

Selective data outsourcing for enforcing privacy*

Valentina Ciriani,¹ Sabrina De Capitani di Vimercati,¹ Sara Foresti,¹

Sushil Jajodia,² Stefano Paraboschi,³ Pierangela Samarati¹

¹DTI - Università degli Studi di Milano - 26013 Crema, Italy
firstname.lastname@unimi.it

²CSIS - George Mason University - Fairfax, VA 22030-4444, USA
jajodia@gmu.edu

³DIIMM - Università degli Studi di Bergamo - 24044 Dalmine, Italy
parabosc@unibg.it

Corresponding author: Pierangela Samarati

DTI - Università degli Studi di Milano

Via Bramante 65 - 26013 Crema, Italy

pierangela.samarati@unimi.it

phone: +39-0373-898061, *fax:* +39-0373-898010

Abstract

Existing approaches for protecting sensitive information outsourced at external “honest-but-curious” servers are typically based on an overlying layer of encryption applied to the whole database, or on the combined use of fragmentation and encryption. In this paper, we put forward a novel paradigm for preserving privacy in data outsourcing, which departs from encryption. The basic idea is to involve the owner in storing a limited portion of the data, while storing the remaining information in the clear at the external server. We analyze the problem of computing a fragmentation that minimizes the owner’s workload, which is represented using different metrics and corresponding weight functions, and prove that this minimization problem is NP-hard. We then introduce the definition of locally minimal fragmentation that is used to efficiently compute a fragmentation via a heuristic algorithm. The algorithm translates the problem of finding a locally minimal fragmentation in terms of a hypergraph 2-coloring problem. Finally, we illustrate the execution of queries on fragments and provide experimental results comparing the fragmentations returned by our heuristics with respect to optimal fragmentations. The experiments show that the heuristics guarantees a low computation cost and is able to compute a fragmentation close to optimum.

keywords: privacy, confidentiality constraints, fragmentation, outsourcing

*A preliminary version of this paper appeared under the title “Enforcing Confidentiality Constraints on Sensitive Databases with Lightweight Trusted Clients,” in *Proc. of the 23rd Annual IFIP WG 11.3 Working Conference on Data and Applications Security (DBSec 2009)*, Montreal, Canada, July 12-15, 2009 [11] and under the title “Keep a Few: Outsourcing Data while Maintaining Confidentiality,” in *Proc. of the 14th European Symposium On Research In Computer Security (ESORICS 2009)*, Saint Malo, France, September 21-25, 2009 [13].

1 Introduction

Data outsourcing is today a well known paradigm that allows individuals and small/medium organizations to outsource their data to external servers and to delegate to them the responsibility of data storage and management. External servers, while typically relied upon for ensuring availability of data, might be trusted neither to access the content nor to fully enforce privacy protection requirements (*honest-but-curious* servers). In the last few years, the problem of outsourcing possibly sensitive data to an external honest-but-curious server has raised considerable attention and various research activities have been carried out, providing the foundation for a large future deployment of the proposed solutions. A common practice that guarantees the protection of outsourced data also from the server's eyes consists in encrypting the data before outsourcing them [16, 21]. Recent proposals are, instead, based on the observation that often what is sensitive is the association among data, and not the individual data themselves, and that therefore encrypting the whole data may be an overdue [1, 10, 12, 14]. In these proposals, the outsourced data are modeled as a relational table, which can be vertically partitioned, thus splitting sensitive associations among two or more servers instead of encrypting the involved attributes thus minimizing the use of encryption. In [1], the authors present an approach where the original relation containing the sensitive information to be outsourced is split into *two fragments* stored at *two non communicating servers*, which must not be known to each other. This limitation on the number of fragments implies that it is not always possible to protect the confidentiality of information by simply splitting attributes between the two fragments and therefore encryption may need to be applied. Furthermore, the query evaluation process requires the data owner to interact with both servers for joining (if needed) the two fragments and to decrypt the attributes possibly encrypted that need to be accessed by the query. In [10, 12, 14], the authors remove the limit on the number of fragments and present an approach where the relation to be outsourced can be split into *two or more unlinkable fragments*, which can even be stored all at the same server. Encryption is only used for protecting individual attributes whose release is not permitted. Furthermore, for query execution efficiency, attributes that are not represented in the clear within a fragment are represented in encrypted form, providing the nice property that each fragment completely represents the original relation. The consequence of this design choice is that to evaluate a query, it is sufficient to access a single fragment, thus avoiding join operations, which are quite expensive. This solution still requires the owner to possibly decrypt the attributes appearing in encrypted form in the fragment for evaluating a condition on them or for returning them to the user.

A common assumption of all the solutions above is that encryption is an unavoidable price to be paid to protect the information. However, although cryptographic tools enjoy today, in most computing platforms, a limited cost and an affordable computational complexity, encryption still carries the burden of managing keys,

which is a complex and expensive task. For instance, if the client is a lightweight device, with limited computation and storage resources, encryption and key management can be considered too expensive. In addition, even for scenarios where the cost of encryption/decryption operations and key management are considered acceptable, the execution of queries on encrypted data greatly increases the computational effort required to the DBMS, considerably impacting its applicability for real-world applications.

In this paper, we propose a paradigm shift for solving the problem of protecting outsourced data, which completely departs from encryption, thus freeing the owner from the burden of its management. In exchange, we assume that the owner, while outsourcing the major portion of the data at one or more external servers, is willing to locally store a limited amount of data. Obviously, from the information stored on the external server it should not be possible to reconstruct a sensitive association (confidentiality constraint) since otherwise privacy would be violated. Since we do not want to remove the assumption that the external servers are all honest-but-curious, the owner is the only entity in the system that can manage sensitive data. Like recent solutions, we therefore exploit data fragmentation to break sensitive associations; but, in contrast to them, we assume the use of fragmentation only. Basically, the owner maintains a small portion of the data, just enough to protect sensitive values or their associations. In [11, 13], we presented an early version of our proposal that here is extended by introducing a different modeling (based on a hypergraph coloring problem) of the problem and an original heuristics for the computation of a fragmentation that minimizes the workload of the owner. In addition, we formally analyze the correctness and computational complexity of our heuristics. We also present a set of experiments, aimed at comparing our heuristics with the exhaustive search of the optimum, to assess its efficiency, in terms of computational time, and its effectiveness, in terms of the quality of the computed fragmentation. The experimental results prove that the fragmentations computed by our heuristics well approximate the optimal solutions. We also provide a characterization of the existing paradigms for data outsourcing in the presence of confidentiality constraints.

The remainder of this paper is organized as follows. Section 2 introduces some basic concepts. Section 3 illustrates the rationale of our approach, also discussing it with respect to the previous proposals. Section 4 introduces the problem of computing a minimal fragmentation. Section 5 discusses different metrics with respect to which the workload of the owner could be minimized. Section 6 proves the NP-hardness of the minimal fragmentation problem and proposes a definition of local minimality which is used in the computation of a fragmentation. Section 7 reformulates the minimization problem as a hypergraph 2-coloring problem and presents a heuristic algorithm for its solution. Section 8 describes how queries can be executed on fragmented data. Section 9 illustrates the experimental results comparing our heuristic algorithm to an exhaustive approach. Section 10 discusses related work. Finally, Section 11 draws our conclusions.

2 Basic concepts

We consider a scenario where, consistently with other proposals (e.g., [1, 10, 15]), the data to be protected are represented with a single relation r over a relation schema $R(a_1, \dots, a_n)$. We use the standard notations of the relational database model. Also, when clear from the context, we use R to denote either the relation schema R or the set of attributes in R .

Protection requirements are represented by *confidentiality constraints*, which express restrictions on the separate or joint visibility (association) of attributes in R , and are formally defined as follows [1, 10].

Definition 2.1 (Confidentiality constraint) *Let $R(a_1, \dots, a_n)$ be a relation schema, a confidentiality constraint c over R is a subset of attributes in R , that is, $c \subseteq R$.*

While simple, confidentiality constraints of this form allow the representation of different protection requirements that may need to be expressed. A *singleton constraint* states that the *values* assumed by an attribute are considered sensitive and therefore cannot be made known to an external party. A non-singleton constraint (*association constraint*) states that the *association* among values of given attributes is sensitive and therefore cannot be made known to an external party.

Example 2.2 *Figure 1(a) illustrates relation PATIENT of a hospital. Figure 1(b) illustrates the set of confidentiality constraints defined over the relation: c_0 and c_1 are two singleton constraints indicating that the lists of SSNs and Names of patients are considered sensitive; $c_2 \dots c_5$ are association constraints stating that the associations among the values assumed by the specified attributes should not be disclosed. Constraints c_2 and c_3 derive from the fact that attributes DoB and ZIP together could be exploited to retrieve the name of patients (i.e., they can work as a quasi-identifier [15]) and therefore they cannot be released in association with sensitive attributes, that is, Illness and Treatment. Constraints c_4 and c_5 protect the sensitive associations between Job and Illness and between Illness and Treatment, respectively.*

The satisfaction of a constraint c_i clearly implies the satisfaction of any constraint c_j such that $c_i \subseteq c_j$. We assume set $\mathcal{C}_f = \{c_1, \dots, c_m\}$ to be *well defined*, that is, \mathcal{C}_f does not contain a constraint that is a subset of another constraint. Formally, $\forall c_i, c_j \in \mathcal{C}_f : i \neq j \Rightarrow c_i \not\subseteq c_j$. For instance, since Name is a sensitive attribute, no constraint needs to be applied in Example 2.2 with respect to the association between Name and sensitive attributes (i.e., Illness and Treatment).

To satisfy confidentiality constraints, we consider an approach based on data *fragmentation*. Fragmenting R means splitting its attributes into different vertical *fragments* (i.e., different subsets of attributes) in such a way that only attributes in the same fragment are visible in association [1, 10]. For instance, splitting Job and Illness into two different fragments offers visibility of the two lists of values, but not of their association. A

fragment is said to *violate* a constraint if it contains all the attributes in the constraint. For instance, a fragment containing both `Job` and `Illness` violates constraint c_4 .

3 Classification of solutions and rationale of our approach

Existing approaches for enforcing confidentiality constraints on data to be outsourced typically assume the joint application of *fragmentation* and *encryption*, and may consider additional assumptions such as the impossibility of external servers to communicate with each other. In this paper, we put forward a novel approach departing from data encryption and from the assumption that external servers do not communicate with each other. The rationale for departing from encryption is that encryption is sometimes considered a too rigid tool, delicate in its configuration, and requiring careful management to fulfill its potential. Systems protecting sensitive information based on an extensive use of encryption suffer consequences due to possible compromises of keys (disclosure of data) or their loss (denial of service). In the real world, key management, particularly the operations at the human side, is a difficult and delicate process, and this causes a preference of many ICT administrators towards designs that avoid to excessively rely on encryption. Also, while the computational cost of symmetric encryption for modern computer architectures is usually negligible, the presence of encryption often causes an increase in the computational load. This increase is due to the limitations in the access to the data, which do not allow the server to efficiently and effectively evaluate generic predicates on the data. Our solution involves the data owner in storing (and managing) a small portion of the data, while delegating the management of all other data to external servers. We consider the management of a small portion of the data to be an advantage with respect to the encryption management and computation that would otherwise be required. Figure 2 presents the basic idea of our paradigm in contrast to the two previous main lines of work [1, 10]. In the figure, E_i represents the set of attributes that are encrypted and C_i represents the set of attributes stored in the clear in fragment F_i .

- *Non-communicating pair of servers* [1]. This approach distributes the data to two external servers. It resorts to encryption whenever it is not possible to distribute the attributes among the two servers satisfying all the confidentiality constraints. In particular, all sensitive attributes are encrypted; other attributes are encrypted if placing them in any of the two fragments would violate at least one constraint. The main shortcoming of this approach is that confidentiality relies, besides on encryption, on the assumption of complete absence of communication between the two external servers, and therefore on the trust in the fact that such a communication will never happen. Such an assumption appears difficult to enforce in real scenarios.

- *Multiple fragments* [10, 12, 14]. Like [1], this approach stores all data at external servers. However, it allows data to be split among more than two fragments. Hence, associations can always be protected by placing the involved attributes in different fragments. For each fragment, all the attributes of the original relation R that do not appear in the clear in the fragment are stored in encrypted form. Protection against joining of fragments is guaranteed by the absence of common attributes in the different fragments. There is no restriction on the communication among external servers, since the complete visibility of all fragments cannot violate confidentiality constraints. As a matter of fact, all fragments could even be stored at a single server.
- *No encryption* [11, 13]. Completely departing from encryption, in this approach we assume that the owner maintains some of the data, so to avoid exposing sensitive attributes or associations externally. Sensitive attributes are maintained at the owner side. Sensitive associations are protected by ensuring that not all attributes in an association are stored externally. In other words, for each sensitive association, the owner should locally store at least one of the involved attributes. With this fragmentation, the original relation R is split into two fragments, called F_o and F_s , stored at the data owner and at the server side, respectively. To guarantee the reconstruction of the content of the original relation R (lossless join property), at the physical level the two fragments F_o and F_s must have a common key attribute. We assume F_o and F_s to have a common tuple id (attribute `tid` as in Figure 3) that can be either: 1) the key attribute of the original relation, if it is not sensitive, or 2) an attribute that does not belong to the schema of the original relation R and that is added to F_o and F_s during the fragmentation process. We consider this a physical-level property and ignore the common attribute in the fragmentation process.

