Constrained Anonymization of Production Data: A Constraint Satisfaction Problem Approach

Ran Yahalom[1], Erez Shmueli[1,2] and Tomer Zrihen[1,2]

[1]Deutsche Telekom Laboratories; and the [2]Department of Information Systems Engineering, Ben-Gurion University, P.O.B. 653, Beer Sheva 84105, Israel.

Abstract. The use of production data which contains sensitive information in application testing requires that the production data be anonymized first. The task of anonymizing production data becomes difficult since it usually consists of constraints which must also be satisfied in the anonymized data. We propose a novel approach to anonymize constrained production data based on the concept of constraint satisfaction problems. Due to the generality of the constraint satisfaction framework, our approach can support a wide variety of mandatory integrity constraints as well as constraints which ensure the similarity of the anonymized data to the production data. Our approach decomposes the constrained anonymization problem into independent sub-problems which can be represented and solved as constraint satisfaction problems (CSPs). Since production databases may contain many records that are associated by vertical constraints, the resulting CSPs may become very large. Such CSPs are further decomposed into dependant sub-problems that are solved iteratively by applying local modifications to the production data. Simulations on synthetic production databases demonstrate the feasibility of our method.

1 Introduction

Testing is a cardinal stage in the life-cycle of every information system. It cannot be performed without data which is usually stored in databases. Clearly, the simplest way to provide this data to a test environment is to copy it from the production environment. However, production data often contains sensitive information which should not be exposed in a non-privileged test environment (such as an external, sometimes even foreign, testing group). Thus, the production data must be anonymized before it is copied to the test environment so that sensitive information is masked from the end-user. However, the task of anonymizing production data is non-trivial since it must also preserve certain characteristics (rules) of the original production data for the anonymized data to be useful for testing. Failing to do so may compromise the testing process. We refer to this problem as the *Constrained Anonymization* problem.

Available anonymization tools (e.g. [12,10,11,13]) provide a means of enforcing common rules, usually a set of predefined integrity rules (such as identity and reference rules)[6] or simple statistical aggregate rules (such as maintaining the average value of some field). However, application-specific rules are enforced via anonymization routines that are developed ad hoc by the user. Although these tools offer powerful scripting capabilities which can be used with such routines, a framework that allows translating a set of application-specific constraints into a set of anonymizing routines without user intervention is clearly missing. Such a framework would drastically decrease the amount of implementation effort required by the user.

In this paper we propose a novel approach for anonymizing constrained production data based on the framework of constraint satisfaction problems (CSPs). Following the formal definition in [14], a CSP is defined by a set of variables, $\{x_1, x_2, \ldots, x_n\}$ and a set of constraints, $\{c_1, c_2, \ldots, c_m\}$. Each variable x_i has a nonempty domain D_i of possible values. Each constraint c_i involves some subset of the variables and specifies the allowable combinations of values for that subset. A solution to the CSP is an assignment $\{x_1 = v_1, x_2 = v_2, \ldots, x_n = v_n\}$ of values to all variables that does not violate any constraint (known as a complete and consistent assignment). Given the sensitive information and rules in the production data, we represent and solve the anonymization problem as a CSP. Harnessing the power of the well established CSP framework allows our approach to cope with a wide variety of anonymization constraints that are supported by this framework.

Two other well known methods for achieving privacy are data encryption and data generalization (which is commonly used in Privacy Preserving Data Publishing[15]). However these methods are inappropriate for solving the constrained annonymization problem. Data encryption cannot be used because encrypted data will not necessarily conform to the rules defined on the production data and once it is decrypted, the original production data is exposed. Likewise, when data is generalized, it may violate even the most basic rules such as value set or identity integrity rules[6].

The remainder of this paper is organized as follows. Section 2 gives a formal definition of the constrained anonymization problem. Section 3 presents the details of our CSP-based approach, including a demonstration on a toy example. In Section 4, we evaluate the performance of our method and demonstrate its feasibility. Section 5 reviews the related work. Finally, we summarize our results in Section 6 and describe future research directions.

