Statistics and Computing **13**: 363–370, 2003

Preserving confidentiality of high-dimensional tabulated data: Statistical and computational issues

ADRIAN DOBRA*, ALAN F. KARR and ASHISH P. SANIL

National Institute of Statistical Sciences, Research Triangle Park, NC 27709-4006, USA

Dissemination of information derived from large contingency tables formed from confidential data is a major responsibility of statistical agencies. In this paper we present solutions to several computational and algorithmic problems that arise in the dissemination of cross-tabulations (marginal sub-tables) from a single underlying table. These include data structures that exploit sparsity to support efficient computation of marginals and algorithms such as iterative proportional fitting, as well as a generalized form of the shuttle algorithm that computes sharp bounds on (small, confidentiality threatening) cells in the full table from arbitrary sets of released marginals. We give examples illustrating the techniques.

Keywords: branch and bound, contingency tables, disclosure limitation, integer programming, marginal bounds, shuttle algorithm

1. Introduction

Statistical agencies, such as the US Census Bureau and Statistics Canada, disseminate immense amounts of tabular information derived from confidential microdata. Protecting this confidentiality (and thereby the privacy of the data subjects) is mandated by law; doing so while releasing as much useful information as possible presents major challenges.

For the past five years, the National Institute of Statistical Sciences (NISS) has been developing systems for disclosure-limited dissemination of tabular summaries of confidential microdata. Such summaries consist of marginal sub-tables of a large contingency table constructed by "summing out" one or more attributes. The full table of frequency counts of data records with the same (categorical) attribute values is assumed not to be releasable. More important, many sub-tables are also not releasable because either on their own or in conjunction with other sub-tables they provide too much information about the full table. Often, and in this paper, "too much information" means that small count (and therefore high risk (Willenborg and de Waal 2001)) cells can be bounded too accurately on the basis of the released sub-tables.

Two classes of software systems have been developed (Dobra *et al*. 2002, Karr, Dobra and Sanil 2003). Table servers are "live," responding dynamically to incoming user queries for sub-tables of the full table, and assessing disclosure risk in light of previously answered queries. Table servers can be built at realistic scales, but defensible release rules and operating policies that the user community views as equitable are major impediments to their use in practice. Optimal tabular releases (OTRs), by contrast, are static releases of sets of sub-tables constructed by maximizing the amount of information released, as given by a measure of utility of that information, subject to a constraint on disclosure risk. Common underlying abstractions such as the query space and released and unreleasable sub-tables and frontiers are discussed in Dobra *et al*. (2002) and Karr, Dobra and Sanil (2003).

In this paper we describe computational and algorithmic issues that must be confronted in order to build scalable implementations of table servers and OTRs. In the next section we introduce some basic notation that is needed to formally define the bounds problem. In Section 3 we describe the infrastructure—data structures and computational techniques—necessary to represent and operate on large (for example, 40-dimensional) contingency tables in the context of safe releases of sets of sub-tables. The common theme is sparsity: real, large tables are very sparse. Section 4 focuses on one method for computing sharp integer bounds based on any set of released sub-tables. The approach, which is called the *generalized shuttle algorithm*, is based on the underlying hierarchical structure of the categorical

*Now at Duke University, Durham, NC.

0960-3174

data and includes a more general version of the shuttle algorithm (Buzzigoli and Giusti 1999). Section 5 contains a concluding discussion.

2. Terminology and notation

Let $\ell_1 = \{1, \ldots, I_1\}, \ldots, \ell_k = \{1, \ldots, I_k\}$ be finite sets. A *count table* (contingency table) indexed by $\ell_1, \ldots, \ell_k$ is then a function

$$n : \ell_1 \times \cdots \times \ell_k \longrightarrow \mathcal{N},$$

where $\mathcal{N}$ is the set of positive integers. For simplicity, we term $\ell(n) = \ell_1 \times \cdots \times \ell_k$ the index set of n. A *cell* of n is defined by coordinates $i \in \ell(n)$ and its value is $n(i)$ (in accordance with commonly used terminology, the value will usually be referred to as the cell *count*). Let $\mathrm{NC}(n) = I_1 \times \cdots \times I_k$ be the number of cells in n, and let $\mathrm{NC}^+(n)$ be the number of cells in n with non-zero values.

For $A = \{i_1, \ldots, i_l\}$ an arbitrary subset of the indices of n, let n_A be the marginal table obtained by "summing out" indices not in A, which has $\ell(n_A) = \ell_{i_1} \times \cdots \times \ell_{i_l}$. By convention, $n_\emptyset = \sum_{i \in \ell(n)} n(i)$, which we term the *grand total* of N.

Let $n_1, \ldots, n_p$ be tables whose index sets $\ell(n_j)$ are subsets of some overall index set ℓ, and suppose that these tables are consistent in the sense that all common marginals agree. (In particular these tables all have the same grand total.) For example, $n_1, \ldots, n_p$ could be marginals of a table n with $\ell(n) = \ell$. Suppressing dependence on ℓ, we denote by $\mathbf{T}(n_1, \ldots, n_p)$ the set of all tables indexed by ℓ whose appropriate marginals coincide with $n_1, \ldots, n_p$. Note that $\mathbf{T}(n_1, \ldots, n_p)$ can be empty—there may exist no table with the n_j as marginals, their consistency notwithstanding. "Good" algorithms (for example, to compute bounds) should accommodate this possibility. For each cell i, let $U(i; \mathbf{T}(n_1, \ldots, n_p))$ and $L(i; \mathbf{T}(n_1, \ldots, n_p))$ be the maximum and minimum values of cell i over all tables in $\mathbf{T}(n_1, \ldots, n_p)$.

