Computers & Operations Research 52 (2014) 113–122

Contents lists available at ScienceDirect

Computers & Operations Research

journal homepage: www.elsevier.com/locate/caor

Rapid calculation of exact cell bounds for contingency tables from conditional frequencies

Stephen E. Wright *, Byran J. Smucker

Department of Statistics, Miami University, Oxford, OH 45056, United States

ARTICLE INFO

Available online 11 July 2014

Keywords:
Statistical disclosure control
Tabular data
Knapsack problems
Dynamic programming

ABSTRACT

We present a fast method for determining the tightest possible bounds, as well as all feasible values, for the underlying cell counts in a two-way contingency table based on knowledge of the corresponding unrounded conditional probabilities, the sample size, and (optionally) bounds on cells and certain sums of cells. This information can be used in statistical inference procedures and also has potential uses in statistical disclosure control, which deals with protecting privacy and confidentiality when data summaries are released to the public. The problem formally consists of a large number of integer linear knapsack optimizations (two per cell). Here we identify special common structure that allows for efficient reuse among cells of intermediate results within a dynamic programming framework. The method runs very quickly on practical examples, thereby enabling a real-time interactive exploration of disclosure risk for two-way rearrangements of large multi-way tables.

1. Introduction

Statistical disclosure control deals with the compromise between data utility and data privacy. Many data providers, governmental or otherwise, make privacy guarantees to those from whom they collect data. A common form of data release is the contingency table, which provides a summary that supplies information to policy makers, researchers, and the public. Small cell counts in such tables, standing alone or combined with other information, potentially expose private information about individuals in the table [22]. By suitably aggregating information, disclosure risk can be reduced by avoiding the revelation of small counts. On the other hand, overly aggregated summaries may be less useful to the public, decision-makers, and researchers who perform statistical inference on the released information; indeed, this last issue also arises even when there are few concerns about privacy. Key to understanding either side of the matter is the identification of the tightest cell bounds possible that can be determined from a given summary. To be useful, such identification should be accomplished in a reasonable amount of time so that table redesign can be explored. This paper proposes an efficient algorithm for solving the cell-bounding problem in the context of releasing a table of conditional probabilities.

A *two-way contingency table* is a rectangular array of whole numbers. The *marginal* counts are the row and column sums of such a table, and its *conditional row probabilities* are given by dividing each entry in the table by the marginal sum for its row. For example, the two-way contingency tables shown in Figs. 1 (a) and (b) both have the same *sample size* (sum of all entries) and yield the same table of conditional row probabilities, which is shown in Fig. 1(c). The central question considered in this paper is: What information about a contingency table can be recovered from knowledge of its row conditionals and sample size? More specifically, what are the tightest bounds for each cell's value based on such knowledge? For example, we demonstrate in Section 3.1 that Figs. 1(a) and (b) are the only contingency tables of sample size 48 having the conditionals shown in Fig. 1(c). Notice that if we knew a suitable bound on one of the row sums or even on a single cell, then the actual contingency table would be revealed. Therefore we briefly indicate in Section 2 how to incorporate such *a priori* information, which might be available on the basis of other released summaries (or insider knowledge) of the true underlying table.

The purpose of the paper is to solve these problems rapidly, even when the tables consist of hundreds or thousands of cells. The motivation for such speed arises at a couple of levels. First, data stewards need the ability to test several rearrangements of a multi-way table for both privacy and statistical utility (the ability to perform inference). Utility is also a concern even when privacy is not, because the released version of a table may be chosen more for convenience or to highlight some aspects of the data over others. Second, there is a need to determine how much information can be obtained by combining two or more released versions of a table. This paper focuses mainly on the first of these, but the same calculations are relevant to the second because they can be

* Corresponding author.
E-mail address: wrightse@miamioh.edu (S.E. Wright).

http://dx.doi.org/10.1016/j.cor.2014.07.002
0305-0548/

a

3	4	7
5	3	8
6	9	15
10	8	18
24	24	48

b

9	12	21
5	3	8
4	6	10
5	4	9
23	25	48

c

3/7	4/7
5/8	3/8
2/5	3/5
5/9	4/9

Fig. 1. Two contingency tables, (a) and (b), with sample size 48 and row conditionals (c).

used in decomposition and heuristic approaches that depend on quickly solving subproblems with a coherent internal structure. This is an area on ongoing research, but the current paper contributes toward it insofar as we can incorporate bounds on cells and certain sums of cells.

Privacy typically focuses on preventing sensitive cells, such as those with very small counts, from being determined too accurately. There is an established tradition of statistical disclosure control being informed by operations research methods, particularly in the case of contingency tables [1,36,31,38,5]. Mathematical programming approaches have been used to implement controlled rounding or perturbation for tables [7,24,25,4,37] and as the basis for cell suppression methodologies [6,20,35]. Controlled rounding and perturbation (also called adjustment) replace cell values with nearby values, perhaps with the requirement that marginals are preserved or that conditionals still sum to unity along a row. Suppression omits some values from the table altogether, but compensates by providing marginals. In fact, contingency tables are often released in the form of marginals alone. These provide reasonable data utility, insofar as marginals are minimal sufficient statistics for parameters in associated log-linear models [19,10]. The problem of tightly bounding cell counts from tables of marginals has been well-studied in the literature (see the references above and also [3,11]).

More recently, researchers have explored the alternative of releasing observed *conditional* probabilities rather than marginals. Conditionals, which represent proportions of cell counts with specified characteristics, are of interest as relevant summaries and because they preserve odds and odds ratios [40]. Some governmental organizations routinely release tables of conditionals for which the corresponding tables of counts are released separately or not at all. In the United States, examples of such organizations include the Substance Abuse and Mental Health Services Administration [45] and the National Science Foundation [29]. Such releases are typically on a scale where privacy is not a big concern, but identifying cell bounds is still needed for the sake of inference. And as illustrated in the foregoing example involving Fig. 1, conditionals potentially offer lower disclosure risk than marginals and might therefore be of value to confidentiality. The study of disclosure risk for released conditional probabilities began about a decade ago [39,18,43,44], but the mathematical structure of the corresponding cell-bounding problems has only recently become well understood [42,46].

Previous studies [43,44] suggested that solving cell-bounding problems for two-way tables of conditionals might impose a heavy computational burden, but Wright and Smucker [46] showed that it is possible to greatly reduce the running time with a suitable reformulation of the problem. Those timing comparisons used the same general-purpose commercial solver on each formulation. The present article describes an easily coded and much faster procedure for determining cell bounds from exact conditionals and sample size. Utilizing the knapsack formulation of [46], we herein derive a streamlined dynamic programming approach and then significantly reduce its overall running time with a divide-and-conquer strategy. The new method not only reveals the cell bounds but all feasible values for each cell. Also, the method is fast enough to allow for real-time interactive exploration of disclosure risk over varying rearrangements and aggregations of a contingency table expressed in the form of conditionals. Consequently, this paper makes a key step toward the practical use of conditionals in statistical disclosure control, in addition to use of conditionals for statistical inference.

Dynamic programming is a very general algorithmic concept in optimization and computer science. It is widely recognized that specialized forms of dynamic programming are among the most efficient ways to solve knapsack problems [21,28,23,33]. In the present setting, however, we need to solve potentially hundreds or thousands of related knapsack problems for a single large contingency table. The key technical contribution of the paper is showing how this work can be organized so as to provide speed-ups of several orders of magnitude over applying the classical dynamic programming algorithm separately to each knapsack problem.