4 Minimal fragmentation

Given a set \mathcal{C}_f of confidentiality constraints over relation R , our goal is to split R into two fragments: F_o , stored at the owner side, and F_s , stored at the server side, in such a way that all sensitive data and associations are protected (i.e., they are not visible at the external server). It is easy to see that, since there is no encryption, singleton constraints can only be protected by storing the corresponding attributes at the owner side. Therefore, each singleton constraint $c=\{a\}$ is enforced by inserting a into F_o and by not allowing a to appear in F_s . Association constraints are enforced via fragmentation, that is, by splitting the attributes involved in the constraint between F_o and F_s . A fragmentation $\mathcal{F}=\langle F_o, F_s \rangle$ should satisfy the following two conditions: 1) all attributes in R should appear in at least one fragment, to avoid loss of information; 2) the external fragment F_s should not violate any confidentiality constraint. Note that the second condition applies only to F_s , since F_o is stored at the owner side and remains under the owner control. F_o can therefore contain sensitive data and/or

associations. In addition to the two conditions above, we require the fragmentation to be non-redundant, that is, each attribute is stored at only one fragment. Besides preventing usual replication problems, this condition intuitively avoids unnecessary storage at the data owner (there is no need to locally maintain the information that is outsourced). We therefore include it in our definition of correct fragmentation, with a note that non-redundancy is not needed for satisfying the constraints and could therefore be relaxed. Fragmentation *correctness* is formally defined as follows.

Definition 4.1 (Fragmentation correctness) *Let $R(a_1, \dots, a_n)$ be a relation schema, $\mathcal{C}_f = \{c_1, \dots, c_m\}$ be a well defined set of confidentiality constraints over R , and $\mathcal{F} = \langle F_o, F_s \rangle$ be a fragmentation for R , where F_o is stored at the owner and F_s is stored at a server. \mathcal{F} is a correct fragmentation for R , with respect to \mathcal{C}_f , iff: 1) $F_o \cup F_s = R$ (completeness); 2) $\forall c \in \mathcal{C}_f, c \not\subseteq F_s$ (confidentiality); 3) $F_o \cap F_s = \emptyset$ (non-redundancy).*

Given a relation schema $R(a_1, \dots, a_n)$ and a set \mathcal{C}_f of confidentiality constraints, our goal is to produce a correct fragmentation that minimizes the owner's workload. For instance, a fragmentation where $F_o = R$ and $F_s = \emptyset$ is clearly correct, but it is also undesirable (unless required by the confidentiality constraints), as it leaves to the owner the burden of storing all information and of managing all possible queries.

The owner's workload may be a difficult concept to capture, since different metrics might be applicable in different scenarios (see Section 5). Regardless of the metrics adopted, we model the owner's workload as a weight function $w: \mathcal{P}(R) \times \mathcal{P}(R) \rightarrow \mathbb{R}^+$ that takes a pair $\langle F_o, F_s \rangle$ of fragments as input and returns the storage and/or the computational load at the owner side due to the management of F_o . Assume the weight function w to be *monotonic* with respect to the set containment relationship on F_o . Formally, given two fragmentations, $\mathcal{F} = \langle F_o, F_s \rangle$ and $\mathcal{F}' = \langle F'_o, F'_s \rangle$: $F_o \subseteq F'_o \Rightarrow w(\mathcal{F}) \leq w(\mathcal{F}')$. The monotonicity of the weight function is a natural property of different metrics, since the owner's workload increases whenever new attributes are added to F_o . A fragmentation that minimizes the owner's workload is formally defined as follows.

Definition 4.2 (Minimal fragmentation) *Let $R(a_1, \dots, a_n)$ be a relation schema, $\mathcal{C}_f = \{c_1, \dots, c_m\}$ be a well defined set of confidentiality constraints over R , $w: \mathcal{P}(R) \times \mathcal{P}(R) \rightarrow \mathbb{R}^+$ be a monotonic weight function, and $\mathcal{F} = \langle F_o, F_s \rangle$ be a fragmentation for R , where F_o is stored at the owner and F_s is stored at a server. \mathcal{F} is a minimal fragmentation for R , with respect to \mathcal{C}_f and w , iff: 1) \mathcal{F} is correct (Definition 4.1) and 2) $\nexists \mathcal{F}'$ such that $w(\mathcal{F}') < w(\mathcal{F})$ and \mathcal{F}' is correct.*

Given a relation schema R , a set \mathcal{C}_f of confidentiality constraints, and a weight function w , different minimal fragmentations may exist and our goal is to compute one of them, as stated by the following problem definition.

Problem 4.3 (MF problem) *Given a relation schema $R(a_1, \dots, a_n)$, a well defined set $\mathcal{C}_f = \{c_1, \dots, c_m\}$ of confidentiality constraints over R , and a monotonic weight function w , determine a minimal fragmentation*

$\mathcal{F}=\langle F_o, F_s \rangle$ for R , with respect to \mathcal{C}_f and w (Definition 4.2).

Before proceeding on the investigation of the problem, we present some possible fragmentation metrics and the corresponding weight functions.

5 Fragmentation metrics

In the outsourcing scenario, storage and computational resources offered by the external server are considered less expensive than the resources of the data owner. The owner has then a natural incentive to rely as much as possible, for storage and computation, on the external server. In the absence of confidentiality constraints, all data would then be remotely stored and all queries would be computed by the external server. In the case of confidentiality constraints, the owner internally stores some attributes, and consequently is involved in some computation.

In this section, we discuss several metrics (and corresponding weight functions to be minimized) that could be used to characterize the owner’s workload for a fragmentation, and therefore to determine which attributes are stored at the owner and which attributes are outsourced at the external server. The different metrics may be applicable to different scenarios, depending on the owner’s preferences and/or on the specific knowledge (on the data or on the query workload) available at design time. We consider four possible scenarios, in increasing level of required knowledge. The first two scenarios measure storage, while the latter two scenarios measure computation. The scenarios and corresponding weight functions are summarized in Figure 4.

- *Min-Attr.* Only the relation schema (set of attributes) and the confidentiality constraints are known. The only applicable metric aims at minimizing the storage required at the owner side by *minimizing the number of attributes* in F_o . The weight $w_a(\mathcal{F})$ of a fragmentation \mathcal{F} is the number of attributes in F_o , that is: $w_a(\mathcal{F})=card(F_o)$. For instance, given fragmentation $\mathcal{F}=\langle \{\text{SSN,Name,DoB,Job,Treatment}\}, \{\text{ZIP,Illness,HDate}\} \rangle$ illustrated in Figure 3, $w_a(\mathcal{F})=5$.
- *Min-Size.* Besides the mandatory knowledge of the relation schema and of the confidentiality constraints on it, the size of each attribute is known. In this case, it is possible to produce a more precise estimate of the storage required at the owner side, aiming at *minimizing the physical size* of F_o , that is, the actual storage required by its attributes. The weight $w_s(\mathcal{F})$ of a fragmentation \mathcal{F} is the physical size of the attributes in F_o , that is: $w_s(\mathcal{F})=\sum_{a \in F_o} size(a)$, where $size(a)$ denotes the physical size of attribute a . For instance, given fragmentation $\mathcal{F}=\langle \{\text{SSN,Name,DoB,Job,Treatment}\}, \{\text{ZIP,Illness,HDate}\} \rangle$ illustrated in Figure 3 and the data workload in Figure 5(a), $w_s(\mathcal{F}) = size(\text{SSN}) + size(\text{Name}) + size(\text{DoB}) + size(\text{Job}) + size(\text{Treatment}) = 95$.

- *Min-Query*. In addition to the relation schema and the confidentiality constraints, a representative profile of the expected query workload is known. The profile defines, for each query, the expected frequency with which the query is executed and the set of attributes appearing in the conditions in the query. The query workload profile is modeled as a set of triples $\mathcal{Q}=\{(q_1, freq(q_1), Attr(q_1)), \dots, (q_l, freq(q_l), Attr(q_l))\}$, where for each $(q_i, freq(q_i), Attr(q_i))$, $i = 1, \dots, l$, q_i is the query to be executed, $freq(q_i)$ is the expected execution frequency of q_i , and $Attr(q_i)$ are the attributes appearing in the WHERE clause of q_i . The first three columns of Figure 5(b) illustrate a possible workload profile for relation PATIENT in Figure 1(a). Knowledge on the workload allows the adoption of a metric evaluating the computational work required to the owner for executing queries. Intuitively, the goal is to *minimize the number of query executions that require processing at the owner*, producing immediate benefits in terms of reduced use of the more expensive, and less powerful, computational services available at the owner. The weight $w_q(\mathcal{F})$ of a fragmentation \mathcal{F} is then the number of times that the owner needs to be involved in evaluating queries, that is, the sum of the frequencies of queries whose set of attributes in the WHERE clause contains at least an attribute in F_o . Formally, $w_q(\mathcal{F})=\sum_{q \in \mathcal{Q}} freq(q)$ s.t. $Attr(q) \cap F_o \neq \emptyset$. For instance, given fragmentation $\mathcal{F}=\langle\{\text{SSN,Name,DoB,Job,Treatment}\}, \{\text{ZIP,Illness,HDate}\}\rangle$ illustrated in Figure 3 and the query workload in Figure 5(b), $w_q(\mathcal{F}) = freq(q_1) + freq(q_2) + freq(q_3) + freq(q_4) + freq(q_6) + freq(q_7) + freq(q_8) = 33$.
- *Min-Cond*. In addition to the relation schema and the confidentiality constraints, a complete profile of the expected query workload is known. The complete profile reports the specific conditions appearing in each query and not only the set of attributes appearing in the WHERE clause of the query. We assume SELECT-FROM-WHERE SQL queries of the form $q = \text{“SELECT } A \text{ FROM } R \text{ WHERE } C\text{”}$, where A is a subset of the attributes in R , and $C = \bigwedge_i co_i$ is a conjunction of basic conditions co_i of the form $(a_i \text{ op } v)$, $(a_i \text{ op } a_j)$, or $(a_i \text{ IN } \{v_1, \dots, v_k\})$, with a_i and a_j attributes in R , $\{v, v_1, \dots, v_k\}$ constant values in the domain of a_i , and op a comparison operator in $\{=, >, <, \geq, \leq, \neq\}$. The query workload profile is then a set of triples $\mathcal{Q}=\{(q_1, freq(q_1), Co(q_1)), \dots, (q_l, freq(q_l), Co(q_l))\}$, where for each $(q_i, freq(q_i), Co(q_i))$, $i = 1, \dots, l$, q_i is the query to be executed, $freq(q_i)$ is the expected execution frequency of q_i , and $Co(q_i)$ is the set of basic conditions appearing in the WHERE clause of query q_i . In the following, for simplicity, we use $Attr(co)$ to denote the attributes on which a condition co operates. The first, second, and fourth columns of Figure 5(b) illustrate a possible workload profile for relation PATIENT in Figure 1(a). The precise characterization of the workload allows the definition of a metric to *minimize the number of conditions that require processing at the owner*. The weight $w_c(\mathcal{F})$ of a fragmentation \mathcal{F} is the number of times that the owner needs to be involved in evaluating conditions in the query execution. Intuitively,

this corresponds to the number of times the execution of queries requires the evaluation of a condition involving an attribute in F_o . Note that conditions are considered separately, hence the evaluation of n different conditions co_i, \dots, co_n of a query q that involve some attributes in F_o will contribute to the weight for $n \cdot freq(q)$ (whereas in the *Min-Query* scenario the weight would be $freq(q)$, since *Min-Query* evaluates only whether the owner participates in a query evaluation regardless of the number of conditions it has to evaluate). Formally, $w_c(\mathcal{F}) = \sum_{q \in \mathcal{Q}} \sum_{co \in Co(q)} freq(q) \text{ s.t. } Attr(co) \cap F_o \neq \emptyset$. For instance, given fragmentation $\mathcal{F} = \langle \{SSN, Name, DoB, Job, Treatment\}, \{ZIP, Illness, HDate\} \rangle$ illustrated in Figure 3 and the query workload in Figure 5(b), $w_c(\mathcal{F}) = 2 \cdot freq(q_1) + 2 \cdot freq(q_2) + freq(q_3) + freq(q_4) + 2 \cdot freq(q_6) + freq(q_7) + freq(q_8) = 60$. Note that the frequencies of queries q_1 , q_2 , and q_6 are considered twice in the computation of $w_c(\mathcal{F})$, since these queries include two conditions operating on attributes in F_o .

Note that the minimization of the set of conditions executed at the owner side has a direct relationship with the minimization of the traffic needed for receiving results to the queries outsourced to the external server. As a matter of fact, minimizing the conditions executed by the owner is equivalent to maximizing the conditions outsourced to the external server, and therefore delegating to it as much computation as possible. Since the result of evaluating a condition on a relation is a smaller relation, the greater the number of conditions outsourced to the external servers, the smaller will be the corresponding results to be received in response.¹

It is easy to see that the four metrics described above are monotonic with respect to the set containment relationship on F_o . As a matter of fact, given two fragmentations $\mathcal{F} = \langle F_o, F_s \rangle$ and $\mathcal{F}' = \langle F'_o, F'_s \rangle$, where $F_o \subseteq F'_o$, the weight of \mathcal{F}' is necessarily greater than or equal to the weight of \mathcal{F} , since the weight of a fragmentation \mathcal{F} is additive in the number of attributes, in the size of attributes, in the frequency of queries, or in the frequency of conditions that include at least an attribute in F_o , which are all positive values.

The different metrics above translate into different instances of Problem 4.3, by substituting w with the corresponding weight function. In synthesis, the resulting instances of the problem aim at minimizing, respectively: the number of attributes in F_o (*Min-Attr* problem); the physical size of fragment F_o (*Min-Size* problem); the number of times queries requiring access to F_o need to be evaluated (*Min-Query* problem); the number of times conditions on F_o need to be evaluated (*Min-Cond* problem). Figure 4 summarizes the different instantiations of Problem 4.3, corresponding to the metrics previously discussed.