Given $\ell = \ell_1 \times \cdots \times \ell_k$, let $\mathcal{RD}(\ell)$ be the family of all index sets of the form $\mathcal{J}_1 \times \cdots \times \mathcal{J}_k$, where each $\mathcal{J}_m$ is a partition of the corresponding ℓ_m. If n is table with $\ell(n) = \ell$, then for each $\mathcal{J} \in \mathcal{RD}(\ell)$ there is a table $n' = n'(n, \mathcal{J})$ obtained from n by aggregating n not only across variables, but also across categories within variables. To illustrate, suppose that n

$$n = \begin{bmatrix} 1 & 2 \\ 3 & 4 \\ 5 & 6 \end{bmatrix}$$

is a 3×2 table, so that $\ell(n) = \{1, 2, 3\} \times \{1, 2\}$. Then $\mathcal{RD}(\ell(n))$ consists of 16 index sets. The element $\mathcal{J} = \{\{1, 2\}, \{3\}\} \times \{\{1, 2\}\}$ of $\mathcal{RD}(\ell(n))$ corresponds to aggregating categories 1 and 2 of the row attribute and summing out the column attribute, so that

$$n'(n, \mathcal{J}) = \begin{bmatrix} 10 \\ 11 \end{bmatrix}.$$

3. Data structures and algorithms for large, sparse tables

The simplest computer representation of a contingency table is as a multi-dimensional array of nonnegative integers. (Even though it is unwieldy to use multi-dimensional arrays in most programming languages, this structure can readily be implemented as a one-dimensional array by representing the multidimensional coordinates as mixed-radix integers (Knuth 1997)). However, the number of cells in a table increases exponentially with the table dimension, and so even moderate sized tables (10–20 dimensional) can have unmanageably large numbers of cells. For instance, a 14-dimensional table derived from the Current Public Survey (CPS) data from 1993, which we have used as a test case, has 4.5 billion cells! The time and space requirements of processing such tables using a naive representation are clearly prohibitive. Fortunately, tables of real data are extremely sparse—the 14-dimensional table mentioned above with 4.5 billion cells has merely 76,000 cells with non-zero counts.

In this section we describe useful data structures and algorithms that exploit the extreme sparsity of the tables to enable us to perform certain operations on the tables. We first present a hash-table based structure for tables and outline algorithms for building the table from microdata and for generating marginal subtables (Section 3.1) followed by an illustration (Section 3.2) of how the basic structures and minor extensions can be used to perform a variant of Iterative Proportional Fitting (IPF) (Bishop, Fienberg and Holland 1975). We note that AD-Trees (Moore and Lee 1998) and certain Online Analytical Processing (OLAP) technologies (Harinarayan, Rajaraman and Ullman 1996) provide complementary methods for handling tables.

3.1. *Building tables and marginal tables*

It is clear that any viable representation for large, sparse tables—with $\mathrm{NC}(n) \gg \mathrm{NC}^+(n)$—must exploit the table's sparsity. The most natural strategy is to store the location (coordinate) and content (count) of only the non-zero cells. This leads to an enormous saving of space, since we only have to store $\mathrm{NC}^+(n)$ items instead of $\mathrm{NC}(n)$ items. However, the naive approach of storing a list of (coordinate, count) pairs suffers from the disadvantage that retrieving the count of an arbitrary cell entails a search through the list of cells. Hence, this query, which could be processed in $O(1)$ time with the multidimensional array representation, now takes $O(\mathrm{NC}^+(n))$ time (or $O(\log \mathrm{NC}^+(n))$ if the list is sorted). Fortunately, we can achieve the space-saving of a list with the fast access time of an array by using a *hash table*.

Hash tables are well-known data structures that support efficient storage and retrieval for sets of (*key*, *value*) pairs—(*coordinate*, *count*) pairs in our case. In a hash table, the data are stored in an array. The essential component is a *hash function* that maps every key (coordinate) to a location in the array and

allows us almost instantly to access data corresponding to a given key. There are many design issues pertaining to construction of a good hash function, selection of a good array size, and several implementation tricks that could be employed. We do not cover the details here since this is a widely-used and well-studied data structure (Cormen, Leiserson and Rivest 1990, Knuth 1997), and has implementations incorporated as standards in computer languages like Java (part of the "Java collections framework" (Sun Microsystems 2002)) and C++ (part of the standard library (Stroustrup 1997)).

Three essential aspects are relevant to us. First, tables and marginal subtables are stored in hash tables where the *key* is a k-variate generalized coordinate, $(x_i, \ldots, x_k)$ with $x_j \in \ell_j \cup \{0\}$ and the *value* is the corresponding cell count. (We call it a generalized coordinate since it can represent cells in both the full table and any marginal subtable, with the key for a cell in a marginal table n_A being a k-variate coordinate with $x_j = 0$ if $j \notin A$). Second, the hash function is defined over the set of all generalized coordinates.[1]

Third, the following operations are supported efficiently:

1. ADDCOUNT(`Tab, coord`): Increments by one the count for the cell with coordinate `coord` in table or marginal table `Tab`.
2. GETCOUNT(`Tab, coord`): Retrieves the cell count for the cell with coordinate `coord` in table or marginal table `Tab`.
3. GETMARGINALCOORD(`M, coord`): Returns the generalized coordinate in the marginal subtable `M` corresponding to `coord` in the full table (if $\mathtt{M} \equiv n_A$, then this can be computed by setting $x_j = 0$ for $j \notin A$).

We assume the full table fits into the computer's main memory. This is a reasonable assumption in cases such as survey data, where the number of subjects, $n_\emptyset$, is seldom more than an order of 10^5 and since $\mathrm{NC}^+(n) < n_\emptyset$ and, usually, $\mathrm{NC}^+(n) \ll n_\emptyset$. It is clear that the hash table representation of the contingency table can be easily constructed by reading through the microdata records sequentially and incrementally updating the table using ADDCOUNT.

Given the hash table representation, computation of an arbitrary marginal sub-table is easy, and is shown in Pseudocode 3.1.

3.2. *Example: IPF*

We illustrate here how the hash table representation can be used to develop efficient algorithms to process large, sparse tables, using Iterative Proportional Fitting (IPF) as an example. Intuitively, IPF finds the "best" reconstruction of the full table consistent with a set of marginal tables of the original full table. More precisely, IPF calculates the maximum likelihood estimate for a log-linear model defined on variables on the full table whose minimal sufficient statistics are the given by the set of marginal tables (Bishop, Fienberg and Holland 1975).