A related context for the application of dynamic programming to contingency tables is to counting, approximately, all possible two-way tables with a given set of marginal sums [12,8]. That work was motivated by efforts to sample from the set of all such tables [13,30], which in turn is relevant to statistical inference on contingency tables [9]. Dynamic programming is used in an entirely different way in the present paper. As will be seen in the next section, our paper essentially calculates bounds on marginal sums (and therefore cells) from conditional probabilities rather than cell bounds from specified marginals.

The paper is organized as follows. Section 2 lays out the notation and the knapsack formulation. The algorithm is described and justified in Section 3, along with implementation details. Section 4 describes the advantages gained by this procedure over previous approaches. In particular, we demonstrate that gaps in the bounds can be easily ascertained, we show that the algorithm's performance is superior (in this context) to that of the classical dynamic programming algorithm for knapsack problems, and we illustrate how the method can be used to explore disclosure risk for varying rearrangements and aggregations of a dataset.

2. Knapsack formulation of the cell-bounding problem

In this section, we review the problem formulation introduced by Wright and Smucker [46]. For the purpose of describing the problem formulation and algorithm, we consider a two-way contingency table having I rows and J columns. The observed count in cell ij is denoted o_{ij}, the row counts are $o_i = \sum_j o_{ij}$, and the sample size is $N = \sum_i o_i$. Assuming that only the sample size and observed conditional probabilities $\hat{p}_{ij} = \hat{p}(j|i) = o_{ij}/o_i$ are known, we wish to identify the tightest bounds for integers n_{ij} satisfying $\hat{p}_{ij} = n_{ij}/\sum_j n_{ij}$ and $\sum_{i,j} n_{ij} = N$. As noted below, we can also account for some types of additional information that may be provided.

A potential advantage of releasing data in the form of conditionals is that expressing the fractions $\hat{p}_{ij}$ in lowest terms tends to obscure most of the cell counts. The present paper deals with the possibility of recovering cell counts from those fractions and the sample size N. To address this problem, we calculate the greatest common divisors $d_i = \gcd(o_{i1}, \ldots, o_{ij})$ of the observed counts in each row i and use these to define the following positive integers:

- each cell's *reduced count* $r_{ij} = o_{ij}/d_i$;
- each row's *reduced sum* $r_i = o_i/d_i = \sum_j r_{ij}$;
- the entire table's *reduced total* $R = \sum_i r_i = \sum_{i,j} r_{ij}$.

Clearly, $\hat{p}_{ij} = r_{ij}/r_i$ and r_i is the lowest common denominator of $\hat{p}_{i1}, \ldots, \hat{p}_{iJ}$, so that the reduced counts r_{ij} are readily obtained from

the fractions $\hat{p}_{ij}$ alone. We therefore work with the reduced counts and their sums instead of the fractions $\hat{p}_{ij}$.

The reduced counts r_{ij} themselves provide lower bounds for the corresponding cells, and they also account for exactly R of the N observations. Our task is to determine which multiples of the r_{ij} can be added up to bridge the gap $N-R$ between the reduced total R and the sample size N. Because the reduced counts r_{ij} within a row must add up to the reduced sum r_i, we need only consider what multiple of r_i might make up the difference $o_i - r_i$. We denote that multiple by $\nu_i r_i$ for row i and write its minimum and maximum attainable values as $\nu_i^- r_i$ and $\nu_i^+ r_i$. The interval of tightest possible bounds on the cell count n_{ij} is therefore $[r_{ij}(\nu_i^- + 1),\ r_{ij}(\nu_i^+ + 1)]$. Because we treat the values o_i as unknown but summing to N, the cell-bounding problem for a given row $\bar{\imath}$ in the table amounts to these two optimization problems:

$$\min/\max \quad \nu_{\bar{\imath}} \quad \text{over all integers } \nu_i \tag{1}$$

$$\text{subject to} \quad \sum_i r_i \nu_i = N - R \quad \text{and } \nu_i \geq 0,\ \forall i. \tag{2}$$

These are integer *knapsack* problems. Although knapsack problems in general are NP-hard, they have been extensively studied and are used in many practical settings [28,23,33]. In particular, the non-negativity coefficients and lone equality constraint of our formulation makes the problem similar to various *change-making* problems that also have been well-studied in the literature. However, an important distinction with change-making problems is that we need to solve many related instances in which the objective functions differ from one instance to the next. A key contribution of the paper is to exploit the common constraint and the special form of the objective functions considered.

Wright and Smucker [46] noted that the knapsack formulation allows for much faster solution than the integer programming formulations proposed in earlier works. Specifically, when using the same commercial solver on both formulations, they indicated that runtimes were reduced by two orders of magnitude. In the present work, we obtain a speed-up of yet another two orders of magnitude by making several key modifications to a well-known knapsack algorithm. This yields a very efficient procedure for solving the entire collection of such problems for all indices $\bar{\imath}$.

In practice, data stewards may choose to release several summaries of the underlying contingency table, perhaps even involving different combinations of perturbations, suppressions, marginals, or conditionals. Consequently, it is important to consider the ramifications of combining information from multiple summaries. This is known to be a very challenging problem even when all the tables consist solely of marginals (see the references in the introduction). At the very least, it is worthwhile to ask what sort of information on cell bounds could be exchanged among tables, as in a decomposition method or an iterative bounding procedure such as the generalized shuttle algorithm [11]. Toward that end, we point out that the knapsack formulation can readily accommodate *a priori* bounds on individual cell counts or even on sums of cell counts within a row. In general, we might express bounds on a single such sum in the form $l \leq \sum_{j \in C} n_{ij} \leq u$, where C is some subset of the column indices. This amounts to $l \leq (\nu_i + 1)\sum_{j \in C} r_{ij} \leq u$, which can be solved for ν_i to yield

$$\frac{l}{\sum_{j \in C} r_{ij}} - 1 \leq \nu_i \leq \frac{u}{\sum_{j \in C} r_{ij}} - 1,$$

or equivalently,

$$\left\lceil \frac{l}{\sum_{j \in C} r_{ij}} \right\rceil - 1 \leq \nu_i \leq \left\lfloor \frac{u}{\sum_{j \in C} r_{ij}} \right\rfloor - 1.$$

Consequently, bounds on a sum of cell counts within a row amount to simple bounds on the cell counts themselves. These, in turn, are easily addressed by the methods considered herein (see Update 3.3 in Section 3.2).

Of course, published tables of conditionals would most likely use rounded decimal-place values instead of reduced fractions, thereby adding a further layer of protection. The assumption, used herein, of exact unrounded fractions can be viewed as a worst-case analysis of disclosure risk, as would be appropriate during the auditing phase performed by the data steward prior to table release. Slavković et al. [42] suggest that the knapsack formulation could also be used to calculate approximate (relaxed) bounds from rounded conditionals. However, integer programming formulations of *exact* cell bounds from rounded conditionals are more complicated and cannot be addressed by simply extending the method presented in this paper. Consequently, that matter lies beyond the scope of the present work and is left to a subsequent report [34].

3. A streamlined dynamic programming algorithm

The knapsack problems (1) and (2) of the previous section can be solved with commercial integer programming packages or various easily coded algorithms [28,23]. We describe a particularly efficient algorithm for calculating the extreme values ν_i^- and ν_i^+ for the entire table. Our method adapts a standard dynamic programming approach for solving knapsack and change-making problems by exploiting two key aspects of the particular context: only one variable appears in the objective at a time, and most of the underlying calculations must be repeated for many of the row indices i. As shown in Section 4.2, these result in speed-ups of orders of magnitude.

