produced by the possible assignments[2].

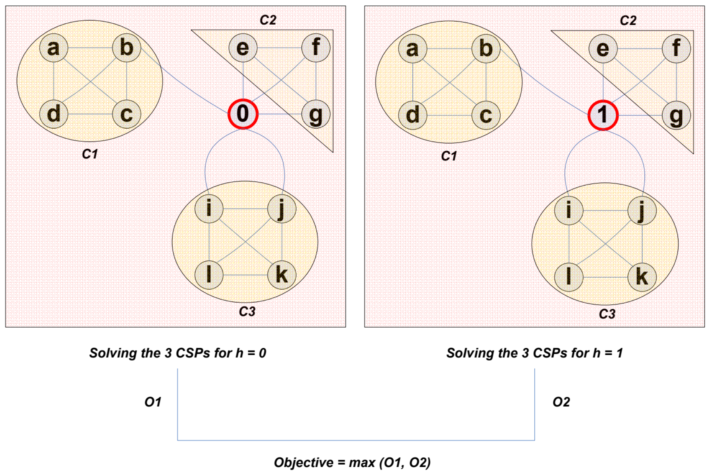

Fig. 7 An example of parallel solving after the application of a decomposition technique.

To make matters clearer, assume that an independent *CSP* is decomposed into two dependent *CSP*s, C_A and C_B, based on two border variables: u_1 and u_2. Since all possible assignments of u_1, u_2 should be tested, we have four different instances, namely: $C_{00}, C_{01}, C_{10}, C_{11}$. C_{xy} indicates the problem instance where $u_1 = x$ and $u_2 = y$; the rest variables' assignments remain unknown. Given two processors, the

[2] The proof of this statement was skipped due to the size limitations of the paper.

first solves the four instances for C_A, whereas the second one solves them for C_B. Suppose that the objective values for $C_{A,00}$ and $C_{B,00}$ are found. The objective value for C_{00} will then be the summation of these two objectives. To calculate the overall objective value and identify the solution of the initial *CSP*, we need to identify the maximum among the objective values of all problem instances. An example of parallel solving, after the application of decomposition, is depicted in Fig. 7. As one can notice, the solution of the initial *CSP* is provided by examining, for all involved *CSP*s, the two potential values of the selected cut-vertex h (i.e. solving each *CSP* for $h = 0$ and $h = 1$). The overall objective is the maximum of the two objectives, an argument that is justified by the binary nature of variable h.

5 Computational experiments and results

In this section, we provide the results of a set of experiments that we conducted to test our proposed framework. In what follows, we present the datasets we used and the different parameters involved in the testing process (such as the minimum support threshold and the number/size of the sensitive itemsets to hide), and we provide experimental results involving the structural decomposition process, where we demonstrate the major gain in the runtime of the hiding algorithm.

To test our framework, we encompassed the inline approach to hide knowledge in three real datasets. All these datasets are publicly available through the FIMI repository[3]. Datasets BMS-WebView-1 and BMS-WebView-2 both contain click stream data from the Blue Martini Software, Inc. and were used for the KDD Cup 2000 [12]. The mushroom dataset was prepared by Roberto Bayardo (University of California, Irvine) [4]. These datasets demonstrate varying characteristics in terms of the number of transactions and items and the average transaction lengths. Table 2 summarizes them.

Table 2 The characteristics of the three real datasets.

Dataset	***N***	***M***	**Avg tlen**	**msup**
BMS-1	59,602	497	2.50	30-70
BMS-2	77,512	3,340	5.60	2-10
Mushroom	8,124	119	23.00	10-50

In all tested settings, the thresholds of minimum support were properly selected to ensure an adequate amount of frequent itemsets and the sensitive itemsets to be hidden were selected randomly among the frequent ones. We conducted several experiments trying to hide sensitive 2-itemsets, 3-itemsets, and 4-itemsets. Our source code was implemented in Perl and C and we conducted all our experiments on a PC running Linux on an Intel Pentium D, 3.2 Ghz processor equipped with 4 GB of main memory. All integer programs were solved using ILOG CPLEX 9.0 [1].

[3] Available at: `http://fimi.cs.helsinki.fi/`.