2 Problem Statement

In this section we formally define the problem of *Constrained Anonymization* of production data:

Definition 1. *Sensitive Field (SF): a field in PD whose values must not be disclosed.*

Definition 2. *Rule (R): A relation/condition defined over a set of fields in PD, which specifies the legal values that cells in these fields can have. In this paper we focus on two types of rules: Integrity and Similarity. An integrity rule defines a condition which must be met for the data to be unimpaired, complete, sound and compliant. An excellent discussion on integrity rules can be found in [6]. For example, an identity rule on field f that requires all values to be different. A similarity rule is aimed at making the anonymized data similar to the production data. For example, a rule that requires the average of field f in the anonymized data to be the same as in the production data.*

Definition 3. *Constrained Anonymization: Given a production database PD, a set of sensitive fields SFs and a set of rules Rs, the objective is to derive the test database,* *denoted by TD, from PD so that all fields in SFs are anonymized (i.e., their original values in the production database are unknown) and all rules in Rs are enforced.*

We assume that the user provides SI and Rs. Extracting SI and Rs automatically from the production data is not within the scope of this paper.

3 CSP Approach for Constrained Anonymization

We now propose a method for solving the constrained anonymization problem by reformulating it as a CSP. In our setting, the user provides the following three inputs: (1) The production database PD, (2) The set of sensitive fields SFs and (3) a set of rules Rs.

Our method first duplicates the test database, denoted by TD, from PD. Then our method decomposes Rs into independent rule subsets RSs. Each rule subset RS that contains at least one sensitive field corresponds to a CSP. First, the set of CSP variables is defined and then RS is translated into CSP constraints defined over these variables. This CSP, denoted by CSP_{RS}, is then further decomposed into separate sub-CSPs, denoted by $CSPS_{RS}$. Finally, $CSP_i \in CSPS_{RS}$ is solved and the solutions are stored in TD. The union of these solutions is a solution to CSP_{RS} and the union of the solutions to all CSP_{RS} is a solution to the whole anonymization problem. The complete method is described in Algorithm 1 and its different stages are outlined in Fig. 1. In the following subsections we describe the main stages of our method in detail. Subsection 3.5 illustrates the whole process by a toy example.

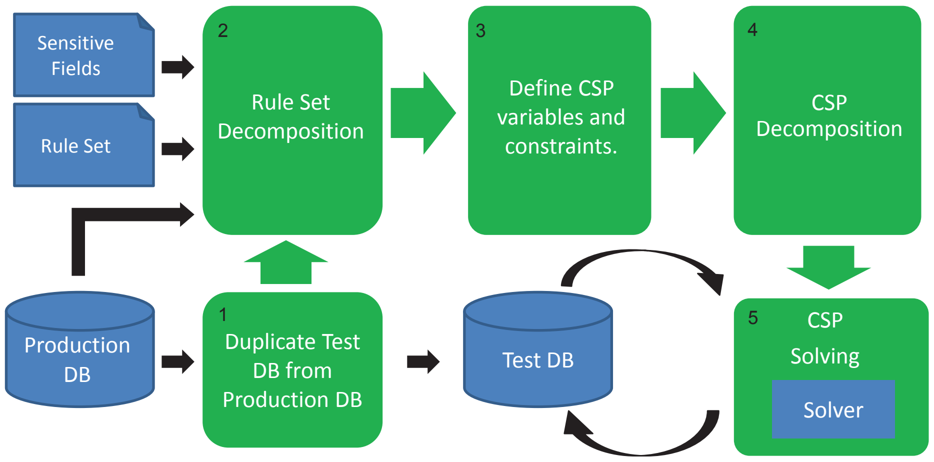

Fig. 1: The different stages of the proposed method.

Algorithm 1 constrainedAnonymizationOfProductionData(PD,SFs,Rs)

Input

PD: the production database PD.

SFs: the set of sensitive fields.

Rs: the set of rules.

Output

TD: the test database.

```
 1: $TD \leftarrow PD$;
 2: $RSs \leftarrow$ decomposeRuleSet($Rs$);
 3: for all ($RS \in RSs \wedge SF \cap F_{RS} \neq \emptyset$)
 4:     $X_{CSP_{RS}}, C_{CSP_{RS}}, CSPs \leftarrow \emptyset$;
 5:     $X_{CSP_{RS}} \leftarrow$ defineCSPvariables($RS$,$SF$,$TD$);
 6:     for all ($r \in RS$)
 7:         $C_r \leftarrow$ defineCSPconstraints($r$,$X_{CSP_{RS}}$);
 8:         $C_{CSP_{RS}} \leftarrow C_{CSP_{RS}} \cup C_r$;
 9:     end for
10:     $CSP_{RS} \leftarrow < X_{CSP_{RS}}, C_{CSP_{RS}} >$;
11:     $C_{vertical}, CSPS_{RS} \leftarrow$ decomposeCSP($CSP_{RS}$);
12:     if ($C_{vertical} = \emptyset$)
13:         for all ($CSP_i \in CSPS_{RS}$)
14:             solve($CSP_i$,$TD$);
15:         end for
16:     else
17:         solveByLocalModification($CSPS_{RS}$, $C_{vertical}$,$TD$);
18:     endif
19: end for
20: return $TD$;
```