In this section, we first sketch a simple example illustrating the main ideas for refining and extending the classical knapsack dynamic programming method to the current setting of solving *2IJ* related knapsack problems. Then we give a formal description of the algorithm along with some suggestions for implementation, followed by a justification of the algorithm.

3.1. An illustration using a small dataset

We first look at a small example using an artificial dataset. Consider the two-way table of conditional probabilities shown in Table 1, which also appeared in the introduction. Our task is to determine which whole numbers ν_i yield $\sum_i \nu_i r_i = N - R = 19$. It turns out that there are precisely two ways to choose values for ν_1, ν_2, ν_3, and ν_4. Although they can be deduced easily with arithmetical reasoning, we work out the details with a systematic approach that generalizes to much larger tables.

We begin with an elementary reminder of how dynamic programming (DP) formalizes the idea of filling a knapsack with different types of items [2,21,28,23]. The method considers each type one at a time and examines how choosing a given number of that type might affect our ability to fill the remaining capacity of the knapsack with the types to be considered subsequently. When considering the first type of item, we ask the question: If someone else were to choose the number of items of other types, how might we top off the knapsack using just

Table 1
A small contingency table with $N=48$ and $R=29$.

	Response level		
Predictor level	α	β	r_i
A	3/7	4/7	7
B	5/8	3/8	8
C	2/5	3/5	5
D	5/9	4/9	9

items of the first type? For the knapsack associated with the contingency table above, each item of the first type occupies $r_1 = 7$ units of the total capacity $N - R = 19$ of the knapsack, so the answer is that we can fill any remaining capacity that is a whole-number multiple of 7 not exceeding 19. In other words, objects of size 7 can accommodate a remainder of 0, 7, or 14 units.

Next we consider the second type of item, keeping in mind what we know about the first type. An item of the second type in the contingency table above uses $r_2 = 8$ units, so we can handle any remaining capacity that can be reduced by a multiple of 8 to yield a multiple of 7. This means that items of size 7 and 8 can accommodate a remainder of 0, 7, 8, 14, 15 or 16 units out of the total 19. Then we move to the third type of item, and so on. The complete example is shown in Table 2.

Notice that the first and second rows exclude the target value 19, indicating that 19 is not a whole-number combination of 7 and 8. On the other hand, the third row does include 19, which can therefore be expressed as a combination of 7, 8, and 5. The fourth row shows the full list of numbers expressible using combinations of 7, 8, 5, and 9. However, in the present application we need to meet the total capacity 19 exactly. To address this restriction in the case of $r_4 = 9$, we discard the fourth DP row and instead determine which entries in the third row correspond to reducing 19 by some whole-number multiple of 9. We see that $19 = 19 - 0(9)$ and $10 = 19 - 1(9)$ both appear in the third row, whereas $19 - 2(9) = 1$ does not; moreover, all larger multiples of 9 exceed the target value 19. This tells us that the values $\nu_4 \in \{0, 1\}$ are the only possible choices, and therefore $\nu_4^- = 0$ and $\nu_4^+ = 1$.

We applied the DP algorithm only once, but it essentially solved both of the bounding knapsack problems associated with the row index $\bar{\imath} = 4$. Most optimization software packages, including those based on dynamic programming, would require solving the minimization and maximization problems separately. The key was that $\bar{\imath} = 4$ appeared last in the index list, so the only difference between minimizing and maximizing occurred at the very end. So when we optimize for a given index $\bar{\imath}$, we should place that index last.

Another important point is that we do not need to redo all of the work when we go on to optimize ν_3. Instead, we simply swap the order of indices 3 and 4 in the DP table, thereby reusing the work done earlier for indices 1 and 2 and shown in Table 3. This time we see that 19 cannot be expressed using a combination of 7, 8, and 9 alone. To optimize ν_3, we consider how to reduce 19 by multiples of 5 to obtain a value in the row shown for $i = 4$. We see that $14 = 19 - 1(5)$ and $9 = 19 - 2(5)$ are both in the table, but $19 = 19 - 0(5)$ and $4 = 19 - 3(5)$ are not. Therefore, $\nu_3^- = 1$ and $\nu_3^+ = 2$.

Although the optimizations for indices $\bar{\imath} = 1, 2$ do require us to start all over after shifting both toward the bottom, they only require one pass through the DP calculations for $i = 3,4$. In Fig. 2. We summarize the reuse of such prior DP calculations in terms of the row indices. The final two optimizations show that only $\nu_2 = 0$ will work, whereas the second shows that the possible values for ν_1 are 0 and 2. The resulting cell bounds for the original contingency table are shown in Table 4. Note that the cell counts in the second row are completely disclosed.

Table 2
DP table to optimize ν_4 for Table 1.

i	r_i	Remainders accommodated by r_1 through r_i
1	7	0, 7, 14
2	8	0, 7, 8, 14, 15, 16
3	5	0, 5, 7, 8, 10, 12, 13, 14, 15, 16, 17, 18, 19
4	9	0, 5, 7, 8, 9, 10, 12, 13, 14, 15, 16, 17, 18, 19

Table 3
DP table to optimize ν_3 for Table 1.

i	r_i	Remainders accommodated so far
1	7	0, 7, 14 (copied)
2	8	0, 7, 8, 14, 15, 16 (copied)
4	9	0, 7, 8, 9, 14, 15, 16
3	5	–

1st DP step	$1 = 1$		$3 = 3$	
2nd DP step	$2 = 2$		$4 = 4$	
3rd DP step	3	4	1	2
optimized index	4	3	2	1

Fig. 2. Reuse of DP calculations for Table 1.

Table 4
Cell bounds for Table 1.

	Response level	
Predictor level	α	β
A	[3, 9]	[4, 12]
B	[5, 5]	[3, 3]
C	[4, 6]	[6, 9]
D	[5, 10]	[4, 8]

For larger contingency tables, we can nest the reuse of DP table calculations by iteratively partitioning the row indices in a binary fashion. As an example, the analogous reuse of DP table elements for a contingency table having eight rows might correspond to the ordering shown in Fig. 3. For an odd number of rows, the partition would put roughly half into the top and bottom parts within each level.

The main aspects of the algorithm have now been illustrated. However, a change in its representation can do much to improve its performance. Instead of listing the specific values in a DP table as above, the details of the algorithm are tracked with a logical *viability* table of true and false values, indexed by integers from zero through the target sum $N - R$. When consideration of $i < \bar{\imath}$ has just been completed, the entry indexed by c in the viability table is true if a remaining quantity c can be accommodated by $r_{i'}$ for $i' \le i$; if it cannot, then entry c is false. When we have updated the viability table to account for all but the final index $\bar{\imath}$, it shows exactly which multiples of $r_{\bar{\imath}}$ are allowed: we simply stride backward from $c = N - R$ in increments of $r_{\bar{\imath}}$. The smallest such increment tells us the minimum value $\nu_{\bar{\imath}}^-$, and the largest such increment yields the maximum $\nu_{\bar{\imath}}^+$. The viability table is easily updated as we move from one index to the next. The foregoing ideas are captured in the algorithm presented below.

3.2. The algorithm

We begin with a formal statement of the main algorithm and then present some remarks on its operation and implementation. The expression $\lfloor a \rfloor$ denotes the greatest integer that is less than or equal to a, whereas $\lceil a \rceil$ is the least integer that is greater than or equal to a.