¹To minimize the traffic caused by a given fragmentation, it is necessary to define a metric and a relative weight function representing a query cost model that should take into consideration also the selectivity of the queries, such as the metric introduced in [12].

6 Problem analysis

We now analyze the complexity of the MF problem and of its instances, and then we characterize the solution space of the MF problem.

6.1 Complexity of the problems

The general MF problem as well as its instances *Min-Attr*, *Min-Size*, *Min-Query*, and *Min-Cond* are NP-hard, as formally stated by the following theorem.

Theorem 6.1 *The MF problem and its instances Min-Attr, Min-Size, Min-Query, and Min-Cond are NP-hard.*

PROOF: We now give the proof of the NP-hardness of the *Min-Attr*, *Min-Size*, *Min-Query*, and *Min-Cond* instances.

Min-Attr. This problem directly corresponds to the classical NP-hard *Minimum Hitting Set* (MHS) problem [20], which can be formulated as follows: *Given a finite set A and a collection C of subsets of A , find a subset H (hitting set) of A such that H contains at least one element from each subset in C and $|H|$ is minimum.* The *Min-Attr* problem can be transformed into the MHS problem by taking R as the finite set A and \mathcal{C} as the collection C . A hitting set H corresponds to the set of attributes in F_o . This correspondence follows from the observation that any solution of the *Min-Attr* problem must insert in F_o at least one attribute for each constraint in \mathcal{C} , to guarantee fragmentation correctness (Definition 4.1). The *Min-Attr* problem is therefore NP-hard.

Min-Size. The instance of this problem, where attribute size is 1 for each attribute $a \in R$ (i.e., $size(a)=1$), corresponds to the *Min-Attr* problem, which can be transformed into the NP-hard MHS problem. Since this specific instance of the *Min-Size* problem is NP-hard, also the general *Min-Size* problem is NP-hard.

Min-Query. The instance of this problem, where the workload consists of one query with frequency 1 for each attribute $a \in R$ (i.e., a query q with $Attr(q)=\{a\}$), corresponds to the *Min-Attr* problem, which can be transformed into the NP-hard MHS problem. Since this specific instance of the *Min-Query* problem is NP-hard, also the general *Min-Query* problem is NP-hard.

Min-Cond. The instance of the *Min-Cond* problem where all conditions operate on one attribute only (i.e., conditions are of the form “ $a_x \text{ op } v$ ”, with v a constant value), can be formulated as an instance of the *Min-Size* problem, where the size of each attribute is the number of times that conditions on it need to be evaluated. Such a specific instance of the *Min-Cond* problem is therefore NP-hard. Consequently, the general *Min-Cond* problem is NP-hard. \square

6.2 Solution space

The solution space \mathfrak{F} of the MF problem is represented by all the fragmentations $\mathcal{F}=\langle F_o, F_s \rangle$ of R such that $F_o \cup F_s = R$ and $F_o \cap F_s = \emptyset$ (i.e., complete and non-redundant fragmentations). Since our definition of minimal fragmentation depends on a weight function w that is monotonic with respect to the set containment relationship on F_o , we can define the following partial order relationship on fragmentations, based on the set containment of F_o .

Definition 6.2 (Dominance) *Let $R(a_1, \dots, a_n)$ be a relation schema. A fragmentation $\mathcal{F}=\langle F_o, F_s \rangle$ for R dominates a fragmentation $\mathcal{F}'=\langle F'_o, F'_s \rangle$ of R , denoted $\mathcal{F} \succeq \mathcal{F}'$, iff $F_o \supseteq F'_o$. \mathcal{F} strictly dominates \mathcal{F}' , denoted $\mathcal{F} \succ \mathcal{F}'$, iff $\mathcal{F} \succeq \mathcal{F}'$ and $\mathcal{F} \neq \mathcal{F}'$.*

Since F_o and F_s cannot have common attributes, a fragmentation $\mathcal{F}=\langle F_o, F_s \rangle$ directly dominates a fragmentation $\mathcal{F}'=\langle F'_o, F'_s \rangle$ if \mathcal{F} can be obtained from \mathcal{F}' by moving an attribute from F'_s to F'_o . For instance, considering relation PATIENT in Figure 1(a), fragmentation $\mathcal{F}=\langle \{\text{SSN, Name, DoB, Job, Treatment}\}, \{\text{ZIP, Illness, HDate}\} \rangle$ directly dominates fragmentation $\mathcal{F}'=\langle \{\text{SSN, Name, DoB, Job}\}, \{\text{Treatment, ZIP, Illness, HDate}\} \rangle$ (i.e., $\mathcal{F} \succ \mathcal{F}'$), since \mathcal{F} can be obtained from \mathcal{F}' by moving **Treatment** from F'_s to F'_o .

The set \mathfrak{F} of all possible complete (i.e., $F_o \cup F_s = R$) and non-redundant (i.e., $F_o \cap F_s = \emptyset$) fragmentations along with the dominance relationship form a lattice.

Definition 6.3 (Fragmentation lattice) *Let $R(a_1, \dots, a_n)$ be a relation schema. The fragmentation lattice for R is a pair (\mathfrak{F}, \succeq) , where \mathfrak{F} is the set of all complete and non-redundant fragmentations for R and \succeq is the dominance relationship defined in Definition 6.2.*

The top element of the lattice represents a fragmentation where all the attributes are stored at the data owner, that is, $F_o=R$ and $F_s=\emptyset$. The bottom element of the lattice represents a fragmentation where all the attributes are outsourced to the server, that is $F_o=\emptyset$ and $F_s=R$. Figure 6 illustrates an example of fragmentation lattice defined for a relational schema obtained projecting relation PATIENT in Figure 1(a) over the set $\{\text{DoB, ZIP, Job, Illness}\}$ of attributes. In the figure, attributes are represented with their initials (i.e., $D, Z, J,$ and I) and fragments F_o and F_s are separated by a comma. It is interesting to note that, if a fragmentation $\mathcal{F}=\langle F_o, F_s \rangle$ violates some constraints, such constraints are also violated by any fragmentation $\mathcal{F}'=\langle F'_o, F'_s \rangle$ such that $\mathcal{F} \succeq \mathcal{F}'$. In fact, if a constraint c is violated by \mathcal{F} , then $c \subseteq F_s$. If $\mathcal{F} \succeq \mathcal{F}'$, then $F_o \supseteq F'_o$ and $F_s \subseteq F'_s$ (since all fragmentations in \mathfrak{F} are complete). Hence, $c \subseteq F'_s$, and \mathcal{F}' violates c . For instance, consider again the relational schema obtained projecting relation PATIENT in Figure 1(a) over the set $\{\text{DoB, ZIP, Job, Illness}\}$ of attributes. In this case, the confidentiality constraints that need to be enforced are constraints c_2 and c_4 in Figure 1(b). Fragmentation $\mathcal{F}=\langle \{\text{DoB, ZIP}\}, \{\text{Job, Illness}\} \rangle$ is not correct since constraint c_4 is violated. Analogously, all the

fragmentations dominated by \mathcal{F} , that is, $\mathcal{F}'_1 = \langle \{\text{DoB}\}, \{\text{ZIP, Job, Illness}\} \rangle$, $\mathcal{F}'_2 = \langle \{\text{ZIP}\}, \{\text{DoB, Job, Illness}\} \rangle$, and $\mathcal{F}'_3 = \langle \{\}, \{\text{DoB, ZIP, Job, Illness}\} \rangle$, violate (at least) c_4 . In the fragmentation lattice in Figure 6, correct fragmentations are framed by solid boxes, while fragmentations that violate at least one constraint are framed by dotted boxes.

Since the number of fragmentations populating the solution space is exponential in the number of attributes composing R , and the minimal fragmentation problem is NP-hard (see Theorem 6.1), we propose a definition of *local minimality*, which can be used to find a correct fragmentation via an efficient heuristics (see Section 7.2). Before defining a locally minimal fragmentation, it is interesting to note that the weight of the fragmentations decreases along each path from the top to the bottom element of the fragmentation lattice (i.e., given a fragmentation \mathcal{F} and a fragmentation \mathcal{F}' : $\mathcal{F} \succ \mathcal{F}' \Rightarrow w(\mathcal{F}) \geq w(\mathcal{F}')$). This property, proved by the following proposition, is a direct consequence of the fact that we assume the weight function to be monotonic with respect to the set containment of F_o .

Proposition 6.4 *Let $R(a_1, \dots, a_n)$ be a relation schema, \mathcal{C}_f be a well defined set of confidentiality constraints over R , $w: \mathcal{P}(R) \times \mathcal{P}(R) \rightarrow \mathbb{R}^+$ be a monotonic weight function, and \mathcal{F} and \mathcal{F}' be two fragmentations for R such that $\mathcal{F} \succ \mathcal{F}'$. We have that $w(\mathcal{F}) \geq w(\mathcal{F}')$.*

PROOF: Let $\mathcal{F} = \langle F_o, F_s \rangle$ and $\mathcal{F}' = \langle F'_o, F'_s \rangle$ be two complete and non-redundant fragmentations such that $\mathcal{F} \succ \mathcal{F}'$. By Definition 6.2 we have that $F_o \supset F'_o$. Since the weight function w is monotonic with respect to the set containment relationship on F_o , we have that $w(\mathcal{F}) \geq w(\mathcal{F}')$. \square

Our definition of local minimality is based on the monotonicity property of the weight function over the fragmentation lattice, as defined in the following.

Definition 6.5 (Locally minimal fragmentation) *Let $R(a_1, \dots, a_n)$ be a relation schema, \mathcal{C}_f be a well defined set of confidentiality constraints, $w: \mathcal{P}(R) \times \mathcal{P}(R) \rightarrow \mathbb{R}^+$ be a monotonic weight function, and $\mathcal{F} = \langle F_o, F_s \rangle$ be a fragmentation for R . \mathcal{F} is a locally minimal fragmentation for R , with respect to \mathcal{C}_f and w iff: 1) \mathcal{F} is correct (Definition 4.1) and 2) $\nexists \mathcal{F}'$ such that $\mathcal{F} \succ \mathcal{F}'$ and \mathcal{F}' is correct.*

According to this definition, a solution is *locally minimal* if all the solutions that it dominates are not correct, that is, it is not possible to obtain a correct fragmentation moving some attribute from F_o to F_s . For instance, consider the fragmentation lattice in Figure 6. The correct fragmentations $\langle DJ, ZI \rangle$, $\langle ZJ, DI \rangle$, and $\langle I, DZJ \rangle$ are locally minimal (i.e., all the fragmentations dominated by them are not correct), since moving any attribute from F_o to F_s would violate at least a constraint. The problem that now we need to solve consists in finding a locally minimal fragmentation.

Problem 6.6 (LMF problem) Given a relation schema $R(a_1, \dots, a_n)$, a well defined set \mathcal{C}_f of confidentiality constraints over R , and a monotonic weight function $w: \mathcal{P}(R) \times \mathcal{P}(R) \rightarrow \mathbb{R}^+$, determine a locally minimal fragmentation $\mathcal{F} = \langle F_o, F_s \rangle$ for R , with respect to \mathcal{C}_f and w (Definition 6.5).

In the following, we first model the LMF problem as a hypergraph coloring problem that captures, as special cases, all the weight functions defined in Section 5. We then illustrate a heuristic algorithm for its solution.

7 Graph modeling of the problem and heuristic algorithm

In this section, we present a modeling of the MF and of the LMF problems as a hypergraph 2-coloring problem. We also introduce a heuristic algorithm that uses such a modeling.

7.1 Hypergraph coloring problem

Before illustrating how to compute a locally minimal fragmentation, we analyze the weight functions discussed in Section 5. We first observe that we can model the data and query workload as a set \mathcal{T} of weighted subsets of R . We will refer to each subset of the attributes in the original relation R , which is characterized by a weight, as a *weighted target* $t \in \mathcal{T}$. Each metric evaluates the intersection between F_o and the weighted targets in \mathcal{T} describing the considered workload: the basic idea is that the workload of the owner raises with the increase of the number of weighted targets that have a non empty intersection with F_o . For each weight function introduced in Section 5, the corresponding set of weighted targets is defined as follows.

- w_a : each attribute $a \in R$ corresponds to a target $t = \{a\}$ whose weight is 1.
- w_s : each attribute $a \in R$ corresponds to a target $t = \{a\}$ whose weight is the size of the attribute.
- w_q : each set $Attr(q_i)$ of attributes characterizing a query q_i corresponds to a target $t = Attr(q_i)$ whose weight is sum of the frequencies of all queries q_i characterized by the same set $Attr(q_i)$.
- w_c : each set $Attr(co)$ of attributes representing the attributes on which a condition co operates corresponds to a target $t = Attr(co)$ whose weight is the frequency of condition co (i.e., the sum of the frequencies of the queries including a condition operating on $Attr(co)$ in the WHERE clause).

In the following, function $w_{\mathcal{T}} : \mathcal{T} \rightarrow \mathbb{R}^+$ determines the weight of each target t in \mathcal{T} . The weight of a fragmentation \mathcal{F} is then computed as the sum of the weights of the targets hit by F_o , that is, having a non empty intersection with F_o : $\sum_{t \in \mathcal{T}} w_{\mathcal{T}}(t)$ s.t. $t \cap F_o \neq \emptyset$. As an example, consider fragmentation $\mathcal{F} = \langle \{\text{SSN, Name, DoB, Job, Treatment}\}, \{\text{ZIP, Illness, HDate}\} \rangle$ in Figure 3 and the workload profile in Figure 5(b).