3.1 Decomposing the rule set

Rules are defined over the fields of PD and either represent relationships within a field or between different fields. It is possible to decompose Rs into disjoint rule subsets RSs such that fields of any two rules in the same subset have to be anonymized together (because they transitively constrain each other through the rules of the subset) and fields of rules in different subsets can be anonymized independently. This decomposition can be efficiently done by finding the connectivity components of the rules set graph, G_{Rs}, in which each rule is represented by a vertex and two vertices are connected if the corresponding rules involve a common field. This rules set decomposition is carried out in Algorithm 1 by the decomposeRuleSet procedure.

3.2 Defining the CSPs

Some, but not necessarily all of the rule subsets resulting from the previous stage are translated into CSPs. Denoting F_{RS} as the set of fields which are associated with rules in RS, when $SF \cap F_{RS} \neq \emptyset$, RS is translated into CSP_{RS}. Otherwise, there is no need to translate *RS* into CSP and the TD values for fields in F_{RS} are untouched. Moreover, for any field $f \notin \bigcup_{RS \in RSs} F_{RS}$, if $f \in SF$, TD values for field f are randomly drawn from the appropriate domain, otherwise they are left untouched. In words: F_{RS} which does not contain any sensitive field is simply copied from *PD*; fields who do not have any rule defined over them and are sensitive are simply generated; fields who do not have any rule defined over them and are non-sensitive are simply copied from *PD*.

Defining the CSP variables A CSP variable is defined for each cell in F_{RS}. We denote x_{ij}^T as the variable that corresponds to the cell in record i, field j and table T. The domains of x_{ij}^T automatically follow from the data type of the values in the j'th field of T and, optionally, from other schema definitions (such as value bounds). The above is conducted in Algorithm 1 by the defineCSPvariables procedure which returns the set of these variables, denoted by $X_{CSP_{RS}} = \{x_{ij}^T\}$.

Defining the CSP constraints Whereas rules are defined over fields, the CSP constraints are defined over variables that correspond to individual cells. Thus, after $X_{CSP_{RS}}$ is defined, each $r \in RS$ is translated into the corresponding CSP constraints (conducted in Algorithm 1 by the defineCSPconstraints procedure). For example, consider a rule requiring that a field f_j is unique. This rule will be translated into a single n-ary $allDifferent$[2] constraint: $allDifferent(x_{0j}^T, x_{1j}^T ..., x_{(n-1)j}^T)$, where n is the number of records in table T. Another example can be a rule that requires the value of field f_j to be larger than the value of field f_k. This rule will be translated into n binary gt [2] constraints: $\{gt(x_{0j}^T, x_{0k}^T), gt(x_{1j}^T, x_{1k}^T), ..., gt(x_{(n-1)j}^T, x_{(n-1)k}^T)\}$.

Due to the tabular structure of the production DB, the CSP constraints may be of four possible types:

1. Horizontal: constraints which contain at least one subset of different variables $\{x_{ij}^T\}$ that have the same i. Constraints for which all variables have the same i are called strictly horizontal, e.g. the above gt constraint.
2. Vertical: constraints which contain at least one subset of different variables $\{x_{ij}^T\}$ that have the same j. Constraints for which all variables have the same j are called strictly vertical, e.g. the above $allDifferent$.
3. Mixed: constraints that are both vertical and horizontal.
4. Cross-table: constraints involving variables from different tables. These constraints may also be any of the other three constraint types.

3.3 Decomposing CSP_{RS}

Once the variables and constraints of CSP_{RS} have been defined, we need to solve it with a CSP solver[1]. In many cases, CSP_{RS} will contain a large amount of variables.

[1] There are many available CSP packages (both commercial and not) from which the solver can be chosen. Deciding which package to use must be carefully considered because not all CSP packages support the required variable domains and/or constraints. Thus, the choice of CSP package can affect the extendability of the method.