Algorithm 1. Set d:=1, δ:=1, L_1:=1, U_1:=I, and

$$f(c) := \begin{cases} \text{true}, & \text{for } c = 0, \\ \text{false}, & \text{for } c \in \{1, \ldots, N - R\}. \end{cases} \tag{3}$$

While $d > 0$ do the following:

1. If $L_d = U_d$, then:
   (a) Set i:=L_d.

1st DP step	1	= 1	= 1	= 1	5	= 5	= 5	= 5
2nd DP step	2	= 2	= 2	= 2	6	= 6	= 6	= 6
3rd DP step	3	= 3	= 3	= 3	7	= 7	= 7	= 7
4th DP step	4	= 4	= 4	= 4	8	= 8	= 8	= 8
5th DP step	5	= 5	7	= 7	1	= 1	3	= 3
6th DP step	6	= 6	8	= 8	2	= 2	4	= 4
7th DP step	7	8	5	6	3	4	1	2
optimized	8	7	6	5	4	3	2	1

Fig. 3. Reuse of DP calculations for Table 1.

(b) If $f(N-R-r_i\xi)$ is false for all whole numbers ξ, then the knapsack is infeasible (terminate). Otherwise, set

$\nu_i^- := \min\{\xi | f(N-R-r_i\xi) \text{ is true}\}$,

$\nu_i^+ := \max\{\xi | f(N-R-r_i\xi) \text{ is true}\}$.

(c) Set $\delta := -1$ and go to step 4.

2. If $\delta = 1$, then:
   (a) Set $L_{d+1} := \lceil (U_d + L_d)/2 \rceil$ and $U_{d+1} := U_d$.
   (b) If $d > 1$, then for $c := 0, \ldots, N-R$ set $f_{d-1}(c) := f(c)$.
   (c) For $i := L_d, \ldots, L_{d+1} - 1$, update f [see details below].
   (d) Go to step 4.
3. If $L_d < L_{d+1}$, then:
   (a) Set $U_{d+1} := L_{d+1} - 1$ and $L_{d+1} := L_d$.
   (b) If $d > 1$, then for $c := 0, \ldots, b$ set $f(c) := f_{d-1}(c)$. Otherwise, reset f using (3).
   (c) For $i := U_{d+1} + 1, \ldots, U_d$, update f [see details below].
   (d) Set $\delta := 1$ and go to step 4.
4. Set $d := d + \delta$.

In Algorithm 1, the index set $\{1, \ldots, I\}$ is iteratively partitioned into finer subsets at step 2a. The variable d denotes the depth within these nested partitions, whereas $\delta \in \{\pm 1\}$ denotes the direction of motion (deeper or shallower). The values L_d and U_d give the range of indices i being processed in the current pass through the while-loop. The function f denotes the viability table of true-false values indicating whether a value c can be expressed in terms of the r_i-values for all i considered so far. The functions f_d denote copies of f that are saved for later reuse.

The viability table update indicated in steps 2c and 3c should be implemented in different ways according to the needs of the audit being performed on the contingency table. More precisely, the data releaser might know that separately released information provides bounds on some row sums. If none of the variables have such *a priori* upper bounds tighter than the trivial bound $\lfloor (N-R)/r_i \rfloor$, then the following update is most efficient. The notation $a \vee b$ denotes inclusive disjunction (the OR-operator) for logical values a and b.

Update 3.2 (*All variables are unbounded*). If $f(r_i)$ is false, then for $c := r_i, \ldots, N-R$ in this order, set $f(c) := f(c) \vee f(c-r_i)$.

Notice that the above update implies that no action is needed for row $i < \bar{\imath}$ when $f(r_i)$ is true. If some rows do have *a priori* bounds u_i that must be enforced, then we can use the following update of f.

Update 3.3 (*Some variables are bounded*). Every variable falls into one of these two cases.

- If the variable has no *a priori* bound, or if that bound satisfies $u_i \geq \lfloor (N-R)/r_i \rfloor$, then for $c := r_i, \ldots, N-R$ in this order, set $f(c) := f(c) \vee f(c-r_i)$.
- If $u_i < \lfloor (N-R)/r_i \rfloor$, then do:
  - For $c := 0, \ldots, N-R-r_i$, set $\varphi(c) := f(c)$.
  - For $k := 1, \ldots, u_i$ and for $c := kr_i, \ldots, N-R$, set $f(c) := f(c) \vee \varphi(c-kr_i)$.

We now make a few notes on efficient implementation. First, representing f as a bit-string or vector of boolean variables allows for logical indexing of vectors, which can improve speed dramatically in some computing environments. Second, considerable savings can be realized by replacing $N-R$ in all of the for-loops of Algorithm 1 and Update 3.3 with $\gamma := \max_c\{0 \leq c \leq N-R : f(c) \text{is false}\}$ and skipping the update of f at i whenever $r_i > \gamma$. Third, if the extremes are only needed for some selected subset of variables, then those indices should be placed last so that they're relabeled as $\bar{\imath}, \ldots, I$. In this case, Algorithm 1 is terminated immediately after step 1b during the iteration in which $i = \bar{\imath}$. Fourth, the following simplifications should precede the invocation of Algorithm 1:

- If either $r_{\bar{\imath}} > N-R$ or $N-R-\min_i r_i < r_{\bar{\imath}} < N-R$, then $\nu_{\bar{\imath}}^+ = \nu_{\bar{\imath}}^- = 0$.
- If $r_{\bar{\imath}} = N-R$, then $\nu_{\bar{\imath}}^+ = (N-R)/r_{\bar{\imath}}$ and $\nu_i^- = 0$ for all $i \neq \bar{\imath}$.
- If $r_{\bar{\imath}} = 1$, then $\nu_i^+ = \lfloor (N-R)/r_i \rfloor$ for all i and $\nu_i^- = 0$ for all $i \neq \bar{\imath}$.

These checks potentially render the problem much simpler or even trivial.

The following section provides additional justification and details, but can be omitted for readers interested only in the algorithm and its results.

3.3. Justification

We now provide a few more details regarding Algorithm 1. Specifically, we derive the algorithm from the classical knapsack dynamic programming approach for solving an equality-constrained knapsack or change-making problem. Recall that the issue here is to address potentially hundreds of different (but very special) objective functions simultaneously.

Momentarily assume that we are interested only in obtaining the minimum value ν_I^- for the final row in the contingency table. The classical knapsack DP algorithm for this situation is given by the following well-known recurrence [21,28,23,33]:

Algorithm 2. Set $g_0^-(c) := \begin{cases} 0, & \text{if } c = 0, \\ \infty, & \text{if } c \in \{1, \ldots, N-R\}. \end{cases}$

For $i := 1, \ldots, I$, do the following:
For $c := 0, \ldots, \min\{N-R, r_i - 1\}$, set $g_i^-(c) := g_{i-1}^-(c)$.
For $c := r_i, \ldots, N-R$ in this order, set $g_i^-(c) := \min\{g_{i-1}^-(c), g_i^-(c-r_i) + \chi_i\}$.

Here χ_i is one if $i = I$ and zero otherwise. If the problem is feasible, then the recurrence ends with $g_I^-(N-R) = \nu_I^-$; otherwise, it ends with $g_I^-(N-R) = \infty$. To calculate the maximum value ν_I^+, we simply replace ∞ with $-\infty$ and 'min' with 'max' in the expressions defining g_i. Algorithm 2 requires $O((N-R)I)$ time and space.