The set of weighted targets is $\mathcal{T}=\{Attr(q_1), \dots, Attr(q_9)\}=\{\{\text{Name, DoB}\}, \{\text{Name, Job}\}, \{\text{Name, Illness}\}, \{\text{DoB, Illness}\}, \{\text{ZIP, Illness}\}, \{\text{Job, Illness}\}, \{\text{DoB, Illness, Treatment}\}, \{\text{DoB, HDate, Treatment}\}, \{\text{HDate}\}\}$. We have that $w_q(\mathcal{F})=w_{\mathcal{T}}(Attr(q_1)) + w_{\mathcal{T}}(Attr(q_2)) + w_{\mathcal{T}}(Attr(q_3)) + w_{\mathcal{T}}(Attr(q_4)) + w_{\mathcal{T}}(Attr(q_6)) + w_{\mathcal{T}}(Attr(q_7)) + w_{\mathcal{T}}(Attr(q_8)) = 4 + 14 + 1 + 1 + 4 + 3 + 6 = 33$. Figure 7 summarizes the definition of the set \mathcal{T} of targets and $w_{\mathcal{T}}$ for the different weight functions considered.

The inputs to the LMF problem (as well as to the MF problem) can now be modeled as a hypergraph as follows.

Definition 7.1 (CT-hypergraph) *Given a relation schema $R(a_1, \dots, a_n)$, a well defined set $\mathcal{C}_f=\{c_1, \dots, c_m\}$ of confidentiality constraints over it, and a monotonic weight function $w:\mathcal{P}(R)\times\mathcal{P}(R)\rightarrow\mathbb{R}^+$ characterized by a set \mathcal{T} of weighted targets with weight function $w_{\mathcal{T}}:\mathcal{T}\rightarrow\mathbb{R}^+$, we define a corresponding undirected CT-hypergraph $H(\mathcal{A}, \mathcal{C}\cup\mathcal{T}, w_{\mathcal{T}})$, where:*

- $\mathcal{A} = R$ is the set of vertices of the hypergraph;
- $\mathcal{C} = \mathcal{C}_f$ is the set of non-weighted hyperedges (\mathcal{C} -edges in the following) of the hypergraph;
- \mathcal{T} is the set of weighted hyperedges (\mathcal{T} -edges in the following) and corresponds to the set \mathcal{T} of targets associated with w ;
- $w_{\mathcal{T}}$ is the weight function defined over \mathcal{T} .

As an example, consider relation PATIENT, its confidentiality constraints in Figure 1, and the data and workload profiles in Figure 5. Figure 8 illustrates the CT-hypergraphs modeling the instances of the MF problem, considering the different metrics and corresponding weight functions (i.e., Min-Attr, Min-Size, Min-Query, and Min-Cond). In the figure, each vertex contains the initial of the attribute that it represents, \mathcal{T} -edges are reported at the top of the figure along with their weights, and \mathcal{C} -edges are reported at the bottom of the figure in boldface. Given a CT-hypergraph $H(\mathcal{A}, \mathcal{C}\cup\mathcal{T}, w_{\mathcal{T}})$, we define a 2-coloring function $color:\mathcal{A}\rightarrow\{Own, Srv\}$ as a function that produces a 2-coloring of the hypergraph: all vertices $v\in\mathcal{A}$ such that $color(v)=Own$ are called *Own-vertices*, and all vertices $v\in\mathcal{A}$ such that $color(v)=Srv$ are called *Srv-vertices*. The weight of a colored CT-hypergraph can then be defined as the sum of the weights of the \mathcal{T} -edges incident to at least an *Own-vertex* (i.e., $weight(H, color)=\sum_{t\in\mathcal{T}} w_{\mathcal{T}}(t)$ s.t. $\exists v\in t, color(v)=Own$).

A correct fragmentation $\mathcal{F}=\langle F_o, F_s \rangle$ for R can now be interpreted as a coloring function over the CT-hypergraph such that the *Own-vertices* are those representing attributes in F_o and the *Srv-vertices* are those representing attributes in F_s . Analogously, a coloring function $color$ can be interpreted as a correct fragmentation if no \mathcal{C} -edge in the hypergraph is incident to only *Srv-vertices*, as formally stated in the following.

Definition 7.2 (Correct coloring) Let $H(\mathcal{A}, \mathcal{C} \cup \mathcal{T}, w_{\mathcal{T}})$ be a CT-hypergraph and $color : \mathcal{A} \rightarrow \{Own, Srv\}$ be a corresponding 2-coloring function. Function $color$ is a correct coloring for H with respect to \mathcal{C} , iff $\forall e \in \mathcal{C}: \exists v \in e$ such that $color(v)=Own$.

It is easy to see that a correct function $color$ induces a correct fragmentation \mathcal{F} such that F_o contains all attributes corresponding to *Own*-vertices and F_s contains all attributes corresponding to *Srv*-vertices. The resulting fragmentation \mathcal{F} satisfies all conditions in Definition 4.1. In particular,

1. *completeness* is guaranteed by the fact that the domain of function $color$ is \mathcal{A} (i.e., every attribute belongs either to F_o or to F_s);
2. *confidentiality* is guaranteed by imposing that $\forall e \in \mathcal{C}: \exists v \in e$ such that $color(v)=Own$ (i.e., for each constraint, at least one of the involved attributes is in F_o);
3. *non-redundancy* is guaranteed by the fact that function $color$ assigns no more than one color to each vertex in \mathcal{A} (i.e., no attribute belongs to both F_o and F_s).

As an example, Figure 9 illustrates a colored CT-hypergraph for the *Min-Att* problem, where *Own*-vertices are gray and *Srv*-vertices are white. The coloring of this CT-hypergraph is correct since each \mathcal{C} -edge is incident to at least an *Own*-vertex and therefore induces a correct fragmentation $\mathcal{F}=\langle\{\{SSN,Name,DoB,Illness\},\{ZIP,Job,Treatment,HDate\}\}\rangle$.

Given the correspondence between the hypergraph coloring and a correct fragmentation, the definition of minimal fragmentation (Definition 4.2) can be reformulated as a *minimal coloring* of the CT-hypergraph as follows.

Definition 7.3 (Minimal coloring) Let $H(\mathcal{A}, \mathcal{C} \cup \mathcal{T}, w_{\mathcal{T}})$ be a CT-hypergraph and $color : \mathcal{A} \rightarrow \{Own, Srv\}$ be a 2-coloring function defined on the vertices of H . Function $color$ is a minimal coloring for H , with respect to \mathcal{C} and $w_{\mathcal{T}}$, iff: 1) $color$ is correct (Definition 7.2) and 2) $\nexists color' : \mathcal{A} \rightarrow \{Own, Srv\}$ such that $weight(H, color') < weight(H, color)$ and $color'$ is correct.

The MF problem directly corresponds to the problem of computing a minimal coloring for H . Analogously, the definition of a locally minimal fragmentation can be reformulated in terms of the $color$ function as follows. Given two fragmentations \mathcal{F} and \mathcal{F}' and the corresponding coloring functions $color$ and $color'$, respectively, if $\mathcal{F} \succ \mathcal{F}'$, then the set of *Own*-vertices induced by $color$ is a superset of the set of *Own*-vertices induced by $color'$. Fragmentation \mathcal{F} with coloring function $color$ is then locally minimal if it is correct and there does not exist another correct fragmentation \mathcal{F}' with coloring function $color'$ such that the set of *Own*-vertices resulting from function $color'$ is a subset of the set of *Own*-vertices resulting from function $color$. The definition of a locally minimal fragmentation can then be reformulated through the following definition of locally minimal coloring.

Definition 7.4 (Locally minimal coloring) Let $H(\mathcal{A}, \mathcal{C} \cup \mathcal{T}, w_{\mathcal{T}})$ be a CT-hypergraph and $\text{color} : \mathcal{A} \rightarrow \{\text{Own}, \text{Srv}\}$ be a 2-coloring function defined on the vertices of H . Function color is a locally minimal coloring for H , with respect to \mathcal{C} and $w_{\mathcal{T}}$, iff: 1) color is correct (Definition 7.2) and 2) $\nexists \text{color}' : \mathcal{A} \rightarrow \{\text{Own}, \text{Srv}\}$ such that $\{a \in \mathcal{A} \mid \text{color}'(a) = \text{Own}\} \subset \{a \in \mathcal{A} \mid \text{color}(a) = \text{Own}\}$ and color' is correct.

The LMF problem directly corresponds to the problem of computing a locally minimal coloring for H . The heuristic algorithm illustrated in Section 7.2 uses this correspondence to compute a fragmentation \mathcal{F} that is locally minimal with respect to \mathcal{C} and w , where w is a monotonic weight function associating a weight with each target t in \mathcal{T} .

7.2 Heuristic algorithm

Our heuristic algorithm, illustrated in Figure 10, takes as input the CT-hypergraph $H = (\mathcal{A}, \mathcal{C} \cup \mathcal{T}, w_{\mathcal{T}})$ modeling an instance of the locally minimal fragmentation problem and returns the set Own of *Own*-vertices in the hypergraph, which corresponds to the set of attributes composing F_o .

Given a vertex v , let $w(v)$ be the sum of the weights of its incident \mathcal{T} -edges, and let $c(v)$ be the number of \mathcal{C} -edges incident to v (which corresponds to the number of constraints that include the attribute represented by v). The ratio $w(v)/c(v)$ reflects the relative cost of including vertex v into Own , obtained as the weight to pay divided by number of constraints that would be solved by including the attribute represented by v into F_o . At initialization, the set of constraints and weighted targets are those given in input to the problem, and the set Own of *Own*-vertices is empty.

We first note that any correct fragmentation must include in F_o all the attributes appearing in singleton constraints. Therefore, vertices representing attributes appearing in singleton constraints must be inserted into Own . Given this observation, the first **for each** loop of the algorithm identifies singleton \mathcal{C} -edges (i.e., \mathcal{C} -edges incident to one vertex only) and removes from the hypergraph the vertices to which these \mathcal{C} -edges are incident. The removed vertices are inserted into Own . Moreover, the algorithm removes all the \mathcal{C} -edges and \mathcal{T} -edges incident to a vertex that has been removed (i.e., constraints and targets including attributes appearing in singleton constraints). In particular, the \mathcal{T} -edges are removed since these targets are incident to at least an *Own*-vertex and therefore there is no further weight to consider for them.

The algorithm then performs a **while** loop that, at each iteration, chooses a vertex v among the vertices with the lowest $w(v)/c(v)$ ratio (i.e., a vertex that maximizes the relative cost for constraint satisfaction), and inserts it into Own . Vertex v and all its incident hyperedges are then removed from the hypergraph. The inclusion of v into Own brings to satisfaction all the constraints in which the attribute represented by v is involved (which therefore need not to be considered anymore). Therefore, the number of constraints $c(v')$ in which the attribute

represented by a vertex v' that appears with v in at least one \mathcal{C} -edge must be updated. Also, the weight of all the targets in which the attribute represented by v is involved is added to the weight of the solution. The total weight $w(v')$ of \mathcal{T} -edges incident to a vertex v' that appears with v in at least one \mathcal{T} -edge must then be updated. The **while** loop terminates if either the hypergraph is empty (i.e., all vertices are in Own) or all vertices v in \mathcal{A} have $c(v)=0$ (i.e., all constraints have been solved).

The set Own obtained at the end of the **while** loop may not represent a locally minimal fragmentation since the inclusion of a vertex with higher ratio might have made unnecessary the inclusion of a vertex, with lower ratio, previously inserted into Own . Hence, the algorithm iteratively considers vertices in Own in reverse order of insertion and, for each considered vertex v , it determines if each \mathcal{C} -edge in the original hypergraph H , is still incident to at least a vertex in $Own \setminus \{v\}$ (i.e., the fragmentation implied by Own still satisfies all the constraints of the problem). If it does, v is removed from Own . Note that it is sufficient to check each vertex once (i.e., only one scan of Own needs to be performed). Indeed, if a vertex v belonging to Own is necessary for the satisfaction of all the confidentiality constraints, the removal of another vertex v' from Own cannot make v unnecessary (since v would have been unnecessary before the removal of v' from Own).

The resulting fragmentation $\mathcal{F} = \langle F_o, F_s \rangle$ is then obtained by inserting all the attributes represented by vertices in Own into F_o , and by setting $F_s = R \setminus F_o$.

Note that the heuristic algorithm proposed basically visits the fragmentation lattice (\mathfrak{F}, \succeq) , starting from the bottom element in (\mathfrak{F}, \succeq) and following a path until it reaches a correct fragmentation \mathcal{F} . Since \mathcal{F} might not be locally minimal, the algorithm then follows a path back to the bottom element, visiting only correct fragmentations. It terminates at the first correct fragmentation \mathcal{F}' that only dominates non correct fragmentations, and returns \mathcal{F}' , which is, by definition, locally minimal. The experimental evaluation in Section 9 shows that the fragmentations computed by our heuristics well approximate optimal solutions.

Example 7.5 *Figure 11 illustrates the execution, step by step, of the heuristic algorithm to solve the Min-Query problem on relation PATIENT and the set of confidentiality constraints in Figure 1, and the query workload in Figure 5(b). For the sake of readability, each vertex v has $w(v)$ and $c(v)$ indicated at the right-top and right-bottom of the vertex, respectively.*

*The algorithm removes from the hypergraph and inserts into Own vertices S and N , which represent attributes **SSN** and **Name**, respectively, since they are involved in singleton constraints c_0 and c_1 . It then removes the edges incident to S and N , that is, the \mathcal{C} -edges representing constraints c_0 and c_1 and the \mathcal{T} -edges representing queries q_1 , q_2 , and q_3 . The vertices and edges removed from the hypergraph are represented with a dotted line, while vertices already inserted into Own (and therefore removed from the hypergraph) are gray and the weight $w(v)$, with which they contribute to the weight of the solution, is represented at the right top of the vertex. The weights*

$w(D)$, $w(J)$, and $w(I)$ of the vertices representing attributes involved in a query together with **SSN** or with **Name** are updated according to the set of hyperedges removed from the hypergraph.