The algorithm involves iteratively refining estimated cell values for the table. Each iteration consists of stepping through the list of marginal tables and scaling the current cell estimates to make the current table estimate consistent with the marginal table. (To illustrate, for a two-way table, entries are alternately re-scaled row-wise to make the row sums "correct" and then column-wise to make the column sums "correct.") Specifically, consider cell i with $\hat{n}(i)$ as the current estimate of its count. Let $\mathrm{mc}(i)$ be the cell count of the cell in the marginal table currently under consideration to which i contributes to, and let $\widehat{\mathrm{mc}}(i)$ be the sum of all current estimates of cells like i that contribute to the marginal sum. Then $\hat{n}(i)$ is adjusted as $\hat{n}(i) \leftarrow [\mathrm{mc}(i)/\widehat{\mathrm{mc}}(i)]\hat{n}(i)$. After adjusting all cells for a given marginal, the current table estimate will be consistent with the marginal under consideration (Bishop, Fienberg and Holland 1975).

The IPF variant we present here only considers non-zero cells—this is equivalent to the original IPF with structural or known zeroes. Implementing it involves a minor extension of the basic data structure. The (*key*, *value*) pair in the hash table now consists of *key* = coordinate as before and *value* = (count,fit) = $(n, \hat{n})$, where $n \equiv n(i)$, $\hat{n} \equiv \hat{n}(i)$ for the table, and $n \equiv \mathrm{mc}(i)$, $\hat{n} \equiv \widehat{\mathrm{mc}}(i)$ for the marginal table. We define procedures ADDFIT(`Tab, coord`) and GETFIT analogous to ADDCOUNT and GETCOUNT in Section 3.1. We also define a procedure UPDATEFIT(`Tab, coord, fit`) that sets the "fit" component to the value of `fit`.

The IPF employs two subroutines MAKEMARGINAL SUMS(`Table,Marginal`) (Pseudocode 3.2) and ADJUSTTABLE (`Table,Marginal`) (Pseudocode 3.3) that compute the $\widehat{\mathrm{mc}}(i)$s and the $\hat{n}(i)$s respectively.

The IPF algorithm can be implemented as outlined in Pseudocode 3.4. This version trades off time efficiency for memory space savings by recomputing marginal tables every time they are required. Since this strategy maintains only two tables in memory at any given time, it allows us to handle arbitrarily large sets of marginals. It is also possible to precompute the marginal tables and avoid the COMPUTEMARGINAL in the inner loop if both the set of marginal tables and the full table can fit into main memory.

Pseudocode 3.1 COMPUTEMARGINAL(`Table,Marginal`): Procedure for computing marginal subtable from the full table.

```
for each coord in Table do
    marginalCoord = GETMARGINALCOORD(Marginal,coord)
    ADDCOUNT(Marginal, marginalCoord)
end for
```

Pseudocode 3.2 MAKEMARGINALSUMS(`Table,Marginal`): Procedure for computing the $\widehat{\mathrm{mc}}(i)$s.

```
for each coord in Table do
    marginalCoord = GETMARGINALCOORD(MarginalTable,coord)
    ADDFIT(Marginal, marginalCoord)
end for
```

Pseudocode 3.3 ADJUSTTABLE(Table,Marginal): Procedure for computing the $\hat{n}(i)$s

```
for each coord in Table do
  marginalCoord = GETMARGINALCOORD(Marginal,coord)
  marginalFit = GETFIT(Marginal,marginalCoord)
  tableFit = GETFIT(Table,coord)
  marginalCount = GETCOUNT(Marginal,marginalCoord)
  E tableFit ← tableFit * (marginalCount/marginalFit)
  UPDATEFIT(Table,coord)
end for
```

Pseudocode 3.4 IPF(Table,ListOfMarginalNames): Main IPF procedure

```
repeat
  for each Marginal in ListOfMarginalNames do
  COMPUTEMARGINAL(Table,Marginal)
  do MAKEMARGINALSUMS(Table,Marginal)
  do ADJUSTTABLE(Table,Marginal)
  end for
until Estimates converge
```

4. The generalized shuttle algorithm

The fundamental idea behind the "shuttle" algorithm is that the upper and lower bounds for the cells in a table **T**, based on knowledge of an arbitrary set of marginal sub-tables, are interlinked. The method builds on (Buzzigoli and Giusti 1999), which for the case of an k-way table with known $(k-1)$-way marginals, sequentially improves the bounds for cells of interest until no further adjustment can be made. The generalized shuttle algorithm was introduced briefly in Dobra and Fienberg (2001) and presented in detail in Dobra (2002).

Let $\mathcal{C}$ denote the set of cells in tables indexed by some element of $\mathcal{RD}(\ell)$. If the set of cell entries that define a "super-cell" $t_1 \in \mathcal{C}$ is included in the set of cells defining another "super-cell" t_2, we write $t_1 \prec t_2$. We can then define a partial ordering "$\prec$" on the cells in **T** by $t_{J_1^1 \ldots J_k^1} \prec t_{J_1^2 \ldots J_k^2} \Leftrightarrow J_1^1 \subseteq J_1^2, \ldots, J_k^1 \subseteq J_k^2$. With this partial ordering, $(\mathcal{C}, \prec)$ has a maximal element, namely the grand total element, and many minimal elements—the cells in tables indexed by ℓ.

Suppose now that $t_1 = t_{J_1^1 \ldots J_k^1}$ and $t_2 = t_{J_1^2 \ldots J_k^2}$ are such that $t_1 \prec t_2$, and further that $J_r^1 = J_r^2$, for $r = 1, \ldots, r_0 - 1, r_0 + 1, \ldots, k$ and $J_{r_0}^1 \neq J_{r_0}^2$. Then we define the *complement* of t_1 with respect to t_2 to be the cell $t_3 = t_{J_1^3 \ldots J_k^3}$, where J_r^3 is set equal to J_r^1 if $r \neq r_0$ and equal $J_r^2 \setminus J_r^1$ otherwise. In this case we write $t_1 \oplus t_3 = t_2$. The operator "$\oplus$" is equivalent to joining two blocks of cells in **T** to form a third block. The blocks to be joined have to be composed from the same categories in $(k-1)$ dimensions and cannot share any categories in the remaining dimension.