To derive Algorithm 1 from Algorithm 2, we begin with three key observations. First, for $i < I$ we see that $g_i^-(c) = 0$ (equivalently, $g_i^-(c)$ is finite) if and only if c is a whole-number linear combination of $r_1, \ldots, r_i$. This implies $g_i^-(c) = -g_i^+(c)$ for all $i < I$, so that the maximization and minimization are carried out simultaneously with no additional effort prior to the final stage $i = I$. Second, because we don't need the full solution vector $(\nu_1, \ldots, \nu_I)$ for either extreme of ν_I, we avoid storing intermediate information for subsequent backtracking. That is, rather than store a list of tables such as g_i^-, we can simply update a single table f from one i to the

next. Moreover, we can represent f using logical entries interpretable, at the completion of stage $i < I$, as $f(c) =$ true if and only if c is a whole-number linear combination of $r_1, \ldots, r_i$. Third, suppose we wish to optimize ν_{I-1} immediately after optimizing ν_I. Rather than redoing all the work for each $i < I-1$, we save g_{I-2}^- for reuse. We then optimize ν_{I-1} cheaply by exchanging the indices $i=I$ and $i=I-1$ and repeating the final two iterations of procedure.

The goals of the last two observations in the preceding paragraph are in conflict: we want to avoid storing all the tables g_i^- without redoing all the prior work. As noted in Section 3.1, a reasonable middle ground lies in a divide-and-conquer strategy. First, we partition the index set $\{1, \ldots, I\}$ into two nearly equal subsets. During the dynamic programming algorithm to optimize ν_I, we save a copy of f when we reach middle index. Then we partition the second half of the index set into two nearly equal sets. When we're at the middle of this second half, we save another copy of f. This process is continued until we reach ν_I, after which we revert to the most recent subpartition and swap the order of the two index sets to be processed. In this way, we need at most $\lfloor \log_2 I \rfloor$ prior copies of f to be kept at any time. The resulting algorithm requires $O((N-R)I \log I)$ time and $O((N-R)\log I)$ space. The algorithm is identical to Algorithm 1 using Update 3.2 in Section 3.2, once we make one further modification. We need not update f at any i for which the previous index has left us with $f(r_i)$ as true because this says that r_i is a whole number combination of the coefficients that preceded it in forming f. In other words, the item i contributes nothing new to feasibility.

Note that Algorithm 1 only reaches step 1a if $\delta = 1$ and it only reaches step 3a if $\delta = -1$. An increase in depth leaves us either at step 1a where we process an isolated index i, or at step 2a where we subdivide the current partition. A decrease in depth leaves us either at step 4 with no other action to take, or at step 3a where we swap the two index sets of the current partition. Steps 2b and 3b are where we store and retrieve copies of f. Aside from initialization, the underlying dynamic programming algorithm consists of steps 1b, 2c, and 3c.

The test in step 1(b) of Algorithm 1 is superfluous when no *a priori* bounds are used. It is included for when an audit of the table includes examining the effect of additional information on disclosure risk.

We conclude with a brief discussion on the treatment of explicit bounds for variables. An easy pre-processing step disposes of nonzero finite lower bounds on variables. Variables with upper bounds $u_i \in \mathbf{Z}_+$ can be handled with Update 3.3, for which the unbounded case is almost the same as Update 3.2. The bounded case is justified by the observation (which can be proved inductively) that c is a feasible combination of the coefficients included up through r_i if and only if $c - kr_i$ is a feasible combination of the values preceding r_i for some $0 \le k \le u_i$. Note that the presence of bounded variables precludes the efficiency gained in Update 3.2 by skipping an update when $f(r_i)$ is true. In general, the updating procedure for a bounded variable involves $(N-R+1-u_i r_i/2)u_i$ multiplications, which is $O(N-R)$ for small u_i and $O((N-R)^2)$ for $r_i \approx 1$ and $u_i \approx N-R$ (recall that $u_i < (N-R)/r_i$). Typically, the latter situation is not of practical concern because very large values of u_i are unlikely to contribute much to disclosure risk. We mention that the update can be addressed in $O(N-R)$ time by a more complicated procedure that eliminates much of the redundant calculation in Update 3.3 for the bounded case, but we omit the details.

4. Results and discussion

In this section we report on key advantages gained by employing the algorithm presented in Section 3.2. First, we illustrate via example how this procedure detects gaps in the bounds and, moreover, clearly shows not only bounds but every feasible value that a cell can take on. Second, we compare the running-time performance of our dynamic programming algorithm to that of a commercial integer programming software package. Third, we describe how the rapid calculation of bounds enables an exploration of various table configurations in essentially real time, even for large tables. This allows quick assessments of disclosure risk.

4.1. Gaps within the integer bounds

An important aspect of the dynamic programming procedure is that it gives not only the bounds on each cell but all feasible values for each cell. Onn [32] noted that gaps in bounds can exist when the bounds are calculated from given marginals, whereas Slavković et al. [41] use algebraic methods to demonstrate that gaps can exist in cell bounds when calculated given conditional probabilities and sample size. Here, we demonstrate that in the case of released conditionals and sample size, these gaps—and indeed, all feasible values for all cells—are easily obtained.

We consider a $2^3 \times 3$ table (from [27]) giving the number of patients in a clinical trial, with three binary predictor variables (Table 5 with $X_1 =$ Center, $X_2 =$ Status, $X_3 =$ Treatment).

Here we consider it a two-way 8×3 table with Recovery as the response variable. For our example, we focus on the fifth row, which has a small reduced row sum of 2. Wright and Smucker [46] showed that the bounds for the cells in this row are [1, 18], [1, 18], and [0, 0], respectively. For the first two of those cells, we ask which integers within the given interval are feasible values.

Consider Table 6, which is analogous to Tables 2 and 3 in Section 3.1. (Here we have reordered the rows for convenience; in an algorithmic implementation, such a reordering would not be performed.) Note that we have represented all rows in Table 6 except for row 5.

Using the reasoning demonstrated in Section 3.1, we start by noting that $N-R=34$ appears in the row labeled 8. This means that $\nu_5 = 0$ can be accommodated by the implied sets of conditional probabilities. By stepping through the list in this row by multiples of 2, we can see that the list of values for row 5 is $\{0, 1, 2, 3, 5, 6, 8, 9, 11, 14, 17\}$. Thus, we immediately obtain the complete list of possible values for a given row, which is a good deal more information than the bounds $[\nu_5^-, \nu_5^+] = [0, 17]$ obtained using a general integer-program solver. These row bounds correspond to bounds of [1, 18] for the first two cells of the fifth row (the third cell has a zero count). We now see that these cells can take on any of the values $\{1, 2, 3, 4, 6, 7, 9, 10, 12, 15, 18\}$, but no others. Clearly, the same can be accomplished for all cells in this table.

Table 5
$2^3 \times 3$ table of observed counts with $N=193$ and $N-R=34$.

X_1	X_2	X_3	Recovery			o_i	r_i
			Poor	Modest	Excel.		
1	1	1	3	20	5	28	28
		2	11	14	8	33	33
	2	1	3	14	12	29	29
		2	6	13	5	24	24
2	1	1	12	12	0	24	2
		2	11	10	0	21	21
	2	1	3	9	4	16	16
		2	6	9	3	18	6

Table 6
DP table to optimize ν_5 for Table 5.

i	r_i	Remainders accommodated by row sums up through r_i
1	28	0, 28
2	33	0, 28, 33
3	29	0, 28, 29, 33
4	24	0, 24, 28, 29, 33
6	21	0, 21, 24, 28, 29, 33
7	16	0, 16, 21, 24, 28, 29, 32, 33
8	6	0, 6, 12, 16, 18, 21, 22, 24, 28, 29, 30, 32, 33, 34

4.2. Computational tests

We performed a series of tests to demonstrate the speed-ups to be gained by the dynamic programming algorithm presented herein. As noted by Wright and Smucker [46], the shift from a general integer programming formulation to a knapsack formulation can reduce runtimes for large instances by two to three orders of magnitude (i.e., from several hours to under a minute). With our adapted dynamic programming approach, we improve some of these times to just milliseconds.