At each iteration of the **while** loop, a vertex with lowest $w(v)/c(v)$ ratio is inserted into *Own* and the \mathcal{C} -edges and \mathcal{T} -edges incident to it are removed from the hypergraph. Vertex J therefore becomes an *Own*-vertex, and \mathcal{C} -edge $\{J, I\}$ (corresponding to c_4) and \mathcal{T} -edge $\{J, I\}$ (corresponding to q_6) are removed from the hypergraph. The subsequent steps proceed in analogous way, inserting into *Own* vertices T and D . When inserting D into *Own*, the set \mathcal{C} of \mathcal{C} -edges becomes empty, meaning that all constraints are satisfied and the algorithm terminates. The computed solution is minimal, since removing any attribute from fragment F_o , represented by *Own*, would violate at least a constraint. The resulting fragmentation, represented by the 2-colored hypergraph in Figure 12, is: $F_o = \{\text{SSN}, \text{Name}, \text{DoB}, \text{Job}, \text{Treatment}\}$; $F_s = \{\text{ZIP}, \text{Illness}, \text{HDate}\}$, whose weight is $w_q(\mathcal{F}) = 33$ (obtained as the sum of the weights $w(v)$ of vertices S , N , D , J , and T in the last hypergraph in the figure).

The execution of the algorithm produces the following results:

- *Min-Attr*: $F_o = \{\text{SSN}, \text{Name}, \text{DoB}, \text{Illness}\}$, $F_s = \{\text{ZIP}, \text{Job}, \text{Treatment}, \text{HDate}\}$, $w_a(\mathcal{F}) = 4$;
- *Min-Size*: $F_o = \{\text{SSN}, \text{Name}, \text{ZIP}, \text{Illness}\}$, $F_s = \{\text{DoB}, \text{Job}, \text{Treatment}, \text{HDate}\}$, $w_s(\mathcal{F}) = 49$;
- *Min-Cond*: $F_o = \{\text{SSN}, \text{Name}, \text{Illness}, \text{Treatment}\}$, $F_s = \{\text{DoB}, \text{ZIP}, \text{Job}, \text{HDate}\}$, $w_c(\mathcal{F}) = 52$.

We can observe that the dynamic computation of the ratios can be adequately managed with a priority queue containing the current vertices of the hypergraph. The priority of the vertices in the queue is dictated by the value of the ratio $w(v)/c(v)$: vertices with lower ratio have higher priority. Each time the value $w(v)/c(v)$ of a vertex v is dynamically updated (due to the removal of some of its incident hyperedges), the vertex v is removed and inserted again in the queue with the new ratio value. Assuming implementation of the dynamic computation of the ratios through a priority queue, the proposed heuristic algorithm has a polynomial time complexity and correctly computes a locally minimal fragmentation, as proved by the following theorems.

Theorem 7.6 *Given a CT-hypergraph $H = (\mathcal{A}, \mathcal{C} \cup \mathcal{T}, w_{\mathcal{T}})$, the computational complexity of the algorithm in Figure 10 is $O(n + (\sum_{t \in \mathcal{T}} |t| + \sum_{c \in \mathcal{C}} |c|) \log(n))$ in time, where $n = |\mathcal{A}|$.*

PROOF: The construction of the priority queue costs $O(n)$. The first **for each** loop and the **while** loop cost $O(n + (\sum_{t \in \mathcal{T}} |t| + \sum_{c \in \mathcal{C}} |c|) \log(n))$, since: *i*) each attribute is visited at most once, *ii*) each hyperedge is considered only once for each attribute to which it is incident, and *iii*), in the worst case, each vertex is removed and inserted in the queue every time its incident hyperedges are updated. The deletion and the insertion in a priority queue can be performed in $\log(n)$ time. Also the extraction of the element with highest priority can be performed in $\log(n)$ time. The last **for each** loop verifies whether removing a vertex from *Own*, all

the hyperedges in \mathcal{C} include at least a vertex in Own . This control is performed for each vertex in Own , and again each hyperedge is considered at most once for each vertex to which it is incident. The cost of this step is therefore $O(n + \sum_{c \in \mathcal{C}} |c|)$. \square

Theorem 7.7 (Correctness) *Given a relation schema $R(a_1, \dots, a_n)$, a well defined set $\mathcal{C} = \{c_1, \dots, c_m\}$ of confidentiality constraints over R , a set $\mathcal{T} = \{t_1, \dots, t_l\}$ of targets over R , a weight function $w_{\mathcal{T}} : \mathcal{T} \rightarrow \mathbb{R}^+$, and the solution Own computed by the algorithm in Figure 10; the algorithm in Figure 10 terminates and the fragmentation $\mathcal{F} = \langle F_o, F_s \rangle$, defined as $F_o = Own$ and $F_s = R \setminus F_o$, is locally minimal (Definition 6.5).*

PROOF: Let Own be the solution computed by the algorithm in Figure 10 and $\mathcal{F} = \langle F_o, F_s \rangle$ be the corresponding fragmentation. To prove that \mathcal{F} represents a locally minimal fragmentation, we need to prove that both the conditions in Definition 6.5 are satisfied.

1. To prove that \mathcal{F} is correct we show that: *i)* $F_o \cup F_s = R$, *ii)* $\forall c \in \mathcal{C}, c \not\subseteq F_s$, and *iii)* $F_o \cap F_s = \emptyset$. Satisfaction of conditions *i)* and *iii)* immediately follows from the fact that F_s is computed as $R \setminus F_o$. Let us then prove that condition *ii)* is satisfied. Suppose, by contradiction, that there exists a constraint $c \in \mathcal{C}$ violated by F_s , that is, such that $c \subseteq F_s$. This is in contradiction with the fact that each attribute appearing in a singleton constraint is in F_o and the **while** loop ends only when the hypergraph is empty or, for all vertices v , $c(v) = 0$, meaning that all \mathcal{C} -edges have been removed from the hypergraph. Moreover, the last **for each** loop removes vertices from Own only if all the edges in \mathcal{C}' (representing the original set of constraints) are incident to at least a vertex in Own (i.e., if the constraints are not violated).
2. The fact that there exists a correct fragmentation \mathcal{F}' such that $\mathcal{F} \succ \mathcal{F}'$ is guaranteed by the last **for each** loop. In fact, if $\mathcal{F} \succ \mathcal{F}'$, then $F_o \supset F'_o$. The last loop verifies whether removing a vertex from Own , each \mathcal{C} -edge is incident to at least a vertex in Own (i.e., \mathcal{F} satisfies all the confidentiality constraints in \mathcal{C}). The loop checks each vertex v in Own and, if its removal from Own does not violate any constraint, removes v from Own . Since the number of vertices in Own that are incident to \mathcal{C} -edges can only decrease, it is sufficient to perform a single loop to guarantee the property.

The algorithm terminates since all the **for each** loops iterate on finite sets, that is, the set \mathcal{A} of vertices, the set \mathcal{C} of \mathcal{C} -edges, the set \mathcal{T} of \mathcal{T} -edges, and the set Own of Own -vertices (which is a subset of \mathcal{A}). Also, at each iteration of the **while** loop, a vertex is removed from the hypergraph. In the worst case, the **while** loop is iterated n times. \square

8 Query execution

Users formulate queries referring to the original relation schema R , without worrying about whether or not R has been fragmented. However, at the physical level, R is stored by means of the two fragments F_o and F_s resulting from the fragmentation process described in the previous sections. Therefore, queries submitted by authorized users and referring to R need to be translated into equivalent queries operating on F_o and/or F_s .

We consider SELECT-FROM-WHERE SQL queries of the same form as the queries populating the workload profiles (see Section 5), that is, $q = \text{“SELECT } A \text{ FROM } R \text{ WHERE } C\text{”}$, where A is a subset of the attributes in R , and $C = \bigwedge_i co_i$ is a conjunction of basic conditions of the form $(a_i \text{ op } v)$, $(a_i \text{ op } a_j)$, or $(a_i \text{ IN } \{v_1, \dots, v_k\})$, with a_i and a_j attributes in R , $\{v, v_1, \dots, v_k\}$ constant values in the domain of a_i , and op a comparison operator in $\{=, >, <, \geq, \leq, \neq\}$. We now describe the query translation process.

8.1 Classification of conditions

Condition $C = \bigwedge_i co_i$ in the WHERE clause of a query can be split into three conditions, namely C_o , C_s , and C_{so} , depending on the attributes involved in the subexpression and that determine the party (owner and/or external server) responsible for its evaluation.

- $C_o = \bigwedge_i co_i : Attr(co_i) \subseteq F_o$ is the conjunction of the conditions in C that can be evaluated only by the *owner*, independently from the server, since they involve only attributes stored at the owner.
- $C_s = \bigwedge_i co_i : Attr(co_i) \subseteq F_s$ is the conjunction of the conditions in C that can be evaluated by the *external server*, independently from the owner, since they involve only attributes stored at the server. Note that, since the attributes stored at the server can be safely communicated to the owner, C_s could also be evaluated by the owner. However, this last option is highly impractical and should be avoided, because it requires to send to the owner the projection over F_s of the attributes $Attr(co_i)$, for each co_i in C_s .
- $C_{so} = \bigwedge_i co_i : Attr(co_i) \cap F_o \neq \emptyset \wedge Attr(co_i) \cap F_s \neq \emptyset$ is the conjunction of conditions in C of the form $(a_i \text{ op } a_j)$, where $a_i \in F_o$ and $a_j \in F_s$ or viceversa. Therefore, these conditions require to evaluate information at the server and at the owner, since they involve both attributes stored at the owner and attributes stored at the server. Note that the conditions in C_{so} involve attributes of F_o that cannot be released to the external server, since they would possibly violate confidentiality constraints. Therefore, C_{so} can be evaluated only by the owner.

Example 8.1 Consider the fragmentation in Figure 3 and the query retrieving the Names and Illness of patients living in area 22030 or 22034, treated with antihistamine, and that were at least eighteen when they have been hospitalized:

```
SELECT Name, Illness
```

```
FROM Patient
```

```
WHERE (ZIP IN {22030,22034}) AND (Treatment="antihistamine") AND (HDate-DoB≥18y)
```

$C_o = \{\text{Treatment} = \text{"antihistamine"}\}$, since attribute `Treatment` belongs to F_o ; $C_s = \{\text{ZIP IN } \{22030, 22034\}\}$, since attribute `ZIP` belongs to F_s ; and, $C_{so} = \{\text{HDate-DoB} \geq 18y\}$, since attributes `DoB` $\in F_o$ and `HDate` $\in F_s$.

8.2 Query evaluation strategies

The evaluation process of a query q on R can follow two different strategies, depending on the order in which conditions C_o , C_s , and C_{so} are evaluated. The *Server-Owner* strategy first evaluates C_s at the server side and then evaluates both C_o and C_{so} at the owner side. The *Owner-Server* strategy first evaluates C_o at the owner side, then evaluates C_s at the server side, and finally checks C_{so} at the owner side again. The left-hand side of Figure 13 illustrates the two algorithms implementing the Server-Owner and Owner-Server strategies, respectively, executed by the owner for translating and evaluating q . In the following, we briefly illustrate the working of the two algorithms.

- *Server-Owner* strategy. The basic idea is that first the server evaluates the conditions in C_s on F_s , and then the owner refines the result by filtering out the tuples that satisfy neither C_o nor C_{so} .

The owner receives from the user the query q (step 1) to be evaluated. For the evaluation of q , she splits the condition C in the WHERE clause of q into three subexpressions, C_o , C_s , and C_{so} , as illustrated in Section 8.1 (step 2), and identifies the set A_{q_s} of attributes that belong to F_s , but that she needs for completing the evaluation of q (i.e., that belong to A or appear in a basic condition in C_{so}) (step 3). The owner then defines and sends to the server the query q_s , operating on F_s and evaluating condition C_s (step 4). The SELECT clause of this query is composed of A_{q_s} and attribute `tid`, which is necessary to join the result of q_s with F_o . The server evaluates q_s and sends its result R_s back to the owner (step 5). The owner defines and executes the query q_{so} operating on the join between R_s and F_o and evaluating both C_o and C_{so} (step 6). The result of the evaluation of query q_{so} corresponds to the result of the original query q and is then returned to the user (step 7).

- *Owner-Server* strategy. The basic idea is that first the owner evaluates the conditions in C_o on F_o , the server then refines the result by filtering out the tuples that do not satisfy C_s , and finally the owner

discards the tuples that do not satisfy C_{so} . The first three steps performed by the owner are the same as for strategy Server-Owner. She then defines and executes query q_o , operating on F_o and evaluating condition C_o (step 4). The SELECT clause of this query is composed of attribute `tid` only, which is the only attribute that can be communicated to the server (step 5) and is needed to perform the join. The owner sends to the server the query q_s , operating on the join between R_o and F_s and evaluating condition C_s , for execution (step 6). The SELECT clause of query q_s (as for Server-Owner strategy) is composed of A_{q_s} and attribute `tid`. The server evaluates q_s and returns its result R_s to the owner (step 7), who defines and executes query q_{so} operating on the join between R_s and F_o and evaluating C_{so} (step 8). The relation resulting from the evaluation of query q_{so} , corresponding to the result of the original query q , is returned to the user (step 9).

Example 8.2 Consider relation `PATIENT` in Figure 1(a), its fragmentation in Figure 3, and the query q introduced in Example 8.1. $A_{q_s} = \{\text{Illness, HDate}\}$, since `Illness` belongs to the SELECT clause of the original query, while `HDate` is involved in a condition in C_{so} . The right-hand side of Figure 13 illustrates the queries generated by the algorithm for translating q into a set of equivalent queries operating on \mathcal{F} , with the Server-Owner and Owner-Server strategies.