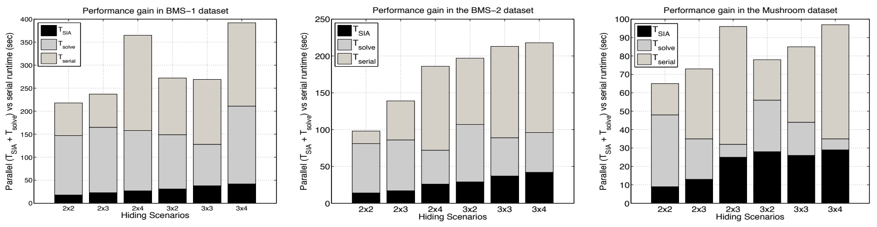

Fig. 8 Performance gain through parallel solving, when omitting the **V** part of the *CSP*.

CPLEX provides us the option of pre-solving the binary integer program, a very useful feature that allows the reduction of the *BIP*'s size, the improvement of its numeric properties (for example, by removing some inactive constraints or by fixing some non-basic variables), and also enables us to early detect infeasibility in the *BIP*'s solution. We used these beneficial properties of pre-solving to allow for early actions when solving the *CSP*s.

To conduct the experiments, we assume that we have all the necessary resources to proceed to a full-scale parallelization of the initial *CSP*. This means that if our original *CSP* can potentially break into *P* independent parts, then we assume the existence of *P* available processors that can run independently, each one solving one resultant *CSP*. Thus, the overall runtime of the hiding algorithm will equal the summation of (i) the runtime of the serial algorithm that produced the original *CSP*, (ii) the runtime of the *Structure Identification Algorithm* (SIA) that decomposed the original *CSP* into numerous independent parts, (iii) the time that is needed to communicate each of the resulting *CSPs* to an available processor, (iv) the time needed to solve the largest of these *CSP*s, (v) the communication time needed to return the attained solutions to the original processor (hereon called "master") that held the whole problem, and finally (vi) the time needed by the master processor to calculate the summation of the objective values returned in order to compute the overall solution of the problem. That is:

$$T_{\text{overall}} = T_{\text{HA}} + T_{\text{SIA}} + T_{\text{spread}} + T_{\text{solve}} + T_{\text{gather}} + T_{\text{aggregate}}$$

In the following experiments, we capture the runtime of (ii) and (iv), namely T_{SIA} and T_{solve}, since we consider both the communication overhead ($T_{\text{spread}} + T_{\text{gather}}$) and the overall solution calculation overhead ($T_{\text{aggregate}}$) to be negligible when compared to these run times. Moreover, the runtime of (i) does not change in the case of parallelization and therefore its measurement in these experiments is of no importance. To allow us compute the benefit of parallelization, we include in the results the runtime T_{serial} of solving the entire *CSP* without prior decomposition.

In our first set of experiments (presented in Fig. 8), we ensure the breaking of the original *CSP* into a controllable number of components by excluding all the constraints involving itemsets from set **V** (see Fig. 1). Thus, to break the original *CSP* into *P* parts, one needs to identify *P* mutually exclusive (in the universe of items) itemsets to hide. However, based on the number of supporting transactions for each

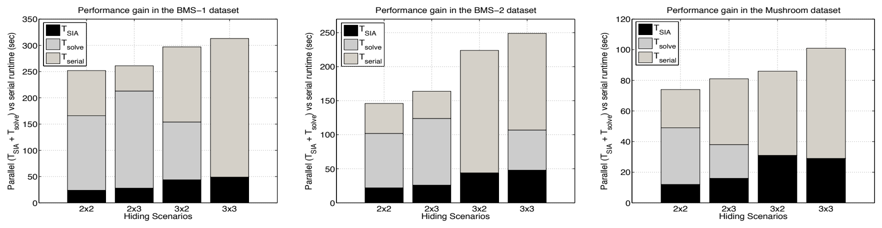

Fig. 9 Performance gain through parallel solving of the entire *CSP*.

of these itemsets in $\mathscr{D}_{\mathscr{O}}$, the size of each produced component may vary significantly. As one can observe in Fig. 8, the time that was needed for the execution of the *SIA* algorithm and the identification of the independent components is low when compared to the time needed for solving the largest of the resulting *CPS*s. Moreover, by comparing the time needed for the serial and the one needed for the parallel solving of the *CSP*, one can notice how beneficial is the decomposition strategy in reducing the runtime that is required by the hiding algorithm. For example, in the 2×2 hiding scenario for BMS-1, serial solving of the *CSP* requires 218 seconds, while parallel solving requires 165 seconds. This means that by solving the *CSP* in parallel using two processors, we reduce the solution time by 53 seconds.

In our second set of experiments, shown in Fig. 9, we included the **V** part of the *CSP*, produced by the inline algorithm. As one can observe, there are certain situations in which the original *CSP* cannot be decomposed ($T_{\text{solve}} = 0$). In such cases, one has to apply either the decomposition approach using articulation points or the weighed graph partitioning algorithm, in order to parallelize the hiding process.