Our timing tests employ three contingency tables drawn from the literature. Each table gives frequencies (counts) recorded as a k-way table, in which every dimension represents a discrete random variable with a small number of levels. To create a table of conditional probabilities, we designate some of the random variables as *predictor* (factor) variables and others as *response* variables, thus flattening the table to two dimensions. The levels of the predictors correspond to the rows of the summary table and the levels of the responses give the columns. We simply aggregate over any random variables not designated as predictors or responses. The datasets include a 6-way table [14] with a $2^5 \times 2$ summary, an 8-way table (from the U.S. Census Bureau's 1993 Current Population Survey) with several different summaries, and a 16-way table [15–17] with a $2^{15} \times 2$ summary. From these three datasets we constructed twelve two-way tables of conditionals according to the row and column selections shown in Table 7. The two-way tables labeled B–J have no zero cells at all, whereas tables L–O each have at least one zero cell (and therefore at least one row with $r_i = 1$).

The two-way tables H–J are derived from the large tables M–O by removing all rows that have one or more zero cells. In effect, all such rows are combined into a single row that might be labeled 'Other'. This is a coarse means of limiting the disclosure risk posed by zero cell counts, which potentially reveal that a subject known to belong to a given row must satisfy the conditions of a complementary column within that row. In practice, it's preferable to aggregate such rows into structurally meaningful categories, using ideas similar to those detailed in the next section.

The table rearrangements, knapsack formulations, and dynamic programming calculations were performed in the MATLAB 7.14 environment. The proposed Algorithm 1 was implemented using Update 3.2 and the additional notes given at the end of Section 3.2, whereas the classical dynamic programming method was implemented directly as given in Algorithm 2. Although the classical DP algorithm offers favorable theoretical complexity for solving knapsack problems relative to other standard methods for combinatorial optimization, it is sometimes outperformed in practice by carefully implemented branch-and-bound procedures [28]. Therefore, we also solved the knapsack problems separately by calls to CPLEX 12.5 (with default settings), a popular commercial software package for integer and linear optimization. All times reported here were obtained on a 2 × quad-core Intel 64-bit (3.40 GHz, 8 GB RAM) platform running the Windows 7 Enterprise operating system.

Table 7
Two-way table configurations used in computational timing tests.

2-way		Random variables			$o_{ij} = 0$
Table	Dataset	Row	Col.	Omit	Removed
B	8-way	4, 6, 7	8	1, 2, 3, 5	–
C	8-way	3, 5, 6, 7	8	1, 2, 4	–
D	8-way	3, 4, 6, 7	8	1, 2, 5	–
F	8-way	1, 3, 6	8	2, 4, 5, 7	–
G	16-way	8, 10–13	14, 16	1–7, 9, 15	–
H	8-way	1, 3–7	8	2	Y
I	8-way	1–7	8	–	Y
J	16-way	1–13, 15, 16	14	–	Y
L	6-way	1–5	6	–	N
M	8-way	1, 3–7	8	2	N
N	8-way	1–7	8	–	N
O	16-way	1–13, 15, 16	14	–	N

Table 8
Results and table dimensions for timing tests.

	Solution time (sec.)			I	nnz	J	N
	Classical DP Alg.	CPLEX	Proposed DP Alg.				
B	0.0351	1.7954	0.0056	12	12	2	48 842
C	1.3501	0.9528	0.0155	60	60	2	48 842
D	1.1794	2.6212	0.0187	60	60	2	48 842
F	0.4263	0.2274	0.0089	30	30	2	48 842
G	0.0184	1.0715	0.0042	32	32	4	21 574
H	21.1923	1.6919	0.0185	240	240	2	44 381
I	98.1193	4.7952	0.0352	557	557	2	41 465
J	54.0825	4.3326	0.0356	672	672	2	17 563
L	0.0121	0.1182	0.0031	32	32	2	1841
M	60.6576	1.6898	0.0077	360	347	2	48 842
N	686.2158	7.3585	0.0204	1440	1138	2	48 842
O	1157.4670	22.6600	0.0542	32 768	2480	2	21 574

The resulting solution times are shown in Table 8, along with information about the contingency table sizes ('nnz' refers to the number of nonzero rows). The classical dynamic programming algorithm is competitive with CPLEX only on the smallest instances and is generally much slower than the adapted DP approach proposed herein. The sometimes vastly superior performance of CPLEX over the classical DP algorithm might surprise some practitioners but can be explained by several observations. First, CPLEX is not just a general-purpose solver but also includes many sophisticated procedures for handling frequently occurring and easily identified problem structures [26]. In particular, knapsacks constitute an important type of subproblem commonly encountered in MIP-solution paradigms and CPLEX readily exploits structure among the coefficients in many of the instances considered here. Second, only the objective function was changed in the many calls to CPLEX for the sequence of knapsack problems associated with a given contingency table, so some information from prior solutions (such as reductions by the pre-solver and some types of cuts) could be reused. Third, it is not so surprising that a MATLAB implementation of the classical DP algorithm (with thousands of repeated calls and the associated implications for memory management) might get bogged down on larger instances.

On the other hand, the proposed adaptation of the dynamic programming algorithm is designed specifically to address the repetitive optimization, even though it uses the same core calculations as the classical DP approach (see Section 3.3). At worst, the adapted dynamic programming implementation is 26 times faster

than CPLEX in optimizing all of the bounds. Eight of the twelve instances show speed-ups (versus CPLEX) of over two orders of magnitude. More importantly, the longest running time of Algorithm 1 for any instance is 54.2 ms. This means that tables with several thousand rows can be processed fast enough to allow a real-time exploration of disclosure risk under a variety of table redesigns in which levels and/or variables are aggregated. That possibility is discussed in greater detail in the next section.

4.3. Interactive exploration of disclosure risk

The methodology presented in this paper allows calculation of cell bounds, even for very large tables, in real time using (say) a graphical user interface. Consequently, it provides an opportunity for data releasers to explore various table configurations by combining variable categories and/or aggregating over several variables, thereby ensuring that the data format ultimately released is not disclosive. As noted by Hundepool et al. [22, Section 5.3.32], such redesign of tables is an effective way to minimize the number of cells at risk while preserving the true counts, and can be used in conjunction with other disclosure control techniques. This section illustrates how such an exploration might proceed. The examples presented here were carried out interactively in just a few minutes.

We first make several observations to facilitate the exploration. As noted by Wright and Smucker [46], relatively dense tables are much more likely to disclose cell counts, though they are less likely to reveal *small* counts. The reason is that dense tables tend to have few small counts in the first place and are therefore more likely to have relatively prime cell counts within most rows, especially if there are three or more columns.

At the other end of the spectrum are relatively sparse tables with many small cell counts. These tables pose a much lower risk for disclosure of any nonzero row, provided the number of columns is small. However, the risk rises as the number of columns increases, for the same reasons noted above. In the case of a large sparse table, an acceptable set of (aggregated) conditionals is one that avoids problematic disclosures while preserving as much data specificity as possible. As we illustrate below, the data releaser can control the sparsity of a dataset by aggregating either within a variable (by combining categories) or across a variable (by omitting it from a role as either a response or predictor).