The choice between the Server-Owner and Owner-Server strategies depends on the possible leakage of information that the Owner-Server strategy may cause. If the external server is supposed to know or can infer query q , the Owner-Server strategy cannot be adopted as it would cause leakage of information. In fact, the external server would learn that the tuples in R_o are all and only the tuples that satisfy C_o . As an example, consider query $q = \text{“SELECT Illness FROM Patient WHERE DoB}>1985/12/31 \text{ AND ZIP}=22031\text{”}$ over relation `PATIENT`. If we adopt the Owner-Server strategy, $q_o = \text{“SELECT tid FROM }F_o \text{ WHERE DoB}>1985/12/31\text{”}$ returns only two tuples tuple with `tid=3` and `tid=4`. The external server then evaluates query $q_s = \text{“SELECT tid, Illness FROM }F_s \text{ JOIN }R_o \text{ ON }F_s.\text{tid}=R_o.\text{tid WHERE ZIP}=22031\text{”}$, which returns only one tuple, with `tid=4`. Knowing q , the external server can, from the result, reconstruct the sensitive associations among `DoB`, `ZIP`, and `Illness` (violating c_2) for the tuple with `tid=4`. We also note that, also assuming that the server does not know q , it knows that the tuples in R_o are all and only the tuples in R that satisfy a given condition. By observing a long enough series of queries, the server could possibly infer information about the content of R (and reconstruct sensitive associations).²

If the server does not know and cannot infer q , both the Server-Owner and Owner-Server strategies can be adopted without privacy violations. In this case, the choice between the two strategies can only be based

²To avoid information leakage, the owner can add noise to the result of query q_o , by artificially inserting the id of some tuples that do not satisfy C_o . In this case, query q_{so} should evaluate both C_{so} and C_o , to remove from the final result the tuples artificially added to R_o .

on performance. In other words, following the criterion usually adopted by distributed database systems, the most selective condition is evaluated first (i.e., the sub-query with the smallest result is anticipated). Therefore, if C_o is more selective than C_s , the Owner-Server strategy is adopted; if C_s is more selective than C_o , the Server-Owner strategy is adopted.

9 Experimental results

We have implemented our heuristic algorithm to assess its efficiency and effectiveness, in terms of both the quality of the returned solution and the execution time for its computation. For comparing the solution obtained by the heuristics with the optimum, we also implemented an exhaustive algorithm visiting the whole fragmentation lattice and solving the MF problem (Problem 4.3). The experiments have been carried out on a server machine with 2 Intel Xeon Quad 2.0GHz L3-4MB, 12GB RAM, 4 drives Western Digital SATA II 1TB, 7200 RPM, 32 MB cache, and a Linux Ubuntu 9.04 operating system (kernel 2.6.28-11-server).

The relation schema considered in the experiments is composed of 50 attributes. The experiments have considered configurations with an increasing number of attributes, from 5 to 50. For each configuration of n attributes, we randomly generated a well defined set of confidentiality constraints composed of a number of constraints varying from 2 to $0.85 \cdot n$, each including a number of attributes ranging between 1 and $0.5 \cdot n$. Analogously, for each configuration of n attributes, we randomly generated a query workload characterized by a number of queries varying from 2 to n , each operating on a number of attributes ranging between 1 and $0.5 \cdot n$, and with frequency ranging between 1 and 10. The results illustrated in the following are computed as the average of the values obtained with 30 runs for each configuration of n attributes.

As for the execution time, the heuristic algorithm considerably outperforms the exhaustive search. For all the runs, execution times of the heuristic algorithm remained below the measurement threshold of 1 *ms*. Consistent with the fact that the MF problem is NP-hard (see Theorem 6.1), the exhaustive algorithm requires exponential time in the number of attributes, and takes more than one day to compute a solution when the number of attributes reaches 37. As a consequence, we run the exhaustive algorithm only for configurations with 37 attributes at most and, based on the observed results, estimated the trend for larger numbers of attributes. Figure 14 reports the computational time of the heuristic algorithm, the computational time of the exhaustive algorithm for configurations with up to 37 attributes, and the estimated trend of the computational time of the exhaustive algorithm for configurations with more than 37 attributes.

In terms of the quality of the solution, the weight of the fragmentation computed by the heuristic algorithm is, for the values observed, always close to the optimum obtained with the exhaustive search. Figure 15 reports the weight of the solution computed by both our heuristic and the exhaustive search algorithm. On average,

the error observed over all the configurations considered is 6.22%. However, the results highlight that, in many configurations, our heuristics computes a locally minimal fragmentation that is also a minimal fragmentation.

10 Related work

Work most closely related to our is in the area of data outsourcing [21, 23]. The works proposed in this scenario mainly differ from the proposal illustrated in this paper since they are based on the assumption that data are entirely encrypted to protect their privacy. As a consequence, such proposals focus mainly on the design of techniques allowing efficient query evaluation on encrypted data. In [21, 23], the authors first propose a solution based on the definition of indexing information, stored together with the encrypted database, to be used by the DBMS to evaluate equality conditions on the encrypted data. The first support for range queries is illustrated in [16] and is based on the traditional index structure used by relational DBMSs to support range conditions, that is, B+ trees. In [16] the authors also propose a hash-based indexing technique for equality conditions. An indexing method that guarantees the efficient evaluation of both range and equality queries has been introduced in [27]. This solution uses B-trees and produces an almost flat distribution of the frequencies of index values, to limit the risk of inference exposure [8]. Different proposals (e.g., [2, 22, 28]) have been studied for providing a better support of SQL clauses, minimizing the burden for the requesting client in the query evaluation process.

The first proposal suggesting the storage of plaintext data, while enforcing confidentiality constraints, is presented in [1]. The solution introduced in [1] breaks sensitive associations by storing on two remote servers the attributes that should not be publicly visible together. The servers are assumed to belong to two different service providers, which never exchange information. The sensitive associations that cannot be split between the two servers are protected by resorting to encryption. That is, one of the attributes belonging to the sensitive association is encrypted, the encrypted attribute is stored on one server, and the encryption key on the other. The work presented in [10, 12, 14] proposes a solution that permits storing multiple fragments on a single server, thus removing the trust in the fact that the servers do not communicate. This proposal is based on the combined use of fragmentation and encryption and aims at leaving in plaintext all the attributes whose values are not considered sensitive per se (i.e., attributes that do not appear in singleton constraints). The fragmentation process is guided by a definition of minimality, aimed at reducing either the number of fragments [10] or the cost of query evaluation over fragmented data [12].

The first solution completely departing from encryption for confidentiality constraint satisfaction is presented in [13]. This proposal is based on the assumption that the data owner is willing to store a limited portion of the information. As a consequence, privacy constraints are enforced by fragmenting the data such that sensitive associations are broken (or completely stored at the owner site). The query evaluation process to adopt in this

scenario is described in [11]. The solution proposed in this paper extends the works in [11, 13], since we propose a new modeling (and therefore a different algorithm) for the fragmentation problem, which is conveniently formulated as a hypergraph 2-coloring problem.

A different, but related, line of work is represented by the proposals in [4, 5, 6], where the authors exploit functional dependencies to the aim of correctly enforcing access control policies. In [5] the authors propose a policy based classification of databases to guarantee the confidentiality of sensitive data. To this purpose, the solution proposed first determines a set of classification instances, representing the combinations of values that need to be protected. Classification instances are then used to determine if the result of the evaluation of a query contains a combinations of values that is considered sensitive and therefore access to the query result should not be permitted. In [4] the authors propose a method for the definition of privacy constraints, based on the reduction of the problem of protecting the data from inference to the problem of enforcing of access control in relational databases. In [6] the authors propose a model for the modification of the database instance visible to the user, guaranteeing both data confidentiality (as specified by the data owner’s policy) and consistency with user’s prior knowledge.

The problem of fragmenting relational databases while maximizing query efficiency has already been studied in the literature and some approaches have been proposed [25, 26]. However, these techniques are not applicable to our problem, since they are only aimed at performance optimization and do not take into consideration protection requirements. Also, the classical proposals introduced for the management of queries both in centralized and distributed systems [3, 7, 24] cannot be applied in our context since they do not take confidentiality constraints into consideration.

The work presented in this paper has some affinity with the work in [17]. Although this approach shares with our problem the common goal of enforcing confidentiality constraints on data, it is concerned with retrieving a data classification (according to a multilevel mandatory policy) that ensures sensitive information is not disclosed. Also, it does not consider the possibility of fragmenting data.

11 Conclusions

The paper presented a model and an efficient technique for enforcing confidentiality constraints defined on data that are outsourced to an external honest-but-curious server. Our approach departs from encryption and is based on the availability at the owner of local trusted storage, which can be efficiently used to store a limited set of data, just enough to guarantee privacy protection. We introduced the problem of computing a minimal fragmentation with respect to different metrics expressing the workload at the owner side, and provided a formulation of the problem in terms of a 2-coloring of a hypergraph. We also presented a heuristics that solves

the hypergraph 2-coloring problem and experimentally proved that our heuristics returns a solution close to the optimum. Several open issues still remain to be addressed such as the development of sophisticated metrics for evaluating the workload of the owner that take into consideration the selectivity of queries and of alternative solutions for guaranteeing that the outsourced data do not reveal sensitive associations. Another interesting aspect that deserves investigation concerns dynamic scenarios, that is, when either the workload of the data owner or the confidentiality constraints may change. Incremental approaches can be devised to compute a new fragmentation starting from the existing one thus avoiding complete recomputation and improper exposure due to independent views (as simply producing a new independent solution could enable the server to merge attribute views at different times and breach confidentiality constraints).

Acknowledgments

The authors would like to thank Eros Pedrini for support in the implementation of the system and in the production of the experimental results. This work was supported in part by the EU within the 7FP project “PrimeLife” under grant agreement 216483 and by the Italian Ministry of Research within the PRIN 2008 project “PEPPER” (2008SY2PH4). The work of Sushil Jajodia was supported in part by NSF grants CT-20013A, CT-0716567, CT-0716323, and CT-0627493; by AFOSR grants FA9550-07-1-0527, FA9550-09-1-0421, and FA9550-08-1-0157; and by ARO grant W911NF-09-01-0352.

References

- [1] G. Aggarwal, M. Bawa, P. Ganesan, H. Garcia-Molina, K. Kenthapadi, R. Motwani, U. Srivastava, D. Thomas, and Y. Xu. Two can keep a secret: a distributed architecture for secure database services. In *Proc. of CIDR 2005*, Asilomar, CA, January 2005.
- [2] R. Agrawal, J. Kierman, R. Srikant, and Y. Xu. Order preserving encryption for numeric data. In *Proc. SIGMOD 2004*, Paris, France, June 2004.
- [3] P.A. Bernstein, N. Goodman, E. Wong, C.L. Reeve, and Jr. J.B. Rothnie. Query processing in a system for distributed databases (SDD-1). *ACM TODS*, 6(4):602–625, December 1981.
- [4] J. Biskup, D.W. Embley, and J. Lochner. Reducing inference control to access control for normalized database schemas. *Information Processing Letters*, 106(1):8–12, March 2008.
- [5] J. Biskup and J. Lochner. Enforcing confidentiality in relational databases by reducing inference control to access control. In *Proc. of ISC 2007*, Valparaíso, Chile, October 2007.

- [6] J. Biskup and L. Wiese. Combining consistency and confidentiality requirements in first-order databases. In *Proc. of ISC 2009*, Pisa, Italy, September 2009.
- [7] S. Ceri and G. Pelagatti. *Distributed Databases: Principles and Systems*. McGraw-Hill Book Company, 1984.
- [8] A. Ceselli, E. Damiani, S. De Capitani di Vimercati, S. Jajodia, S. Paraboschi, and P. Samarati. Modeling and assessing inference exposure in encrypted databases. *ACM TISSEC*, 8(1):119–152, February 2005.
- [9] S. Cimato, M. Gamassi, V. Piuri, R. Sassi, and F. Scotti. Privacy-aware biometrics: Design and implementation of a multimodal verification system. In *Proc. of ACSAC 2008*, Anaheim, CA, December 2008.
- [10] V. Ciriani, S. De Capitani di Vimercati, S. Foresti, S. Jajodia, S. Paraboschi, and P. Samarati. Fragmentation and encryption to enforce privacy in data storage. In *Proc. of ESORICS 2007*, Dresden, Germany, September 2007.
- [11] V. Ciriani, S. De Capitani di Vimercati, S. Foresti, S. Jajodia, S. Paraboschi, and P. Samarati. Enforcing confidentiality constraints on sensitive databases with lightweight trusted clients. In *Proc. of DBSec 2009*, Montreal, Canada, July 2009.
- [12] V. Ciriani, S. De Capitani di Vimercati, S. Foresti, S. Jajodia, S. Paraboschi, and P. Samarati. Fragmentation design for efficient query execution over sensitive distributed databases. In *Proc. of ICDCS 2009*, Montreal, Canada, June 2009.
- [13] V. Ciriani, S. De Capitani di Vimercati, S. Foresti, S. Jajodia, S. Paraboschi, and P. Samarati. Keep a few: Outsourcing data while maintaining confidentiality. In *Proc. of ESORICS 2009*, Saint Malo, France, September 2009.
- [14] V. Ciriani, S. De Capitani di Vimercati, S. Foresti, S. Jajodia, S. Paraboschi, and P. Samarati. Combining fragmentation and encryption to protect privacy in data storage. *ACM TISSEC*, 13(3):22:1–22:33, July 2010.
- [15] V. Ciriani, S. De Capitani di Vimercati, S. Foresti, and P. Samarati. k -Anonymity. In T. Yu and S. Jajodia, editors, *Secure Data Management in Decentralized Systems*. Springer-Verlag, 2007.
- [16] E. Damiani, S. De Capitani di Vimercati, S. Jajodia, S. Paraboschi, and P. Samarati. Balancing confidentiality and efficiency in untrusted relational DBMSs. In *Proc. of CCS 2003*, Washington, DC, October 2003.