As a very common example, consider a CSP resulting solely from the $allDifferent$ constraint previously mentioned and its n associated variables. Solving such a CSP is impractical in many cases due to the overwhelming amount of variables. In general, it is reasonable to assume that for any solver s, there exists some manageable number of variables k_s (which may vary between different solvers) for which any CSP with more than k_s variables cannot be handled. Thus, our method further decomposes CSP_{RS} into a set of separate sub-CSPs, denoted by $CSPS_{RS}$, where the number of variables in each sub-CSP does not exceed k_s (Algorithm 2). This decomposition is based on the standard CSP decomposition into separate sub-problems according to its constraint graph[14]. We begin by finding the connectivity components of the constraint graph associated with CSP_{RS}. Then, constraints from $C_{CSP_{RS}}$ whose variables are part of a connectivity component of size $> k_s$ are removed from $C_{CSP_{RS}}$. If we assume that the total number of fields in PD is $\leq k_s$, only vertical constraints will be removed. The removed constraints are stored in $C_{vertical}$ and the standard decomposition is applied on CSP_{RS} (with the non-vertical set of constraints). This ensures that each $CSP_i \in CSPS_{RS}$ will involve less than k_s variables. We then iteratively merge any two sub-CSPs whose combined number of variables does not exceed k_s[2].

3.4 Solving $CSPS_{RS}$

If $C_{vertical}$ is empty, all $CSP_i \in CSPS_{RS}$ can be solved independently, even simultaneously on different machines. That is, each CSP_i is formulated in terms of the CSP solver, solved and the results are stored in TD (this is done by the $solve$ procedure used in Algorithm 1). However, if $C_{vertical}$ is not empty, the sub-CSPs in $CSPS_{RS}$ are dependant and must be solved as one CSP. Fortunately, in the context of Constrained Anonymization, we can modify the sub-CSPs so that the resulting sub-CSPs are still dependant but can be solved separately in a sequential manner. A key aspect of our scenario is the fact that there always exists at least one known solution to any sub-CSP corresponding to the relevant values in PD. Our method takes advantage of this solution by repeatedly attempting to make local modifications to it. We refer to this as the local modifications, or LM heuristic which is described in Algorithm 3. More specifically, for each $CSP_i \in CSPS_{RS}$, we derive and solve a new CSP, denoted by CSP_i^*, for which $X_{CSP_i^*} = X_{CSP_i}$ and $C_{CSP_i^*} = C_{CSP_i} \cup C_{vertical}$, where any variable from $C_{vertical}$ not included in $X_{CSP_i^*}$ is replaced by its corresponding value from TD. Since CSP_i^* has no more than k_s variables and at least one solution, it is necessarily tractable. If CSP_i^* has exactly one solution, it will necessarily be the set of values currently in TD. Otherwise, the solution will constitute a local modification that differs from TD to a varying extent (by up to k_s different values).

3.5 A Toy Example

We now demonstrate our method on a toy production database as illustrated in Fig. 2. We denote the EMPLOYEE table by T^E and the BUDGET table by T^B. We assume

[2] It has been our experience that, when possible, solving fewer problems with a larger number of variables outperforms solving more problems with a smaller number of variables.

Algorithm 2 decomposeCSP(X,C)

Input

X: variable set of the CSP being decomposed.

C: constraint set of the CSP being decomposed.

Output

$CSPS_{RS} = \{CSP_i | CSP_i =< X_i, C_i >\}$: the set of sub-problems for which $|X_{CSP_i}| \leq k_s, \forall i$.

$C_{vertical}$: the set of vertical constraints from C that link the sub-problems in $CSPS_{RS}$.

```
 1: $C_{vertical}, CSPS_{RS} \leftarrow \emptyset$;
 2: for all ($c \in C$)
 3:     if ($c$ is vertical and $X_c$ are part of a connectivity component of size $> k_s$)
 4:         $C_{vertical} \leftarrow C_{vertical} \cup \{c\}$;
 5:     end if
 6: end for
 7: while ($C \neq \emptyset$)
 8:     remove constraint $c$ from $C$;
 9:     for all ($x \in X_c$)
10:         if ($\exists CSP_i \in CSPs$ such that $x \in X_i$)
11:             $C_i \leftarrow \{c\} \cup C_i$;
12:             $X_i \leftarrow X_i \cup \{x_j | x_j \in X_c\}$;
13:         else
14:             $CSP_i \leftarrow < \{x_j | x_j \in X_c\}, \{c\} >$;
15:             $CSPS_{RS} \leftarrow CSPS_{RS} \cup \{CSP_i\}$;
16:         end if
17:     end for
18: end while
19: while ($\exists CSP_i, CSP_j \in CSPS_{RS} : |X_{CSP_i}| + |X_{CSP_j}| \leq k_s$)
20:     $CSP_i \leftarrow < (X_{CSP_i} \cup X_{CSP_j}), (C_{CSP_i} \cup C_{CSP_j}) >$;
21: end while
22: return $CSPS_{RS}$,$C_{vertical}$;
```

that all fields in the production DB except *age* and *percent_fulltime* are sensitive and that *Rs* only consists of the following integrity rules:

1. The *id* field of T^B is a unique primary key.
2. The *id* field of T^E is a unique primary key.
3. The *id* field of T^E must be a valid cellular phone number that conforms to the following regular expression pattern: "05[0247][0-9][0-9][0-9][0-9][0-9][0-9]".
4. The *hire_date* must be a valid date between the 1/1/2007 and the 12/31/2010.
5. The *end_date* must be a valid date between the 1/1/2007 and the 12/31/2010.
6. The *hire_date* must precede the *end_date*.
7. The following equation must hold: $\frac{fulltime\_salary \cdot percent\_fulltime}{100} = monthly\_pay$.
8. The *budget_id* field is a foreign key which references the primary key in T^B.
9. For any budget b, the sum of *monthly_pay* for all employees funded by b must be at most the value of the *value* field corresponding to b.

Algorithm 3 solveByLocalModification($CSPS_{RS}$,$C_{vertical}$,TD)

Input

$CSPS_{RS} = \{CSP_i | CSP_i =< X_i, C_i >\}$: a set of CSP sub-problems (generated by the decomposition stage) for which $|X_{CSP_i}| \leq k_s, \forall i$.

$C_{vertical}$: the set of vertical constraints that link the sub-problems in $CSPS_{RS}$.

TD: the test database.

```
1:  for all ($CSP_i \in CSPS_{RS}$)
2:      $X_{CSP_i^*} \leftarrow X_{CSP_i}$;
3:      $C_{verticalTemp} \leftarrow C_{vertical}$;
4:      for all ($\{x | x \in X_c \wedge c \in C_{verticalTemp} \wedge x \notin X_{CSP_i^*}\}$)
5:          replace $x$ with its corresponding cell value in $TD$;
6:      end for
7:      $C_{CSP_i^*} \leftarrow C_{CSP_i} \cup C_{verticalTemp}$;
8:      $CSP_i^* \leftarrow< X_{CSP_i^*}, C_{CSP_i^*} >$;
9:      solve($CSP_i^*$);
10: end for
```

	id	*value*
0	0	15717
1	1	14287
i / *j*	**0**	**1**

(a) The BUDGET table.

	id	*age*	*hire_date*	*end_date*	*fulltime_salary*	*percent_fulltime*	*monthly_pay*	*budget_id*
0	546410616	30	20100130	20101031	18000	59	10620	0
1	521972696	22	20100605	20100915	15250	86	13115	1
2	508207463	25	20101002	20101026	5700	72	4104	0
i / *j*	**0**	**1**	**2**	**3**	**4**	**5**	**6**	**7**

(b) The EMPLOYEE table.

Fig. 2: A production database containing two tables that store company records of employees and budgets by which their salaries are funded. The record (denoted by i) and field (denoted by j) indices of the table cells are also indicated.

Note that the resulting anonymized test data may greatly differ from the original production data because no similarity rules are defined. Also note that since this toy example is very small, we neither apply the rule set decomposition nor the *LM* approach. The resulting CSP variables are:

1. $\{x_{ij}^B | i = 0,1 \wedge j = 0,1\}$.
2. $\{x_{ij}^E | i = 0,1,2 \wedge j = 0,2,3,4,5,6,7\}$.

and the CSP constraints are:

1. $allDifferent(x_{00}^B, x_{10}^B)$.
2. $allDifferent(x_{00}^E, x_{10}^E, x_{20}^E)$.
3. $\{regular(regexp_1, x_{i0}^E) | i = 0,1,2\}$ where $regexp_1$ = "05[0247][0-9][0-9][0-9][0-9][0-9][0-9]".

4. $\{regular(regexp_2, x^E_{ij}) | i = 0,1,2 \wedge j = 2,3\}$ where $regexp_2$ = "20(0[7-9]|10)(0[1-9]|1[0-2])(0[1-9]|1[0-9]|2[0-9]|30|31)".
5. $\{x^E_{i2} < x^E_{i3} | i = 0,1,2\}$.
6. $\{\frac{x^E_{i4} \cdot x^E_{i5}}{100} = x^E_{i6} | i = 0,1,2\}$.
7. $\{x^E_{i7} = x^B_{00} | i = 0,2\}$and $x^E_{17} = x^B_{10}$.
8. $x^E_{06} + x^E_{26} \leq x^B_{01}$ and $x^E_{16} \leq x^B_{11}$.