For our examples, we focus on the 8-way table ($N=48\,842$) mentioned previously. This dataset is from the 1993 U.S. Current Population Survey (CPS), a monthly survey that collects demographic and other data of interest. Table 9 gives information about the variables. If the first seven variables are used to predict salary, then there are 1185 zero cells and 302 zero rows (see the first line of Table 10). Consequently, even though there are many small counts, there are no disclosed cells with positive counts and this basic table could likely be released without concern.

Arranging a multiway table, as above, into two columns for a lone response variable may obscure too much information on the relationships among the other variables. Suppose that, in addition to salary (X_8), the number of hours worked (X_7) is designated as a response. This situation is shown in the second line of Table 10, where we see that several small cells are now disclosed (including 11 cells with a count of 1) and hence privacy is potentially compromised. Consider, for instance, the predictor levels $X_1 =$ '<25', $X_2 =$ 'Private', $X_3 =$ 'College', $X_4 =$ 'Unmarried', $X_5 =$ 'White', $X_6 =$ 'Male'. Together, these yield exactly one person in the dataset who worked less than full-time ($X_7 =$ '<40') and earned a relatively high salary ($X_8 =$ '>50'). This cell is disclosed if the conditionals are released in this configuration.

To attempt to remedy this while maintaining the same predictor/response structure, the data steward might collapse categories within a variable or aggregate over a relatively unimportant variable. It can quickly be ascertained that aggregating over any of the 6 predictors alone will not produce a table free of small disclosures (lines 3–8 in Table 10). Furthermore, aggregating X_3 into just two categories (college degree or no college degree)

Table 9
CPS variables and their levels.

Variable	Interpretation	Levels
X_1	Age	<25, 25–55, >55
X_2	Employment	Gov't, Private, Self-employed, Other
X_3	Education	$<$HS, HS, College, Bachelor, Bachelor+
X_4	Marital Status	Married, Unmarried
X_5	Race	Non-white, White
X_6	Sex	Female, Male
X_7	Hours Worked	<40, 40, >40
X_8	Salary	<50, 50+

Table 10
First exploration of 8-way CPS table with $N=48\,842$. Each line summarizes a table reconfiguration in the sequence of steps constituting the exploration.

Partition of random variables			$I \times J$	# of rows			# of disclosed cells	
Rows	Cols.	Omit		All zero	$r_i=1$	Nonzero disclosed	$n_{ij}=0$	$0<n_{ij}<5$
1–7	8	–	1440×2	302	581	0	1185	0
1–6	7, 8	–	480×6	52	36	30	1185	17
2–6	7, 8	1	160×6	1	1	13	149	3
1, 3–6	7, 8	2	120×6	1	3	4	133	1
1, 2, 4–6	7, 8	3	96×6	2	1	20	112	8
1–3, 5, 6	7, 8	4	240×6	11	7	38	413	22
1–4, 6	7, 8	5	240×6	11	8	17	382	12
1–5	7, 8	6	240×6	14	12	18	432	10
1 , 2, 3[a], 4–6	7, 8	–	192×6	11	6	89	327	79
1, 2, 3[b], 4–6	7, 8	–	192×6	7	7	5	324	1
1[c], 2, 3[a], 4–6	7, 8	–	128×6	2	3	92	126	100
1[c], 2, 3[b], 4–6	7, 8	–	128×6	1	3	44	133	25
1–6	7[d], 8	–	480×4	52	39	1	695	0

[a] X_3 has been aggregated to two levels, 'Bachelor degree' and 'No Bachelor degree'.
[b] X_3 has been aggregated to two levels, 'No college' and 'At least some college'.
[c] X_1 has been aggregated to two levels, '≤ 55' and '>55'.
[d] X_7 has been aggregated to two levels, '<40' and '40 or more'.

results in many more small disclosed cells (line 9, Table 10). On the other hand, aggregating X_3 in a slightly different way ('No college' vs. 'At least some college') results in relatively few disclosures and only a single small disclosure (line 10, Table 10). The disclosure is a cell with just a single count (predictors $X_1 =$ '25–55', $X_2 =$ 'Private', $X_3 =$ 'No college', $X_4 =$ 'Unmarried', $X_5 =$ 'White', $X_6 =$ 'Female', with response $X_7 =$ '<40' and $X_8 =$ '>50') tucked into a row that has a total count of 2261.

As can be seen, it is increasingly difficult to find an acceptable table as the number of response columns grows. This is because it becomes less likely that row sums can be reduced, which makes cell disclosures more likely. In a sparse table such as this with many small cells, row disclosures often mean small-cell disclosures. Although aggregating within or across variables yields a table that is less sparse (reducing the number of cells with small counts), it tends to tighten the bounds and results in more small-cell disclosures, not fewer (see lines 11 and 12 of Table 10). If this approach is taken, the table must be made sufficiently dense (i.e., more compact) to leave few or no cells with small counts.

All of this suggests that the most effective redesigns for reducing disclosure risk might focus instead on reducing the number of response columns. However, rather than simply reverting to a single response variable (as at the beginning of this exploration), it might be preferable to aggregate categories within some of the current columns among the two response variables currently under consideration. For instance, suppose we recalculate the bounds after aggregating X_7 into two categories ('<40' and '≥ 40') instead of three and returning X_1–X_6 to their original categories. By comparing the second and final lines of Table 10, we see that the number of small disclosed cells (counts of 4 or less) is reduced from 17 to 0. Also, the number of disclosed nonzero rows is reduced from 30 (with 167 nonzero cells) to just 4 (with 4 nonzero cells). This demonstrates how greater control of small positive counts can be exerted by adjusting the number of response columns rather than the number of predictor rows. The same approach can also deal with problematic zero cells within nonzero rows, although clearly aggregation of rows is necessary to eliminate all-zero rows.

4.4. Conclusion

This paper exploits a recently discovered knapsack formulation of the cell-bounding problem when the sample size and unrounded conditional probabilities are released. A practically efficient algorithm is developed by incorporating dynamic programming within a binary divide-and-conquer strategy. The resulting procedure has several desirable properties. First, it reveals not only the bounds on each cell, but all possible values that each cell could take on. Second, it is orders of magnitude faster than previous approaches. Third, its speed enables real-time exploration and reshaping of the data to ensure that the released table is not disclosive.

Several important directions for further research remain. One is to extend the approach presented herein to handle the additional complications that arise when published values are rounded; a paper on that issue is in preparation. Another avenue concerns the evaluation of disclosure risk when releasing several tables of conditionals (or a mix of conditionals and marginals). Finally, there is much work to be done concerning how best to use the cell-bound information for the purpose of statistical inference.

Acknowledgments

The authors thank Andrew J. Sage and the anonymous referees for suggestions that improved the exposition of the ideas presented herein. Miami University's "RedHawk" computing cluster was used for much of the preliminary work that led to this paper.