Example 1. In Fig. 1 we give the set **T** corresponding to a two-dimensional table having two rows and three columns. The number of cells in **T** is $21 = (2^2 - 1) \cdot (2^3 - 1)$. The cell at the top of the hierarchy is the grand total. This maximal cell can be decomposed into "elementary" cells, i.e., cells contained in the initial table n, by sequentially going through pairs of cells in **T** that are complements of each other. The continuous and the dotted arrows show two different ways of decomposing $n_\emptyset$. The arrows point from bigger elements to smaller elements with respect to the ordering "$\prec$". If we follow the continuous arrows, we come across the following cells:

$$t_{\{1,2\}\{1,2,3\}} = n_\emptyset, t_{\{1,2\}\{1,2\}} = n_{11} + n_{12} + n_{21} + n_{22}, t_{\{1\}\{1,2\}} = n_{11} + n_{12},$$
$$t_{\{2\}\{1,2\}} = n_{21} + n_{22}, t_{\{1,2\}\{3\}} = n_{13} + n_{23}, t_{\{1\}\{2\}} = n_{12}, t_{\{1\}\{1\}} = n_{11},$$
$$t_{\{1\}\{3\}} = n_{13}, t_{\{2\}\{1\}} = n_{21}, t_{\{2\}\{2\}} = n_{22}, t_{\{2\}\{3\}} = n_{23}.$$

The dependencies existent among the above cells are fully characterized in terms of the hierarchical structure induced by the

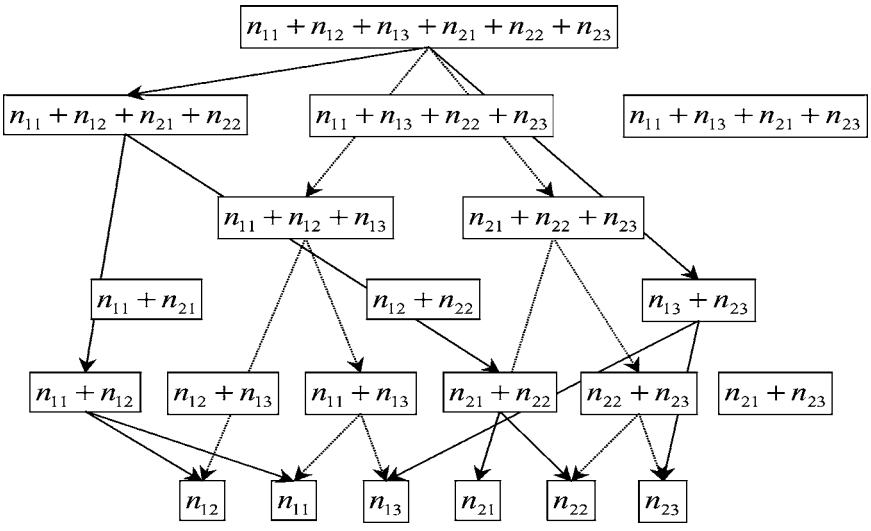

Fig. 1. *Hierarchy of cells associated with a* 2×3 *table* $n = \{n_{ij} : 1 \leq i \leq 2, 1 \leq j \leq 3\}$

ordering "$\prec$":

$$t_{\{1,2\}\{1,2\}} \oplus t_{\{1,2\}\{3\}} = t_{\{1,2\}\{1,2,3\}},\ t_{\{1\}\{1,2\}} \oplus t_{\{2\}\{1,2\}} = t_{\{1,2\}\{1,2\}},$$
$$t_{\{2\}\{1\}} \oplus t_{\{2\}\{2\}} = t_{\{2\}\{1,2\}},\ t_{\{1\}\{3\}} \oplus t_{\{2\}\{3\}} = t_{\{1,2\}\{3\}}.$$

Denote by $L(t)$ and $U(t)$ the current lower and upper bounds for the "super-cell" $t \in \mathbf{T}$. Let $L(\mathbf{T}) = \{L(t) : t \in \mathbf{T}\}$ and $U(\mathbf{T}) = \{U(t) : t \in \mathbf{T}\}$. If $\mathcal{N} \subset \mathbf{T}$ is the set of cells in $\mathbf{x} \in \mathbf{T}(n^1, \ldots, n^p)$, then $L(\mathcal{N})$ and $U(\mathcal{N})$ are the bounds arrays to be determined. Every $t \in \mathbf{T}$ has a value $V(t)$ assigned to it. If t corresponds to an entry in a fixed marginal, we "know" the value $V(t)$ of that entry, so we set both current bounds of t to $V(t)$.

Let $\mathbf{T}_0$ be the set of cells in $\mathbf{T}$ for which the lower bound is currently equal to the upper bound. If we fix all the cells in $\mathcal{N}$ at a certain value, then all the remaining cells in $\mathbf{T}$ will also be fixed: $\mathcal{N} \subset \mathbf{T}_0$ implies $\mathbf{T} = \mathbf{T}_0$. Consider $\mathbf{M} \subset \mathbf{T}$ to be the set of cells in the fixed marginals $n_1, \ldots, n_p$. When the iterative procedure described below starts, $\mathbf{T}_0$ will contain only the cells in the fixed marginals, i.e., $\mathbf{T}_0 = \mathbf{M}$. For the remaining cells in $\mathbf{T}$, we set $L(t) = 0$ and $U(t) = \mathbf{n}_\emptyset$. Denote by $L_0(\mathbf{T})$ and $U_0(\mathbf{T})$ this initial set of upper and lower bounds induced by $n_1, \ldots, n_p$.