The *CSP* sub-problems after CSP decomposition are:

1. $CSP_1 =< \{x^B_{00}, x^B_{10}, x^E_{07}, x^E_{17}, x^E_{27}\}, \{allDifferent(x^B_{00}, x^B_{10}), x^E_{07} = x^B_{00}, x^E_{17} = x^B_{10}, x^E_{27} = x^B_{00}\} >$.
2. $CSP_2 =< \{x^E_{00}, x^E_{10}, x^E_{20}\}, \{allDifferent(x^E_{00}, x^E_{10}, x^E_{20}), regular(regexp_1, x^E_{00}), regular(regexp_1, x^E_{10}), regular(regexp_1, x^E_{20})\} >$.
3. $CSP_3 =< \{x^E_{02}, x^E_{03}\}, \{x^E_{02} < x^E_{03}, regular(regexp_2, x^E_{02}), regular(regexp_2, x^E_{03})\} >$.
4. $CSP_4 =< \{x^E_{12}, x^E_{13}\}, \{x^E_{12} < x^E_{13}, regular(regexp_2, x^E_{12}), regular(regexp_2, x^E_{13})\} >$.
5. $CSP_5 =< \{x^E_{22}, x^E_{23}\}, \{x^E_{22} < x^E_{23}, regular(regexp_2, x^E_{22}), regular(regexp_2, x^E_{23})\} >$.
6. $CSP_6 =< \{x^E_{04}, x^E_{05}, x^E_{06}, x^E_{24}, x^E_{25}, x^E_{26}, x^B_{01}\}, \{\frac{x^E_{04} \cdot x^E_{05}}{100} = x^E_{06}, \frac{x^E_{24} \cdot x^E_{25}}{100} = x^E_{26}, x^E_{06} + x^E_{26} \leq x^B_{01}\} >$.
7. $CSP_7 =< \{x^E_{14}, x^E_{15}, x^E_{16}, x^B_{11}\}, \{\frac{x^E_{14} \cdot x^E_{15}}{100} = x^E_{16}, x^E_{16} \leq x^B_{11}\} >$.

Using the CSP solver provided in the CHOCO package [1], each sub-problem is solved and the results are inserted into the test DB as illustrated in Fig. 3. A close inspection will show that all values of the test DB indeed satisfy all *Rs*.

	id	*value*
0	14	19170
1	13	14307
i* \ *j	**0**	**1**

(a) The anonymized BUDGET table.

	id	*age*	*hire_date*	*end_date*	*fulltime_salary*	*percent_fulltime*	*monthly_pay*	*budget_id*
0	576676653	30	20100318	20100429	11800	53	6254	14
1	544324350	22	20100431	20100831	14350	84	12054	13
2	570192018	25	20081021	20101128	13400	55	7370	14
i* \ *j	**0**	**1**	**2**	**3**	**4**	**5**	**6**	**7**

(b) The anonymized EMPLOYEE table.

Fig. 3: The test DB resulting from anonymizing the production DB of Fig. 2.

4 Experimental Results

To demonstrate the feasibility of our method, we simulated the anonymization of production databases, based on the toy example of section 3.5. The production databases were generated by a variation of our anonymization method which relies on the *IG* heuristic described in Appendix A. The production databases that were generated differ by the number of records they contain in the *EMPLOYEE* (n) tables (the number of records in the *BUDGET* table was always $n/10$). Each production database, was anonymized using the *LM* heuristic and without applying any heuristic (*NH*).

All experiments were conducted on an Intel Core 2 Duo CPU 2.4 GHz personal computer with 3 GB of RAM, running Windows 7 Enterprise and MySQL 5.136 with default settings. For the solving stage, we used the CHOCO CSP solver [1], version 2.1.1 with $k_s = 100$ (which was determined in preliminary experiments).

Table 1 compares the time required for anonymization of each production database, using *LM* and *NH*. Since *Rs* includes two *allDifferent* constraints, it led to large CSP_{RS}. Even for relatively small sized n (i.e., 5,000), *NH* was not able to complete its execution (denoted by *NA* in the table). Thus, we could evaluate *NH* only for the experiments involving relatively few records (less than 5,000). Regarding *LM*, even though our example contained several rules of different types and despite the fact that CSP is NP-hard in general, the solving process was completed in a reasonable time.

Table 1: Execution time (in seconds) required to anonymize the different production DBs in each experiment.

Heuristic \ n	100	1,000	5,000	10,000	100,000
NH	2	35	NA	NA	NA
LM	2	13	70	141	2179

5 Related Work

As opposed to encryption and generalization, which derive the test values by modifying the corresponding production values, our CSP-based approach falls within the category of data generation methods. Data generation methods derive the test value of each cell in the database with complete disregard to the original production value of that cell. A number of data generation methods have been published, each focusing on aspects such as data dependencies or fast generation of a large amount of data.

In [9], the authors introduce the data dependency graph and describe a modified topological sort used to determine the order in which data is generated. However, it is unclear how this method can handle: (1) cycles in the graph; (2) rules that involve several fields in which no field is explicitly represented as a formula of the others, for example, $X^2 + Y^2 = 1$. In [4], the authors present a C-like Data Generation Language (DGL) which allows the generation of data using the definition of its data types, rows, iterators and distributions. However, dependencies in data (such as foreign keys) must be explicitly handled by the user. In [8] the authors suggest a parallel synthetic data generator (PSDG) designed to generate "industrial sized" data sets quickly using cluster computing. PSDG depends on an XML based synthetic data description language (SDDL) which codifies the manner in which generated data can be described and constrained. However, PSDG suffers from the same limitations of [9]. In [3], the authors suggest a system to generate query aware data. The system is given a query and produces data that is tailored to the specific query. However it is not very useful for general purpose generation of test data for ad hoc testing. In [7], a hardware based technique to quickly generate large databases on multi-processor systems is described. The authors focus on low-level implementation details (such as process spawning and table partitioning strategies) to generate data using a dedicated hardware architecture. In [5], the authors focus on generating data for performance tests. More specifically, they try to generate data which preserves the query optimizer characteristics of the original data. The generation process must preserve three data characteristics: consistence, monotony and dependence between columns. To summarize, as dependencies and rules defined over fields become more complicated, none of the existing methods is general enough to handle them.

6 Summary and future work

We have presented a novel, CSP-based approach for anonymizing production data which involves complex constraints. Due to the power of CSP, our method can support a wide variety of general constraints that are frequently encountered in production data. Such constraints include not only mandatory integrity constraints but also constraints that can be defined in order to make the test data similar to the production data.

After formulating the Constrained Anonymization problem as a CSP, we decompose it into separate and smaller sub-CSPs. Often, the resulting sub-CSPs are independent and therefore simpler to solve. Nonetheless, sub-CSPs which contain vertical constraints will remain dependent and intractable due to the overwhelming amount of variables. In this case, our method modifies the sub-CSPs so that they are still dependant but can be solved separately in a sequential manner. This is possible due to the fact that there always exists at least one known solution to any sub-CSP which can be taken from the production database.

We have demonstrated the feasibility of our method by simulating the anonymization of synthetic production DBs (see Table 1). Our simulations emphasize the necessity of the proposed heuristics when the production DB contains a typically large number of records.

In future work we intend to: (1) Develop an automated method for identifying *Rs* (both integrity rules and similarity rules), perhaps by analyzing the DB schema and/or applying various data-mining solutions to the production data; (2) Formally define a measure of production data anonymization which will allow us to evaluate the quality of the anonymization algorithm; (3) Improve the performance of our method by solving independent sub-CSPs in parallel on distributed machines; (4) Evaluate our method on a real (or a benchmark) database.

References

1. Choco solver. http://choco.emn.fr, 2010.
2. N. Beldiceanu, M. Carlsson, and J.X. Rampon. Global constraint catalog. 2005.
3. C. Binnig, D. Kossmann, E. Lo, and M. T. Özsu. Qagen: generating query-aware test databases. In *SIGMOD '07: Proceedings of the 2007 ACM SIGMOD international conference on Management of data*, pages 341–352, New York, NY, USA, 2007. ACM.
4. N. Bruno and S. Chaudhuri. Flexible database generators. In *VLDB '05: Proceedings of the 31st international conference on Very large data bases*, pages 1097–1107. VLDB Endowment, 2005.
5. M. Castellanos, B. Zhang, I. Jimenez, P. Ruiz, M. Durazo, U. Dayal, and L. Jow. Data desensitization of customer data for use in optimizer performance experiments. In *Proceedings of the 26th IEEE International Conference on Data Engineering*, 2010.
6. K. Duncan and D. Wells. A Rule-Based Data Cleansing. *Journal of Data Warehousing*, 4(3):146–159, 1999.
7. J. Gray, P. Sundaresan, S. Englert, K. Baclawski, and P.J. Weinberger. Quickly generating billion-record synthetic databases. *ACM SIGMOD Record*, 23(2):252, 1994.
8. J. E. Hoag and C. W. Thompson. A parallel general-purpose synthetic data generator. *SIGMOD Rec.*, 36(1):19–24, 2007.
9. K. Houkjar, K. Torp, and R. Wind. Simple and realistic data generation. In *Proceedings of the 32nd international conference on Very large data bases*, page 1246. VLDB Endowment, 2006.
10. Camouflage Software Inc. Camouflage transformers. Data Sheet, http://www.datamasking.com, 2009.
11. Camouflage Software Inc. Enterprise-wide data masking with the camoufl age translation matrix. Data Sheet, http://www.datamasking.com, 2009.
12. Camouflage Software Inc. Secure analytics - maximizing data quality & minimizing risk for banking and insurance firms. White Paper, http://www.datamasking.com, 2009.
13. Grid-Tools GridTools Ltd. Simple data masking. Data Sheet, http://www.grid-tools.com, 2009.
14. S.J. Russell, P. Norvig, J.F. Canny, J. Malik, and D.D. Edwards. *Artificial intelligence: a modern approach*. Prentice hall Englewood Cliffs, NJ, 1995.
15. K. Wang, R. Chen, and P.S. Yu. Privacy-Preserving Data Publishing: A Survey on Recent Developments. 2010.

A The incremental generation (*IG*) heuristic

Consider a setting in which we would like to utilize our CSP approach to generate a constrained database. In this case the *LM* heuristic cannot be used since it relies on modifying an existing solution. Thus, we need a different heuristic to handle $CSPS_{RS}$ (resulting from the decomposition of CSP_{RS}) when $C_{vertical} \neq \emptyset$. For some special cases of $C_{vertical}$, we can apply an iterative process in which we solve $CSP_i \in CSPS_{RS}$ based on previously solved $CSP_j, \forall j < i$. More specifically, in the i'th iteration, we solve a new CSP, denoted by CSP_i^*, for which $X_{CSP_i^*} = X_{CSP_i}$ and $C_{CSP_i^*} = C_{CSP_i} \cup C_{vertical}$. For any variable belonging to a constraint $c \in C_{vertical}$ not included in $X_{CSP_i^*}$, if $x \in X_{CSP_j^*}$ and $j < i$, it is replaced by its solution in CSP_j^*, otherwise it is removed from X_c. This *Incremental Generation* (*IG*) approach is described in Algorithm 4. Note that when $C_{vertical}$ contains only one $allDifferent$ constraint (like in our evaluation setting), the *IG* heuristic will find a solution to $CSPS_{RS}$ and $C_{vertical}$, if and only if, a solution exists to CSP_{RS}.

Algorithm 4 solveByIncrementalGeneration($CSPS_{RS}$,$C_{vertical}$,TD)

Input

$CSPS_{RS} = \{CSP_i | CSP_i =< X_i, C_i >\}$: a set of CSP sub-problems (generated by the decomposition stage) for which $|X_{CSP_i}| \leq k_s, \forall i$.

$C_{vertical}$: the set of vertical constraints that link the sub-problems in $CSPS_{RS}$.

TD: the empty test database.

```
1:  for all ($CSP_i \in CSPS_{RS}$)
2:      $X_{CSP_i^*} \leftarrow X_{CSP_i}$;
3:      $C_{verticalTemp} \leftarrow C_{vertical}$;
4:      for all ($\{x | x \in X_c \wedge c \in C_{verticalTemp} \wedge x \notin X_{CSP_i^*}\}$)
5:          if (the cell corresponding to $x$ in $TD$ is not empty)
6:              replace $x$ with its corresponding cell value in $TD$;
7:          else
8:              $X_c \leftarrow X_c \setminus \{x\}$;
9:          end if
10:     end for
11:     $C_{CSP_i^*} \leftarrow C_{CSP_i} \cup C_{verticalTemp}$;
12:     $CSP_i^* \leftarrow < X_{CSP_i^*}, C_{CSP_i^*} >$;
13:     solve($CSP_i^*$);
14: end for
```