References

[1] Almeida MT, Carvalho FD. Exact disclosure prevention in two-dimensional statistical tables. Comput Oper Res 2005;32:2919–36.
[2] Bellman R. Dynamic programming. Princeton, NewJersy: Princeton University Press; 1957.
[3] Buzzigoli L, Gusti A. An algorithm to calculate the upper and lower bounds of the elements of an array given its marginals. In: Statistical data protection (SDP 1998) proceedings. Luxembourg; 1998. Eurostat. p. 131–47.
[4] Castro J. Minimum-distance controlled perturbation methods for large-scale tabular data protection. Eur J Oper Res 2006;171:39–52.
[5] Castro J. Recent advances in optimization techniques for statistical tabular data protection. Eur J Oper Res 2012;216(2):257–69.
[6] Cox L. Network models for complementary cell suppression. J Am Stat Assoc 1995;90(432):1453–62.
[7] Cox L, Ernst L. Controlled rounding. INFOR 1982;20:423–32.
[8] Cryan M, Dyer M, Randall D. Approximately counting integral flows and cell-bounded contingency tables. SIAM J Comput 2010;39(7):2683–703.
[9] Diaconis P, Efron B. Testing for independence in a two-way table: new interpretations of the chi-square statistic. Ann Stat 1985;13(3):845–913.
[10] Dobra A, Fienberg S, Rinaldo A, Slavković A, Zhou Y. Emerging Applications of Algebraic Geometry, volume 149 of The IMA Volumes in Mathematics and its Applications, chapter Algebraic statistics and contingency table problems: Log-linear models, likelihood estimation and disclosure limitation. Springer Science+Business Media, Inc.; 2009. p. 63–88.
[11] Dobra A, Fienberg SE. The generalized shuttle algorithm. In: Gibilisco P, Riccomagno E, Rogantin MP, Wynn HP, editors. Algebraic and geometric methods in statistics. Cambridge: Cambridge University Press; 2010. p. 135–56.
[12] Dyer M. Approximate counting by dynamic programming. In: Proceedings of the thirty-fifth annual ACM symposium on theory of computing. New York: ACM; 2003. p. 693–99.
[13] Dyer M, Kannan R, Mount J. Sampling contingency tables. Rand Struct Algorithms 1997;10(4):487–506.
[14] Edwards D, Havranek T. A fast procedure for model search in multidimensional contingency tables. Biometrika 1985;72:339–51.
[15] Erosheva EA. Bayesian estimation of the grade of membership model. New York: Oxford University Press; 2003.
[16] Erosheva EA. Statistical data mining and knowledge discovery. Boca Raton: Chapman & Hall/CRC; 2004.
[17] Erosheva EA, Fienberg SE, Joutard C. Describing disability through individual-level mixture models for multivariate binary data. Ann Appl Stat 2007;1:346–84.
[18] Fienberg SE, Slavković AB. Preserving the confidentiality of categorical statistical databases when releasing information for association rules. Data Min Knowl Discov 2005;11:155–80.
[19] Fienberg SE, Slavković, AB. A survey of statistical approaches to preserving confidentiality of contingency table entries. In: Aggarwal CC, Yu PS, Elmagarmid AK, editors, Privacy-preserving data mining, advances in database systems, vol. 34. US: Springer; 2008. p. 291–312.
[20] Fischetti M, Salazar-González JJ. Models and algorithms for the 2-dimensional cell suppression problem in statistical disclosure control. Math Program 1999;84:283–312.
[21] Gilmore PC, Gomory RE. The theory and computation of knapsack functions. Oper Res 1966;14:1045–74.
[22] Hundepool A, Domingo-Ferrer J, Franconi L, Giessing S, Nordholt ES, Spicer K, et al. Statistical disclosure control. Chichester: John Wiley & Sons; 2012.
[23] Kellerer H, Pferschy U, Pisinger D. Knapsack problems. Berlin: Springer; 2004.
[24] Kelly JP, Golden BL, Assad AA. Using simulated annealing to solve controlled rounding problems. ORSA J Comput 1990;2:174–85.
[25] Kelly JP, Golden BL, Assad AA, Baker EK. Controlled rounding of tabular data. Oper Res 1990;38:760–72.
[26] Klotz E, Newman AM. Practical guidelines for solving difficult mixed integer linear programs. Surv Oper Res Manag Sci 2013;18:18–32.
[27] Koch G, Amara J, Atkinson S, Stanish W. Overview of categorical analysis methods. SAS-SUGI 1983;8:785–95.
[28] Martello S, Toth P. Knapsack problems: algorithms and computer implementations. Chichester: John Wiley & Sons; 1990.
[29] Morgan M, Burrelli JS, Rapoport AI. Modes of financial support in the graduate education of science and engineering doctorate recipients. NSF Report 00-319, National Science Foundation, Arlington, VA; 2000.
[30] Morris BJ. Improved bounds for sampling contingency tables. Rand Struct Algorithms 2002;21(2):135–46.
[31] Muralidhar K, Sarathy R. Data shuffling: a new masking approach for numerical data. Manag Sci 2006;52:658–70.
[32] Onn S. Entry uniqueness in margined tables. In: Domingo-Ferrer J, Franconi L, editors, Privacy in statistical databases—PSD 2006. Lecture Notes in Computer Science, vol. 4302. Springer-Verlag; 2006. p. 94–101.
[33] Pisinger D, Toth P. Knapsack problems. In: Du DZ, Pardalos P, editors. Handbook of combinatorial optimization, vol. 1. Boston: Kluwer; 1998. p. 299–428.
[34] Sage AJ, Wright SE. Identifying tightest bounds on cell counts for contingency tables of rounded conditional frequencies. Working Paper, Miami University; 2013.

[35] Salazar-González J-J. Mathematical models for applying cell suppression methodology in statistical data protection. Eur J Oper Res 2004;154:740–54.
[36] Salazar-González J-J. Framework for different statistical disclosure limitation methods. Oper Res 2005;53:819–29.
[37] Salazar-González J-J. Controlled rounding and cell perturbation: statistical disclosure limitation methods for tabular data. Math Programm 2006;105(2–3):583–603.
[38] Salazar-González J-J. Statistical confidentiality: optimization techniques to protect tables. Comput Oper Res 2008;35:1638–51.
[39] Slavković AB. Statistical disclosure limitation beyond the margins: characterization of joint distributions for contingency tables [Ph.D. thesis]. Carnegie Mellon University; 2004.
[40] Slavković AB. Partial information releases for confidential contingency table entries: present and future research efforts. J Priv Confid 2010;1(2):253–64.
[41] Slavković AB, Fienberg SE. Bounds for cell entries in two-way tables given conditional relative frequencies. In: Domingo-Ferrer J, Torra V, editors, Privacy in statistical databases—PSD 2004, Lecture Notes in Computer Science No. 3050. Springer-Verlag; 2004. p. 30–43.
[42] Slavković AB, Zhu X, Petrović S. Fibers of multi-way contingency tables given conditionals: relation to marginals, cell bounds and Markov bases. Technical Report, The Pennsylvania State University; 2013.
[43] Smucker B, Slavković AB. Cell bounds in two-way contingency tables based on conditional frequencies. In: Domingo-Ferrer J, Saygin Y, editors, PSD 2008. Lectures Notes in Computer Science, vol. 5262. Springer-Verlag Berlin Heidelberg; 2008. p. 64–76.
[44] Smucker BJ, Slavković AB, Zhu X. Cell bounds in k-way tables given conditional frequencies. J Official Stat 2012;28(1):121–40.
[45] The TEDS report: Marijuana admissions aged 18 to 30: early vs. adult initiation. Substance abuse and mental health services administration, Center for Behavioral Health Statistics and Quality. Rockville, MD, 2013.
[46] Wright SE, Smucker BJ. An intuitive formulation and solution of the exact cell-bounding problem for contingency tables of conditional frequencies. J Priv Confid 2013;5:133–56.