6 Conclusions

In this paper, we introduced a framework for decomposition and parallel solving that is suitable for exact knowledge hiding. The proposed framework uses structural decomposition to partition the original *CSP* into independent components. Then, it offers two novel approaches for further breaking of these components into a set of dependent *CSP*s. By exploiting the features of the objective function, we provided a way of joining the partial solutions of the *CSP*s and deriving the overall hiding solution. Finally, through experimental evaluation on three real datasets, we demonstrated the benefit of decomposition towards speeding up the hiding process.

7 Acknowledgments

This work was partially funded by the European Union under the FP6-IST-FET programme, Project n. FP6-14915, GeoPKDD: Geographic Privacy-Aware Knowledge

Discovery and Delivery, www.geopkdd.eu. The authors would like to thank Dimitrios Syrivelis for his valuable comments and overall support in this work.

References

1. ILOG CPLEX 9.0 User's Manual. ILOG Inc., Gentilly, France (2003)
2. Agrawal, R., Shafer, J.C.: Parallel mining of association rules. IEEE Transactions on Knowledge and Data Engineering (TKDE) **8**(1), 962–969
3. Atallah, M., Bertino, E., Elmagarmid, A., Ibrahim, M., Verykios, V.S.: Disclosure limitation of sensitive rules. In: Proceedings of the IEEE Knowledge and Data Engineering Exchange Workshop (KDEX), pp. 45–52 (1999)
4. Bayardo, R.: Efficiently mining long patterns from databases. Proceedings of the ACM SIGMOD International Conference on Management of Data (1998)
5. Clifton, C., Kantarcioğlu, M., Vaidya, J.: Defining privacy for data mining. National Science Foundation Workshop on Next Generation Data Mining (WNGDM) pp. 126–133 (2002)
6. Clifton, C., Marks, D.: Security and privacy implications of data mining. In: Proceedings of the ACM SIGMOD International Conference on Management of Data, pp. 15–19 (1996)
7. Gkoulalas-Divanis, A., Verykios, V.S.: An integer programming approach for frequent itemset hiding. In: Proceedings of the ACM Conference on Information and Knowledge Management (CIKM) (2006)
8. Gkoulalas-Divanis, A., Verykios, V.S.: A hybrid approach to frequent itemset hiding. In: Proceedings of the IEEE International Conference on Tools with Artificial Intelligence (ICTAI) (2007)
9. Han, E.H., Karypis, G., Kumar, V.: Scalable parallel data mining for association rules. In: Proceedings of the ACM SIGMOD International Conference on Management of Data, pp. 277–288 (2007)
10. Kantarcioğlu, M., Clifton, C.: Privacy-preserving distributed mining of association rules on horizontally partitioned data. IEEE Transactions on Knowledge and Data Engineering (TKDE) **16**(9), 1026–1037 (2004)
11. Karypis, G., Kumar, V.: A fast and high quality multilevel scheme for partitioning irregular graphs. SIAM Journal of Scientific Computing **20**(1), 359–392 (1998)
12. Kohavi, R., Brodley, C., Frasca, B., Mason, L., Zheng, Z.: KDD-Cup 2000 organizers' report: Peeling the onion. SIGKDD Explorations **2**(2), 86–98 (2000)
13. Menon, S., Sarkar, S., Mukherjee, S.: Maximizing accuracy of shared databases when concealing sensitive patterns. Information Systems Research **16**(3), 256–270 (2005)
14. Oliveira, S.R.M., Zaïane, O.R.: Privacy preserving frequent itemset mining. In: Proceedings of the IEEE International Conference on Privacy, Security and Data Mining (CRPITS), pp. 43–54 (2002)
15. Saygin, Y., Verykios, V.S., Clifton, C.: Using unknowns to prevent discovery of association rules. ACM SIGMOD Record **30**(4), 45–54 (2001)
16. Sun, X., Yu, P.S.: A border-based approach for hiding sensitive frequent itemsets. In: Proceedings of the IEEE International Conference on Data Mining (ICDM), pp. 426–433 (2005)
17. Yokoo, M., Durfee, E.H., Ishida, T., Kuwabara, K.: The distributed constraint satisfaction problem: Formalization and algorithms. IEEE Transactions on Knowledge and Data Engineering **10**(5), 673–685 (1998)
18. Zaïane, O.R., M.El-Hajj, Lu, P.: Fast parallel association rule mining without candidacy generation. In: Proceedings of the IEEE International Conference on Data Mining (ICDM), pp. 665–668 (2001)