As the algorithm progresses, the current bounds $L(\mathbf{T})$ and $U(\mathbf{T})$ are improved (lower bounds increase and upper bounds decrease), and more and more cells are added to $\mathbf{T}_0$. When the bounds associated with t become equal, t is added to $\mathbf{T}_0$, and is assigned value $V(t) = L(t) = U(t)$. We state the bounds problem in a new equivalent form: *Find sharp integer bounds for the cells in* $\mathbf{T}$ *if the values of some cells* $\mathbf{T}_0 \subset \mathbf{T}$ *are fixed.*

Let $\mathcal{Q} = \mathcal{Q}(\mathbf{T})$ denote the triplets of cells $\mathcal{Q}(\mathbf{T}) = \{(t_1, t_2, t_3) \in \mathbf{T} \times \mathbf{T} \times \mathbf{T} : t_1 \oplus t_3 = t_2\}$ that represent the cell dependencies to be satisfied. Let $S[L_0(\mathbf{T}), U_0(\mathbf{T})]$ be the set of integer tables consistent with $L_0(\mathbf{T})$ and $U_0(\mathbf{T})$. It is easy to see that

$$\{V(\mathcal{N}) : V(\mathbf{T}) \in S[L_0(\mathbf{T}), U_0(\mathbf{T})]\} = \mathbf{T}(n_1, \ldots, n_p).$$

To improve the current bounds, we go sequentially through all dependencies in $\mathcal{Q}$ and update upper and lower bounds in the following way. Consider a triplet $(t_1, t_2, t_3) \in \mathcal{Q}$ with $t_1 \prec t_2$ and $t_3 \prec t_2$. If $t_1, t_2, t_3 \in \mathbf{T}_0$, we check whether we came across an inconsistency. The procedure stops if $V(t_1) + V(t_3) \neq V(t_2)$. Assume that $t_1, t_3 \in \mathbf{T}_0$, and $t_2 \notin \mathbf{T}_0$. Then t_2 can only take one value, namely $V(t_1) + V(t_3)$. If $V(t_1) + V(t_3) \notin [L(t_2), U(t_2)]$, we have encountered an inconsistency and stop. Otherwise we set $V(t_2) = L(t_2) = U(t_2) = V(t_1) + V(t_3)$, and include t_2 in the set $\mathbf{T}_0$ of cells having a fixed value. Similarly, if $t_1, t_2 \in \mathbf{T}_0$ and $t_3 \notin \mathbf{T}_0$, t_3 can only be equal to $V(t_2) - V(t_1)$. If $V(t_2) - V(t_1) \notin [L(t_3), U(t_3)]$, we again discovered an inconsistency. Otherwise, we set $V(t_3) = L(t_3) = U(t_3) = V(t_2) - V(t_1)$ and $\mathbf{T}_0 = \mathbf{T}_0 \cup \{t_3\}$. In the case that $t_2, t_3 \in \mathbf{T}_0$ and $t_1 \notin \mathbf{T}_0$ is handled analogously.

Now we examine the situation when at least two of the cells t_1, t_2, t_3 do not have a fixed value. For each cell not having a fixed value, we update its upper and lower bounds so that the new bounds satisfy the dependency $t_1 \oplus t_3 = t_2$. Suppose that $t_1 \notin \mathbf{T}_0$. If $U(t_2) - L(t_3) < L(t_1)$ or if $L(t_2) - U(t_3) > U(t_1)$, an inconsistency is detected and the procedure stops. Otherwise, the updated bounds for t_1 will be $U(t_1) = \min\{U(t_1), U(t_2) - L(t_3)\}$ and $L(t_1) = \max\{L(t_1), L(t_2) - U(t_3)\}$. If $t_3 \notin \mathbf{T}_0$, we update $L(t_3)$ and $U(t_3)$ in the same way. Finally, assume that $t_2 \notin \mathbf{T}_0$. If $U(t_1) + U(t_3) < L(t_2)$ or if $L(t_1) + L(t_3) > U(t_2)$, we stop the algorithm. Otherwise, we set $U(t_2) = \min\{U(t_2), U(t_1) + U(t_3)\}$ and $L(t_2) = \max\{L(t_2), L(t_1) + L(t_3)\}$. After updating the bounds of $t \in \mathbf{T}$, we check whether the new upper bound is equal to the new lower bound. If so, we add t to $\mathbf{T}_0$ and set $V(t) = L(t) = U(t)$.

We continue cycling through dependencies in $\mathcal{Q}$ until the upper bounds no longer decrease, the lower bounds no longer increase and no new cells are added to $\mathbf{T}_0$. The algorithm terminates in a finite number of steps: either an inconsistency is detected or bounds cannot be improved.

If an inconsistency is detected, $S[L_0(\mathbf{T}), U_0(\mathbf{T})]$ is empty and no bounds are generated. However, $S[L_0(\mathbf{T}), U_0(\mathbf{T})]$ could still be empty even if the shuttle procedure did not come across any inconsistencies and has converged to bounds $L_s(\mathbf{T})$ and $U_s(\mathbf{T})$. These two bounds arrays define the same feasible set of tables as the arrays $L_0(\mathbf{T})$ and $U_0(\mathbf{T})$ we started with, namely $S[L_s(\mathbf{T}), U_s(\mathbf{T})] = S[L_0(\mathbf{T}), U_0(\mathbf{T})]$. The fact that, for any $t \in \mathbf{T}$, we have $L_0(t) \leq L_s(t) \leq U_s(t) \leq U_0(t)$, could make one think that $S[L_s(\mathbf{T}), U_s(\mathbf{T})]$ might be strictly included in $S[L_0(\mathbf{T}), U_0(\mathbf{T})]$. Nevertheless, the dependencies in $\mathcal{Q}(\mathbf{T})$ imply the equality of the two sets of feasible integer tables (Dobra 2002).

4.1. *Convergence properties*

The bounds produced by the generalized shuttle algorithm (provided it did not stop due to an inconsistency) are *valid* in the sense that the sharp bounds (over the set of all tables consistent with the prescribed marginals) for each cell $t_0 \in \mathbf{T}$ lie inside the interval defined by the shuttle bounds for t_0. However, there are two cases when the shuttle bounds are sharp: dichotomous k-dimensional tables with all $(k-1)$-dimensional marginals fixed, and when fixed the marginals are the minimal sufficient statistics of a decomposable log-linear model. In both instances, explicit formulas for the bounds exist, and show next that generalized shuttle algorithm yields the same result.

4.2. *Dichotomous k-way tables with fixed $(k-1)$-way marginals*

Consider a k-way table n for which $\ell_1 = \cdots = \ell_k = \{1, 2\}$. Collapsing n across categories is equivalent to collapsing across variables, thus the set $\mathbf{T}$ associated with the dichotomous table n is the set of cells in every marginal of n. Assume that the $(k-1)$-dimensional marginals of n are fixed. Of course, every lower-dimensional marginal of n is also known. The only cells in $\mathbf{T}$ that are unknown are those in the original table.

The $(k-1)$-dimensional marginals of n are the minimal sufficient statistics of the log-linear model of no $(k-1)$-order

interaction. This log-linear model has only one degree of freedom because n is dichotomous (Fienberg 1999). Consequently, we can uniquely express the count in any cell as a function of one single fixed cell alone—only one more quantity is needed in order to determine the entries for the full table.

Let $c = n(1, \ldots, 1)$. In Proposition 1 (Dobra 2002) we give an explicit formula for computing the count in an arbitrary cell based on c and on the set of fixed marginals.

Proposition 1. *Consider an index $i^0 \in \ell$. Let $\{q_1, \ldots, q_l\} \subset K$ be such that, for $r \in K$,*

$$i_r^0 = \begin{cases} 1, & \text{if} \quad r \in K \setminus \{q_1, \ldots, q_l\}, \\ 2, & \text{if} \quad r \in \{q_1, \ldots, q_l\}. \end{cases} \tag{1}$$

For $s = 1, \ldots, l$, let $C_s = K \setminus \{q_s\}$. Then

$$n(i^0) = (-1)^l \cdot c - \sum_{s=0}^{l-1} (-1)^{l+s} \cdot n_{C_{(l-s)}}\bigl(1, \ldots, 1, i^0_{q_{(l-s)}+1}, \ldots, i^0_k\bigr). \tag{2}$$

Let n_C be the sub-table corresponding to the index set C. (Typically, C is a clique in the graph.)

The upper and lower bounds can therefore be obtained by imposing the non-negativity constraints $n(i^0) \geq 0$, $i^0 \in \ell$, in these relations. For example, the sharp lower bound for $n(1, \ldots, 1)$ is

$$\max\left\{ \sum_{s=0}^{l-1} (-1)^s \cdot n_{C_{(l-s)}}\bigl(1, \ldots, 1, i^0_{q_{(l-s)}+1}, \ldots, i^0_k\bigr) : l \text{ even} \right\}, \tag{3}$$

where i^0 is as in (1), and the corresponding upper bound is given by

$$\min\left\{ \sum_{s=0}^{l-1} (-1)^s \cdot n_{C_{(l-s)}}\bigl(1, \ldots, 1, i^0_{q_{(l-s)}+1}, \ldots, i^0_k\bigr) : l \text{ odd} \right\}. \tag{4}$$

The generalized shuttle algorithm converges to the bounds in (3) and (4) (Dobra 2002). Moreover, one can obtain *all* feasible tables consistent with the $(k-1)$-dimensional marginals of n by replacing every possible value c that the cell $(1, 1, \ldots, 1)$ can take in (2) applied for all cells i^0. In particular, all the cells in n can take the same number of values—the difference between the upper and lower bounds is constant for all cells, and is equal with the number of feasible integer tables consistent with the $(k-1)$-dimensional marginals of n.

4.3. *The decomposable case*

Log-linear models are a common way of representing and studying contingency tables with fixed marginals. In particular, a *graphical* log-linear model corresponds to conditional independence relationships that can be summarized by means of an independence graph (Madigan and York 1995). Decomposable log-linear models (Lauritzen 1996) are a sub-class of graphical models with closed form structure and special properties that lead to explicit formulas for computing bounds (Dobra and Fienberg 2000). The complete proof of the next theorem appears in Dobra (2002).

Theorem 1. *Let $C_1, \ldots, C_p$ be index sets corresponding to the minimal sufficient statistics of a decomposable log-linear model, let $n_j = n_{C_j}$ be consistent tables indexed by the C_j, and let $S_2, \ldots, S_p$ be the associated set of separators in the decomposable independence graph with cliques $C_1, \ldots, C_p$. Then:*

(i) *$\mathbf{T}(n_1, \ldots, n_p)$ contains at least one table.*

(ii) *The sharp upper bound for cell i given $n_1, \ldots, n_p$ is*

$$U(i; \mathbf{T}(n_1, \ldots, n_p)) = \min\bigl\{ n_j\bigl(i_{C_j}\bigr) : j = 1, \ldots, p \bigr\}.$$

(iii) *The sharp lower bound for cell i given $n_1, \ldots, n_p$ is*

$$L(i; \mathbf{T}(n_1, \ldots, n_p)) = \max\left\{ \sum_{j=1}^{p} n_j\bigl(i_{C_j}\bigr) - \sum_{j=2}^{p} (n_j)_{S_j}\bigl(i_{S_j}\bigr), 0 \right\}.$$

Therefore, when the fixed marginals define a decomposable graphical model, pairwise consistency of marginals implies the existence of a feasible table.

4.4. *Finding a feasible integer table*

As noted above, the generalized shuttle algorithm can converge to bounds $L_s(\mathbf{T})$ and $U_s(\mathbf{T})$ even if there does not exist an integer table consistent with the fixed marginals $n_1, \ldots, n_p$. Since bounds make no sense when no table exists, we augment our procedure by proposing a method that will actually determine a feasible table in $\mathbf{T}(n_1, \ldots, n_p)$, provided one exists.

This is done by sequentially choosing possible values for cells in $\mathcal{N}$. Once a new cell has been fixed, bounds for all the cells in $\mathbf{T}$ are updated using the shuttle algorithm. If the shuttle procedure did not stop because of an inconsistency, we pick a value for another cell. Otherwise, we choose a new value for the cell fixed at the current step. If all the cells in $\mathcal{N}$ have been fixed and the shuttle procedure completed successfully, which indicates that the dependencies in $\mathcal{Q}(\mathbf{T})$ are satisfied, we have determined a table in $\mathbf{T}(n_1, \ldots, n_p)$. We denote by $T_0^{(0)} = T_0$ the set of cells fixed by the shuttle procedure when computing the bounds $L_s(\mathbf{T})$ and $U_s(\mathbf{T})$. Also, initialize the bounds arrays $L^{(0)}(\mathbf{T}) = L_s(\mathbf{T})$ and $U^{(0)}(\mathbf{T}) = U_s(\mathbf{T})$. Here is the procedure in pseudo-code:

Step 1. Set $l = 1$.

Step 2. Check whether $\mathcal{N}^l = \mathcal{N} \cup (\mathbf{T} \setminus \mathbf{T}_0^{(l-1)})$ is empty. If so, a feasible table has been determined; stop the algorithm.

Step 3. Select a cell $t^l \in \mathcal{N}^l$.

Step 4. **FOR** every integer $v_l \in [L^{(l-1)}(t_l), U^{(l-1)}(t_l)]$ **DO**

- Initialize new bound arrays $L^{(l)}(\mathbf{T}) = L^{(l-1)}(\mathbf{T})$ and $U^{(l)}(\mathbf{T}) = U^{(l-1)}(\mathbf{T})$.
- Set $V(t_l) = L^{(l)}(t_l) = U^{(l)}(t_l) = v_l$ and put $\mathbf{T}^{(l)} = \mathbf{T}^{(l-1)} \cup \{t_l\}$.
- Run the generalized shuttle algorithm to update $L^{(l)}(\mathbf{T})$, $U^{(l)}(\mathbf{T})$ and $\mathbf{T}^{(l)}$.

Table 1. *A 2 × 2 × 2 × 2 table (left-hand panel) and the real (not necessarily integer) bounds computed using linear programming (right-hand panel)*

		C	no		yes		C	no		yes	
A	B	D	no	yes	no	yes	D	no	yes	no	yes
no	no		1	0	0	1		[0, 1]	[0, 0.67]	[0, 1.67]	[0, 1]
	yes		0	0	1	0		[0, 0.67]	[0, 0.67]	[0, 1]	[0, 0.67]
yes	no		0	0	1	0		[0, 0.67]	[0, 0.67]	[0, 1]	[0, 0.67]
	yes		0	1	0	0		[0, 0.67]	[0, 1]	[0, 0.67]	[0, 0.67]

- If the generalized shuttle algorithm did not stop because of an inconsistency, set $l = l + 1$ and go to Step 2.

END FOR.

Step 5. The algorithm stops; there does not exist a feasible integer table in $\mathbf{T}(n_1, \ldots, n_p)$.

Instead of "blindly" searching the entire space of possible combinations of values for the cells in $\mathcal{N}$ defined by the bounds arrays $L_s(\mathcal{N})$ and $U_s(\mathcal{N})$, the procedure reduces the search space continually as the algorithm progresses because the bounds are updated at each step. Several heuristics exist for substantially increasing the speed of this search procedure (Dobra 2002). Moreover, if we do not stop the algorithm at Step 2 once the first feasible table is generated, then *all* the integer tables in $\mathbf{T}(n_1, \ldots, n_p)$ will be identified.

4.5. *Calculating sharp bounds*

We can now obtain sharp bounds for any cell $t_0 \in \mathcal{T}$. Let $L_s(\mathbf{T})$ and $U_s(\mathbf{T})$ be the bounds produced by the generalized shuttle algorithm. Then

$$L_s(t_0) \leq L_{t_0} \leq U_{t_0} \leq U_s(t_0).$$

We need to "adjust" $L_s(t_0)$ and $U_s(t_0)$ to the sharp bounds L_{t_0} and U_{t_0} by making use of the simple fact that, for any integer $v \in [L_s(t_0), L_{t_0}) \cup (U_{t_0}, U_s(t_0)]$, there does not exist an integer array $V(\mathbf{T}) \in S[L_s(\mathbf{T}), U_s(\mathbf{T})]$ such that $V(t_0) = v$. To determine L_{t_0}, we start with $v = L_s(t_0)$ and attempt to determine an array $V(\mathbf{T}) \in S[L_s(\mathbf{T}), U_s(\mathbf{T})]$ with $V(t_0) = v$. If such an array exists, $L_{t_0} = v$. Otherwise, we do the same thing for $v = L_s(t_0)+1$, and so on. Here is the complete algorithm in pseudo-code. (Again, $\mathbf{T}_0$ represents the cells t with $L_s(t) = U_s(t)$.)

FOR every integer value $v_l \in [L_s(t_0), L_s(t_0) + 1, \ldots, U_s(t_0)]$ **DO**

- Initialize new bound arrays $L^v(\mathbf{T}) = L_s(\mathbf{T})$ and $U^v(\mathbf{T}) = U_s(\mathbf{T})$.
- Set $V(t_0) = L^v(t_0) = U^v(t_0) = v$ and set $\mathbf{T}^v = \mathbf{T}_0 \cup \{t_0\}$.
- Run the generalized shuttle algorithm to update $L^v(t_0)$, $U^v(t_0)$ and $\mathbf{T}^v$.
- Using the procedure described in the previous section, find out whether $S[L^v(t_0), U^v(t_0)]$ is empty.
- If $S[L^v(t_0), U^v(t_0)]$ is not empty, set $L_{t_0} = v$ and stop the algorithm.

END FOR

Determination of U_{t_0} is similar: start with $v = U_s(t_0)$ and attempt to determine an array $V(\mathbf{T}) \in S[L_s(\mathbf{T}), U_s(\mathbf{T})]$ with $V(t_0) = v$. If such an array exists, U_{t_0} is equal to v; otherwise, we set $v = U_s(t_0) - 1$ and continue. In this way, we obtain sharp bounds not only for the cells in $\mathcal{N}$ but also for any cell in a cross-classification obtained by collapsing **n** across categories.

Example 2. The left-hand panel of Table 1 contains a 2 × 2 × 2 × 2 contingency table. The generalized shuttle algorithm was used to compute the upper and lower bounds induced by fixing the 6 two-way marginals of this table. There is only one table consistent with this set of two-way marginals, so for all the cells in this table, the sharp integer upper and lower bounds are equal to the actual cell entries. Of course, only rarely is it known that there is only one such table.

The right panel in Table 1 contains comparable bounds compute using linear programming (specifically, the simplex algorithm), which is commonly applied to calculate bounds on table entries. For all cells, one of the bounds is different from the corresponding sharp integer bound. In some cases, the simplex algorithm produced fractional bounds. In other cases, although simplex converged to integer bounds, these bounds are not sharp. Especially, cell (1, 1, 2, 1), marked with a box in Table 1, contains a count of 0, while the actual upper bound is 1.67. Therefore the distance between the integer and the real bounds can be strictly greater than 1.

Adjusting the bounds can be done *in parallel* for all the cells in the table (Dobra 2002). Other methods for reducing the computational effort required by the generalized shuttle algorithm in the case of large sparse multi-way tables, as well as an example of calculating bounds in parallel for a 2^{16}, table are presented in Dobra (2002).

5. Conclusions

In this paper we have outlined how to deal with the challenging issues associated with storing and manipulating large, sparse

tables in the context of statistical disclosure limitation. Dissemination systems and software implementing these techniques are described in Dobra *et al*. (2002) and Karr, Dobra and Sanil (2003). To the extent that they exploit sparsity, these techniques exhibit strong scalability properties. For instance, the fundamental data structures and methods for calculating marginal subtables seem able to handle almost any table that is likely to arise in practice. By contrast, the generalized shuttle algorithm of Section 4, in its current form, entails computations for every cell in the underlying table, and consequently does not scale well. Deriving a scalable version is just one of many research challenges that remain.

The generalized shuttle algorithm can compute the bounds induced by an arbitrary set of marginal totals. This procedure performs well if the set of tables consistent with the fixed marginals is relatively small. Moreover, one can modify the generalized shuttle algorithm to obtain a procedure for enumerating *all* tables consistent with the prescribed marginals, and thereby explore the space induced by any configuration of fixed marginals. It can also be shown that the generalized shuttle algorithm can be used to construct a controlled rounding of a table of counts having a set of fixed marginals (Dobra 2002); a heuristic algorithm for minimizing the loss of information is discussed in Dobra (2002). Finally, the shuttle algorithm computes bounds not only for the cells in the original table, but also for all the cells in cross-classifications derived by table re-design. Consequently, it is possible to make use of the shuttle algorithm to re-design a table in an optimal way according to some data utility criteria.

The generalized shuttle algorithm can also be employed as an alternative to existing algorithms for suppressing cell entries in tables of arbitrary dimensions, which we plan to explore in the near future. Another possible application of the generalized shuttle algorithm is tables containing missing counts and/or structural zeroes. Because we have identified the complete hierarchical configuration of a multi-dimensional array represented by a table of counts and because the shuttle procedure is intimately linked with this special data structure, the methods described in Section 4 perform equally well without further adjustments even when there is not information on some cell counts.

Acknowledgments

Support for the research was provided by National Science Foundation grant EIA-9876619 to NISS. The authors thank Stephen E. Fienberg and Stephen F. Roehrig for suggestions, comments and useful discussions. The example in Table 1 was suggested by Bernd Sturmfels.

Note

1. A generalized coordinate is essentially a length-k array of integers. We refer the reader to (Cormen, Leiserson and Rivest 1990) or any textbook on data structures for examples of hash functions for array-valued keys, since a discussion of even the simplest hash functions involves technical notions such as *collisions* and *chaining*.

References

Bishop Y.M.M., Fienberg S.E., and Holland P.W. 1975. Discrete Multivariate Analyses: Theory and Practice. MIT Press, Cambridge, MA.

Buzzigoli L. and Giusti A. 1999. An algorithm to calculate the lower and upper bounds of the elements of an array given its marginals. In: Statistical Data Protection (SDP'98) Proceedings, Luxembourg, Eurostat. pp. 131–147.

Cormen T.H., Leiserson C.E., and Rivest R.L. 1990. Introduction to Algorithms. MIT Press/McGraw-Hill.

Dobra A. 2002. Statistical Tools for Disclosure Limitation in Multiway Contingency Tables. PhD thesis, Department of Statistics, Carnegie Mellon University.

Dobra A. and Fienberg S. E. 2000. Bounds for cell entries in contingency tables given marginal totals and decomposable graphs. Proc. Nat. Acad. Sciences 97: 11885–11892.

Dobra A. and Fienberg S.E. 2001. Bounds for cell entries in contingency tables induced by fixed marginal totals with applications to disclosure limitation. Statist. J. United Nations ECE 18:363–371, 2001. Presented at the 2nd Joint ECE/Eurostat Work Session on Statistical Data Confidentiality, 14–16 March, Skopje, Macedonia.

Dobra A., Karr A.F., Fienberg S.E., and Sanil A.P. 2002. Software systems for tabular data releases. Int. J. Uncertainty, Fuzziness and Knowledge Based Systems 10(5): 529–544.

Fienberg S.E. 1999. Fréchet and bonferroni bounds for multi-way tables of counts with applications to dis-closure limitation. In: Statistical Data Protection (SDP'98) Proceedings, Luxembourg, Eurostat. pp. 115–129.

Harinarayan V., Rajaraman A., and Ullman, J.D. 1996. Implementing data cubes efficiently. In: Proceedings of the ACM SIGMOD International Conference on Management of Data, Vol. 25(2) of ACM SIGMOD Record, June 4–6, ACM Press, New York, pp. 205–216.

Karr A.F., Dobra A., and Sanil A.P. 2003. Table servers protect confidentiality in tabular data releases. Comm. ACM 46(1): 57–58.

Knuth D.E. 1997. Fundamental Algorithms, Vol. 1 of The Art of Computer Programming, 3rd edn. Addison-Wesley, Reading, Massachusetts.

Lauritzen S.L. 1996. Graphical Models. Clarendon Press, Oxford, UK.

Madigan D. and York J. 1995. Bayesian graphical models for discrete data. Internat. Statis. Rev. 63: 215–232.

Moore A.W. and Lee M.S. 1998. Cached sufficient statistics for efficient machine learning with large datasets. J. Artificial Intell. Res. 8: 67–91.

Bjarne Stroustrup. 1997. The C++ Programming Language, 3rd edn. Addison-Wesley, Reading, MA.

Sun Microsystems. Java programming language. http://java.sun.com, 2002.

Willenborg L.C.R.J. and de Waal T. 2001. Elements of Statistical Disclosure Control. Springer-Verlag, New York.