- [17] S. Dawson, S. De Capitani di Vimercati, P. Lincoln, and P. Samarati. Maximizing sharing of protected information. *Journal of Computer and System Sciences*, 64(3):496–541, May 2002.
- [18] M. Gamassi, M. Lazzaroni, M. Misino, V. Piuri, D. Sana, and F. Scotti. Accuracy and performance of biometric systems. In *Proc. of IMTC 2004*, Como, Italy, 2004.
- [19] M. Gamassi, V. Piuri, D. Sana, and F. Scotti. Robust fingerprint detection for access control. In *Proc. of RoboCare Workshop 2005*, Rome, Italy, May 2005.
- [20] M.R. Garey and D.S. Johnson. *Computers and Intractability; a Guide to the Theory of NP-Completeness*. W.H. Freeman, 1979.
- [21] H. Hacigümüs, B. Iyer, and S. Mehrotra. Providing database as a service. In *Proc. of ICDE 2002*, San Jose, CA, February 2002.
- [22] H. Hacigümüs, B. Iyer, and S. Mehrotra. Efficient execution of aggregation queries over encrypted relational databases. In *Proc. of DASFAA 2004*, Jeju Island, Korea, March 2004.
- [23] H. Hacigümüs, B. Iyer, S. Mehrotra, and C. Li. Executing SQL over encrypted data in the database-service-provider model. In *Proc. of SIGMOD 2002*, Madison, WI, June 2002.
- [24] D. Kossmann. The state of the art in distributed query processing. *ACM Computing Surveys*, 32(4):422–469, December 2000.
- [25] S. Navathe, S. Ceri, G. Wiederhold, and J. Dou. Vertical partitioning algorithms for database design. *ACM TODS*, 9(4):680–710, December 1984.
- [26] S. Navathe and M. Ra. Vertical partitioning for database design: a graphical algorithm. In *Proc. of SIGMOD 1989*, Portland, OR, June 1989.
- [27] H. Wang and Laks V.S. Lakshmanan. Efficient secure query evaluation over encrypted XML databases. In *Proc. of VLDB 2006*, Seoul, Korea, September 2006.
- [28] Z.F. Wang, W. Wang, and B.L. Shi. Storage and query over encrypted character and numerical data in database. In *Proc. of CIT 2005*, Shanghai, China, September 2005.

- Figure 1: An example of relation (a) and of confidentiality constraints over it (b)
- Figure 2: Data protection paradigms
- Figure 3: An example of physical fragments for relation PATIENT in Figure 1(a)
- Figure 4: Classification of the weight metrics and minimization problems
- Figure 5: An example of data (a) and query workload (b) knowledge for relation PATIENT in Figure 1(a)
- Figure 6: An example of a fragmentation lattice
- Figure 7: Weight functions and their targets
- Figure 8: CT-hypergraph representation of the instances of the different problems
- Figure 9: An example of a solution to the *Min-Att* problem
- Figure 10: Algorithm that computes a locally minimal fragmentation
- Figure 11: An example of algorithm execution
- Figure 12: An example of a solution to the *Min-Query* problem
- Figure 13: Algorithm for evaluating query q on fragmentation \mathcal{F} and an example of its execution
- Figure 14: Execution time of the heuristic and the exhaustive search algorithm
- Figure 15: Weight of the fragmentations computed by the heuristic and by an exhaustive search

PATIENT

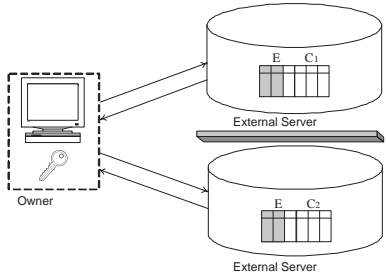
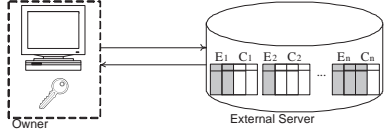
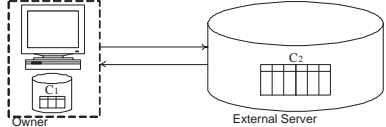
<u>SSN</u>	<u>Name</u>	<u>DoB</u>	<u>ZIP</u>	<u>Job</u>	<u>Illness</u>	<u>Treatment</u>	<u>HDate</u>
123-45-6789	A. White	1980/12/09	22030	employee	gastritis	antacid	2010/01/01
987-65-4321	B. Taylor	1975/01/18	22034	nurse	latex allergy	antihistamine	2010/01/02
963-85-2741	C. Hellman	1990/05/27	22032	manager	asthma	antihistamine	2010/01/09
147-85-2369	D. Ripley	1995/11/30	22031	employee	flu	antibiotic	2010/01/14

(a)

$c_0 = \{\text{SSN}\}$
 $c_1 = \{\text{Name}\}$
 $c_2 = \{\text{DoB, ZIP, Illness}\}$
 $c_3 = \{\text{DoB, ZIP, Treatment}\}$
 $c_4 = \{\text{Job, Illness}\}$
 $c_5 = \{\text{Illness, Treatment}\}$

(b)

Figure 1:

Scenario	Data representation
<p>non-communicating pair of servers [1]</p> 	$E \cup C_1 \cup C_2 = R$ $C_1 \cup C_2 \subseteq R$
<p>multiple fragments [10, 12, 14]</p> 	$E_1 \cup C_1 = \dots = E_n \cup C_n = R$ $C_1 \cup \dots \cup C_n \subseteq R$ $\forall i, j, i \neq j, C_i \cap C_j = \emptyset$
<p>no encryption [11, 13]</p> 	$C_1 \cup C_2 = R$

Legend C_i : plaintext attributes in F_i
 E_i : encrypted attributes in F_i

Figure 2:

F_o

tid	SSN	Name	DoB	Job	Treatment
1	123-45-6789	A. White	1980/12/09	employee	antacid
2	987-65-4321	B. Taylor	1975/01/18	nurse	antihistamine
3	963-85-2741	C. Hellman	1990/05/27	manager	antihistamine
4	147-85-2369	D. Ripley	1995/11/30	employee	antibiotic

F_s

tid	ZIP	Illness	HDate
1	22030	gastritis	2010/01/01
2	22030	latex allergy	2010/01/02
3	22034	asthma	2010/01/09
4	22031	flu	2010/01/14

Figure 3:

	Problem	Metrics	Weight function $w(\mathcal{F})$
Storage	Min-Attr	Number of attributes	$card(F_o)$
	Min-Size	Size of attributes	$\sum_{a \in F_o} size(a)$
Computation/traffic	Min-Query	Number of queries	$\sum_{q \in \mathcal{Q}} freq(q)$ <i>s.t.</i> $Attr(q) \cap F_o \neq \emptyset$
	Min-Cond	Number of conditions	$\sum_{q \in \mathcal{Q}} \sum_{co \in Co(q)} freq(q)$ <i>s.t.</i> $Attr(co) \cap F_o \neq \emptyset$

Figure 4:

Attribute a	$size(a)$
SSN	9
Name	20
DoB	8
ZIP	5
Job	18
Illness	15
Treatment	40
HDate	8

(a)

Query q	$freq(q)$	$Attr(q)$	$Co(q)$
q_1	4	Name, DoB	(Name > 'F'), (DoB < 1980/01/01)
q_2	14	Name, Job	(Name < 'G'), (Job = 'employee')
q_3	1	Name, Illness	(Name > 'C'), (Illness = 'asthma')
q_4	1	DoB, Illness	(DoB ≥ 1960/01/01), (Illness = 'measles')
q_5	15	ZIP, Illness	(ZIP = 22030), (Illness = 'flu')
q_6	4	Job, Illness	(Job = 'nurse'), (Illness = 'latex allergy')
q_7	3	DoB, Illness, Treatment	(DoB < 1970/01/01), (Illness = 'hypertension'), (Treatment = 'diet')
q_8	6	DoB, HDate, Treatment	(HDate - DoB ≥ 18y), (Treatment = 'antihistamine')
q_9	5	HDate	(HDate ≥ 1990/01/01)

(b)

Figure 5:

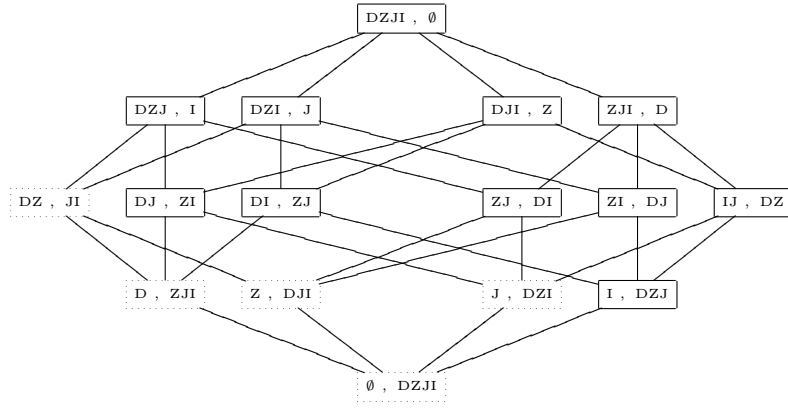


Figure 6:

Weight function	Target \mathcal{T}	$w_{\mathcal{T}}(t) \forall t \in \mathcal{T}$
$w_a(\mathcal{F})$	$\mathcal{T} = \{\{a\} a \in R\}$	$w_{\mathcal{T}}(t) = 1$
$w_s(\mathcal{F})$	$\mathcal{T} = \{\{a\} a \in R\}$	$w_{\mathcal{T}}(t) = \text{size}(a) \quad \text{s.t. } \{a\} = t$
$w_q(\mathcal{F})$	$\mathcal{T} = \{\text{Attr}(q) q \in \mathcal{Q}\}$	$w_{\mathcal{T}}(t) = \sum_{q \in \mathcal{Q}} \text{freq}(q) \quad \text{s.t. } \text{Attr}(q) = t$
$w_c(\mathcal{F})$	$\mathcal{T} = \{\text{Attr}(co) \exists q \in \mathcal{Q}, co \in Co(q)\}$	$w_{\mathcal{T}}(t) = \sum_{q \in \mathcal{Q}} \sum_{co \in Co(q)} \text{freq}(q) \quad \text{s.t. } \text{Attr}(co) = t$

Figure 7:

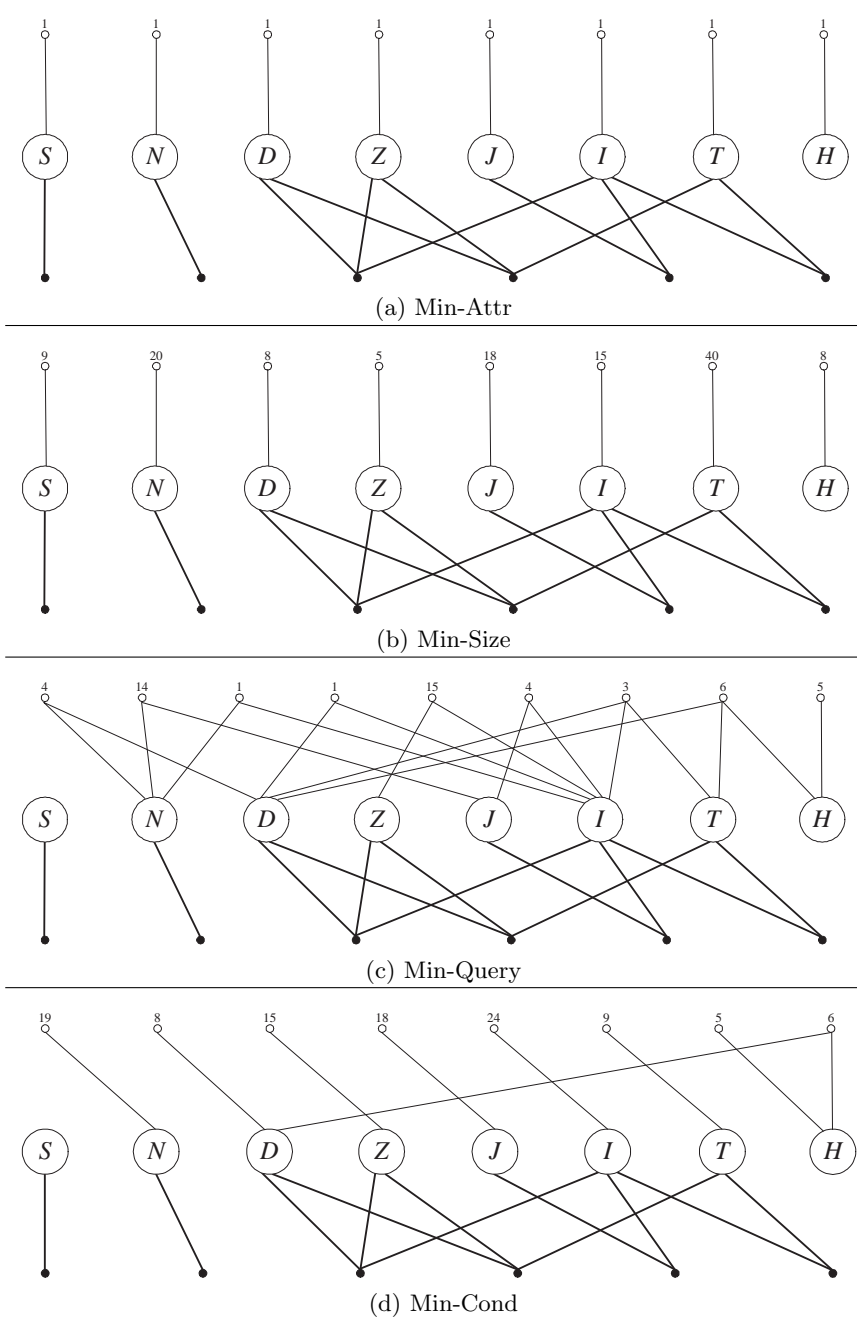


Figure 8:

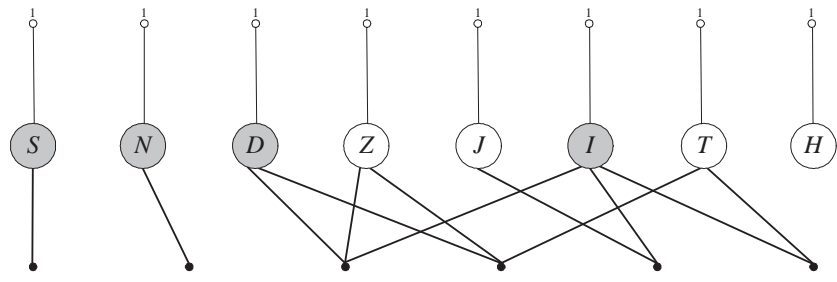


Figure 9:

```

INPUT
 $H = (\mathcal{A}, \mathcal{C} \cup \mathcal{T}, w_{\mathcal{T}})$  /* the CT-hypergraph */
*/
OUTPUT
 $Own$  /* the set of  $Own$ -vertices */

MAIN
 $\mathcal{C}' := \mathcal{C}$ 
 $Own := \emptyset$  /* initialization of the solution */
for each  $v \in \mathcal{A}$  do /* insert into  $Own$  the vertices corresponding to attributes in singleton constraints */
  if  $\exists e' \in \mathcal{C} : e' = \{v\}$  then
     $Own := Own \cup \{v\}$  /* update the solution */
    for each  $e \in \mathcal{C}$  incident to  $v$  /* delete the  $\mathcal{C}$ -edges incident to  $v$  */
       $\mathcal{C} := \mathcal{C} \setminus \{e\}$ 
    for each  $e \in \mathcal{T}$  incident to  $v$  /* delete the  $\mathcal{T}$ -edges incident to  $v$  */
       $\mathcal{T} := \mathcal{T} \setminus \{e\}$ 
     $\mathcal{A} := \mathcal{A} \setminus \{v\}$  /* delete  $v$  from  $H$  */
let  $v$  be a vertex in  $\mathcal{A}$  with minimum  $w(v)/c(v)$ 
while  $(v \neq \text{NULL}) \wedge (c(v) \neq 0)$  do /* there are still constraints to be solved */
   $Own := Own \cup \{v\}$  /* update the solution */
  for each  $e \in \mathcal{C}$  incident to  $v$  /* delete the  $\mathcal{C}$ -edges incident to  $v$  */
     $\mathcal{C} := \mathcal{C} \setminus \{e\}$ 
  for each  $e \in \mathcal{T}$  incident to  $v$  /* delete the  $\mathcal{T}$ -edges incident to  $v$  */
     $\mathcal{T} := \mathcal{T} \setminus \{e\}$ 
   $\mathcal{A} := \mathcal{A} \setminus \{v\}$  /* delete  $v$  from  $H$  */
let  $v$  be a vertex in  $\mathcal{A}$  with minimum  $w(v)/c(v)$ 
for each  $v \in Own$  do /* scan vertices in reverse order of insertion in  $Own$  */
  if  $\nexists e \in \mathcal{C}' : e \cap Own = \{v\}$  then /* check if  $v$  is redundant */
     $Own := Own \setminus \{v\}$ 
return( $Own$ )

```

Figure 10:

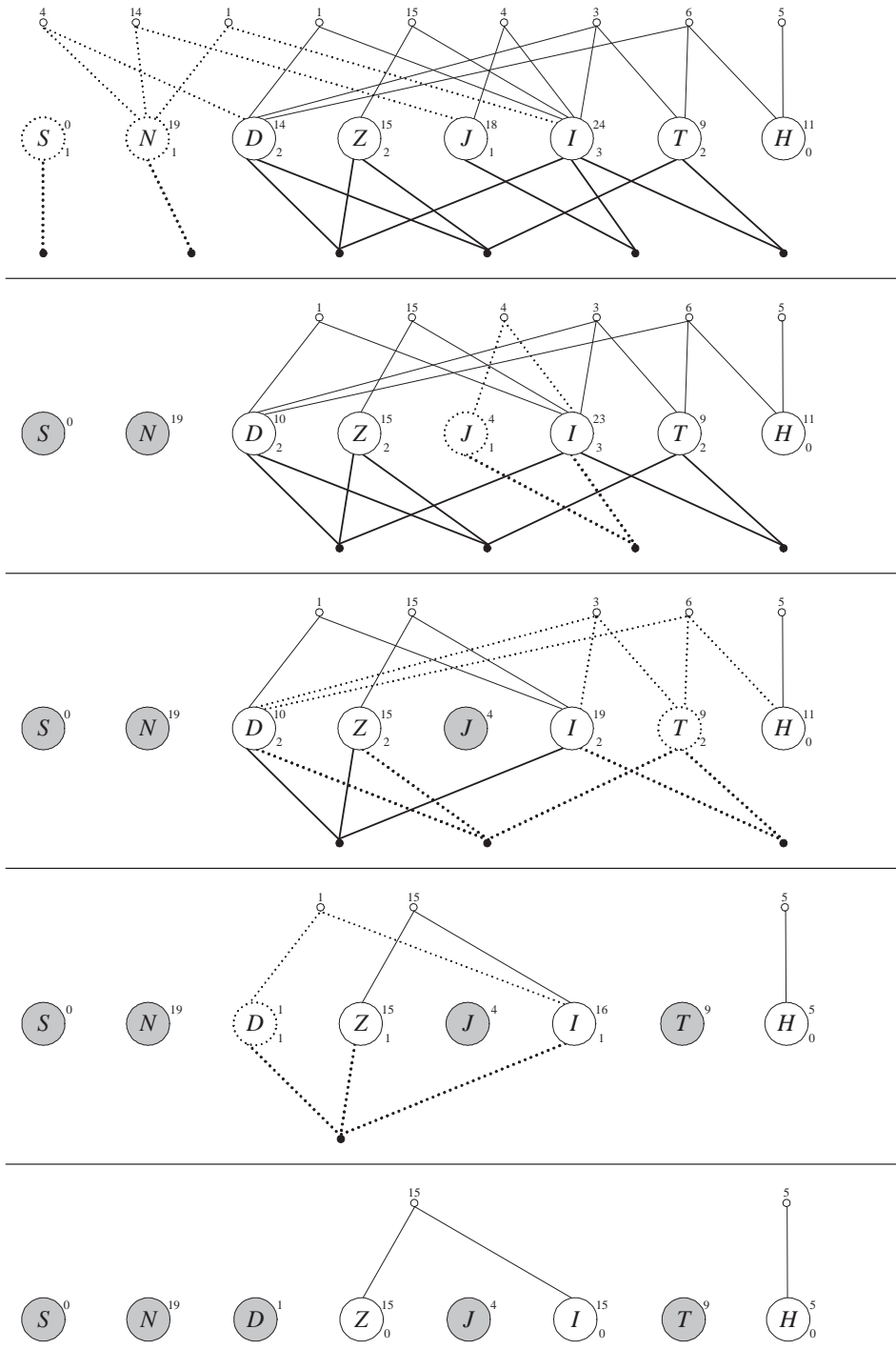


Figure 11:

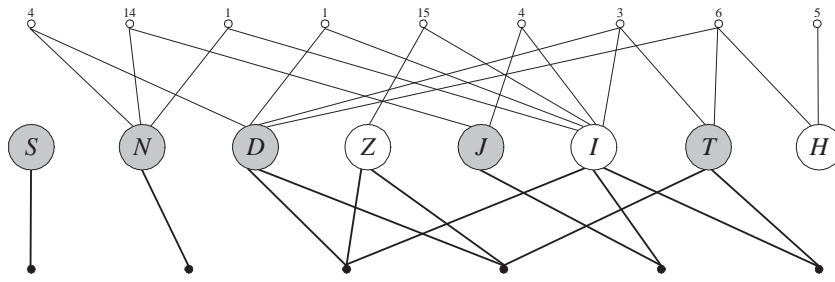


Figure 12:

Algorithm	Example
<p>Server-Owner strategy</p> <ol style="list-style-type: none"> Let $q = \text{SELECT } A$ FROM R WHERE C Split C into subexpressions $C_o, C_s,$ and C_{so} $A_{q_s} := (F_s \cap A) \cup \{a \in F_s \mid \exists co \in C_{so}, a \in Attr(co)\}$ Send $q_s = \text{SELECT } tid, A_{q_s}$ FROM F_s WHERE C_s to the external server Receive the result R_s of q_s from the server Execute $q_{so} = \text{SELECT } A$ FROM F_o JOIN R_s ON $F_o.tid = R_s.tid$ WHERE $C_o \wedge C_{so}$ Return the result R_q of q_{so} to the user 	<p>$q = \text{SELECT Name, Illness}$ FROM Patient WHERE (ZIP IN {22030,22034}) AND (Treatment = "antihistamine") AND (HDate-DoB \geq 18y)</p> <p>$C_o = \{\text{Treatment} = \text{"antihistamine"}\}; C_s = \{\text{ZIP IN } \{22030,22034\}\};$ $C_{so} = \{\text{HDate-DoB} \geq 18y\}$ $A_{q_s} = \{\text{Illness, HDate}\}$ $q_s = \text{SELECT } tid, \text{Illness, HDate}$ FROM F_s WHERE ZIP IN {22030,22034}</p> <p>$q_{so} = \text{SELECT Name, Illness}$ FROM F_o JOIN R_s ON $F_o.tid = R_s.tid$ WHERE (Treatment = "antihistamine") AND (HDate-DoB \geq 18y)</p>
<p>Owner-Server strategy</p> <ol style="list-style-type: none"> Let $q = \text{SELECT } A$ FROM R WHERE C Split C into subexpressions $C_o, C_s,$ and C_{so} $A_{q_s} := (F_s \cap A) \cup \{a \in F_s \mid \exists co \in C_{so}, a \in Attr(co)\}$ Execute $q_o = \text{SELECT } tid$ FROM F_o WHERE C_o Send the result R_o of q_o to the external server Send $q_s = \text{SELECT } tid, A_{q_s}$ FROM F_s JOIN R_o ON $F_s.tid = R_o.tid$ WHERE C_s to the external server Receive the result R_s of q_s from the server Execute $q_{so} = \text{SELECT } A$ FROM F_o JOIN R_s ON $F_o.tid = R_s.tid$ WHERE C_{so} Return the result R_q of q_{so} to the user 	<p>$q = \text{SELECT Name, Illness}$ FROM Patient WHERE (ZIP IN {22030,22034}) AND (Treatment = "antihistamine") AND (HDate-DoB \geq 18y)</p> <p>$C_o = \{\text{Treatment} = \text{"antihistamine"}\}; C_s = \{\text{ZIP IN } \{22030,22034\}\};$ $C_{so} = \{\text{HDate-DoB} \geq 18y\}$ $A_{q_s} = \{\text{Illness, HDate}\}$ $q_o = \text{SELECT } tid$ FROM F_o WHERE Treatment = "antihistamine"</p> <p>$q_s = \text{SELECT } tid, \text{Illness, HDate}$ FROM F_s JOIN R_o ON $F_s.tid = R_o.tid$ WHERE ZIP IN {22030,22034}</p> <p>$q_{so} = \text{SELECT Name, Illness}$ FROM F_o JOIN R_s ON $F_o.tid = R_s.tid$ WHERE HDate-DoB \geq 18y</p>

Figure 13:

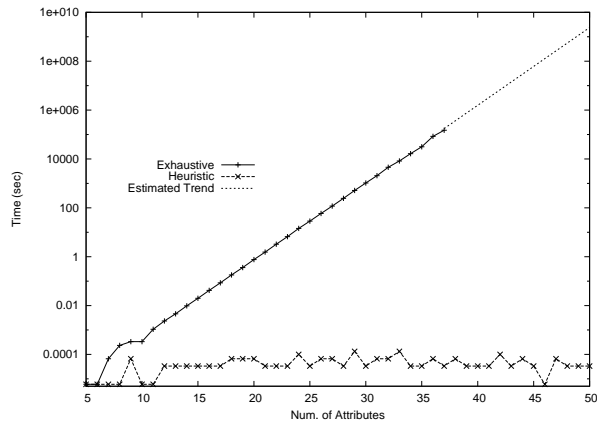


Figure 14:

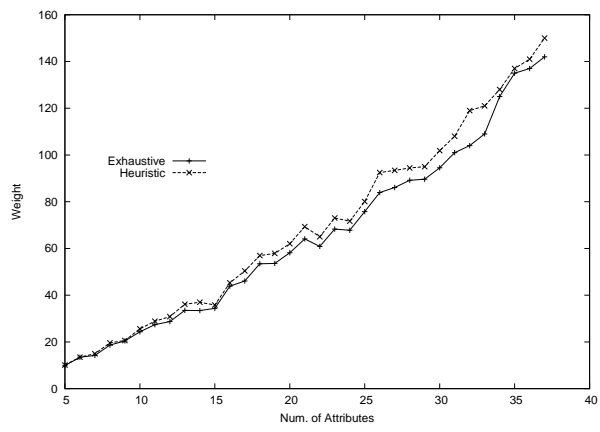


Figure 15: