Spring 2016

A Computational Investigation of Large Gaps in Contingency Tables

Noah J. Watson
James Madison University

A Computational Investigation of Large Gaps in Contingency Tables

An Honors Program Project Presented to

the Faculty of the Undergraduate

College of Science and Mathematics

James Madison University

by Noah James Watson

May 2016

Accepted by the faculty of the Department of Mathematics and Statistics, James Madison University, in partial fulfillment of the requirements for the Honors Program.

FACULTY COMMITTEE:	HONORS PROGRAM APPROVAL:
Project Advisor: Edwin O'Shea, Ph.D Associate Professor, Mathematics and Statistics	Bradley R. Newcomer, Ph.D., Director, Honors Program
Reader: Elizabeth A. Arnold, Ph.D Associate Professor, Mathematics and Statistics	
Reader: Lihua Chen, Ph.D Associate Professor, Mathematics and Statistics	

PUBLIC PRESENTATION

This work is accepted for presentation, in part or in full, at [venue] Department of Mathematics and Statistics

Colloquia on [date] April 25 .

Contents

1 Introduction

A major concern in information disclosure is to make sure when releasing information about a survey that nobody's privacy is compromised. Even when only information about the entire sample is released it is sometimes possible to reconstruct information about the individuals involved. If the information release is in the form of margins of a multi-dimensional contingency table then one technique for detecting disclosures of information is to use integer programming to find upper and lower bounds on the cells of the table using the information from the released margins. If the upper and lower bounds are far apart then there is no major disclosure of information. But if the bounds are tight then some information has been disclosed. The integer programming problem is NP-complete in computational complexity [9] but the linear relaxation of the integer program which can be solved in polynomial time. This raises the question of whether or not the bounds given by the linear relaxation are always faithful to the true bounds given by the integer program.

Some research has already been done on the gaps between the problems arising from contingency tables. It was thought [3] that the linear relaxation was always reliable for bounding cells. Sullivant constructed a family of tables on $n \geq 4$ binary random variables and a specified collection of margins such that the gap between the linear programming approximation of the cell bounds and the true integer programming cell bound for one of these margins is $2^{n-3} - 1$ and in doing so showed that the linear programming relaxation may not always a good method of detecting disclosures of information [10]. However, O'Shea later showed that Sullivant's gaps are statistically rare and in doing so gives credence to the notion that linear programming relaxations might still be a good method of detecting disclosures [7].

In this paper our goal is to provide an extensive catalog of the gaps that can occur on tables of few variables, in particular, on tables where there are no more than 5 variables. We will start with some background of the general theory of the integer programming gap. We will then show how to apply it to the contingency tables problem. Finally we will state our results.

2 Integer programming gap

The general integer programming problem in standard form is given by,

$$\text{Minimize } z \cdot c \text{ subject to } Az = b \tag{2.1}$$

where A is a $d \times k$ integer matrix, $b \in \mathbb{Z}^d$ and $c \in \mathbb{Q}^k$ are all fixed and z is allowed to vary where $z \in \mathbb{Z}^k$ and $z \geq 0$. The matrix A is called the constraint matrix and c is called the cost vector. The minimum of $z \cdot c$ is called the optimal value of 2.1 and the z that obtains this value is called an optimal solution. If we replace the condition that $z \in \mathbb{N}^k$ with $z \in \mathbb{R}^k$ and $z \geq \mathbf{0}$ then we obtain the *linear programming relaxation* of problem 2.1.

$$\text{Minimize } z \cdot c \text{ subject to } Az = b \tag{2.2}$$

where A is a $d \times k$ integer matrix, $b \in \mathbb{Z}^d$ and $c \in \mathbb{Q}^k$ are all fixed and z is allowed to vary where $z \in \mathbb{R}^k$ and $z \geq \mathbf{0}$. If an integer program is feasible and bounded then the linear relaxation is also feasible and bounded. Moreover, the optimal value of the linear relaxation is less than or equal to the optimal value of the integer program since the latter has more constraints to satisfy.

We define the *integer programming gap*, $\text{gap}(A, c)$, to be the maximum difference between optimal values of 2.1 and 2.2 as b ranges over all vectors such that 2.1 is feasible and bounded. It is this quantity that we wish to study in the context of the contingency table problem.

Given some constraint matrix A and cost vector c, we will outline a way to find $\text{gap}(A, c)$. A vector $u = (u_1, u_2, ..., u_k) \in \mathbb{N}^k$ is called non-optimal if it is not an optimal solution of 2.1 when we let $b = Au$. If we consider some polynomial ring $\mathbb{R}[x_1, x_2, ...x_k]$ we can represent each non-optimal vector u by $x^u = x_1^{u_1} x_2^{u_2} \cdots x_k^{u_k}$. Let $M(A, c)$ be the ideal in the polynomial ring $\mathbb{R}[x_1, x_2, ...x_k]$ generated by these x^u as u varies over all non-optimal vectors. We can compute $M(A, c)$ by taking the Graver basis of A and using [8, Algorithm 4.4.2] to find the largest monomial ideal contained in the ideal generated by the polynomial representation of the Graver basis [5]. A monomial ideal in $\mathbb{R}[x_1, x_2, ...x_k]$ is called irreducible if it is generated by powers of the variables. That is, if it is of the form,

$$I(u,\tau) = \left\langle x_{i_1}^{u_{i_1}+1}, x_{i_2}^{u_{i_2}+1}, ..., x_{i_r}^{u_{i_r}+1} \right\rangle$$

where $\tau = \{i_1, i_2, ..., i_r\}$ and $u \in \mathbb{N}^k$ and u is zero off of τ. Every monomial ideal M in $\mathbb{R}[x_1, x_2, ...x_k]$ can be written uniquely as an irredundant intersection of finitely many irreducible monomial ideals. These monomial ideals are called the irreducible components of M. We can define the gap value of each such component $I(u,\tau)$ with respect to a constraint matrix A and a cost vector c by the optimal value of the auxiliary linear program

$$\text{Maximize } u \cdot c - v \cdot c \text{ subject to } Av = Au \text{ and } \forall j \in \tau\ v_j \geq 0 \tag{2.3}$$

For each $I(u,\tau)$ define the set $(u,\tau) = \{u + v' \mid v' \in \mathbb{N}^k \text{ and } \forall j \in \tau\ v'_j = 0\}$. We call (u,τ) the *standard pair* for $I(u,\tau)$ and we say it has *root* u and *free directions* $\overline{\tau}$ [6]. The main result of Hoşten and Sturmfels [5] is that the gap value of $I(u,\tau)$ is equal to the maximum difference between the optimal values of 2.1 and 2.2 as b ranges over all vectors in $\mathbb{Z}^d$ so that the optimal solution of 2.1 is in (u,τ). Furthermore it also has been shown that gap(A,c) equals the maximum gap value of any irreducible component $I(u,\tau)$ of the monomial ideal $M(A,c)$. These results are given in [5] in the more general context of lattice programs. We will end this section with a demonstrative example.

Example Consider the family of integer programs indexed by $n \in \mathbb{N}$ that are given by

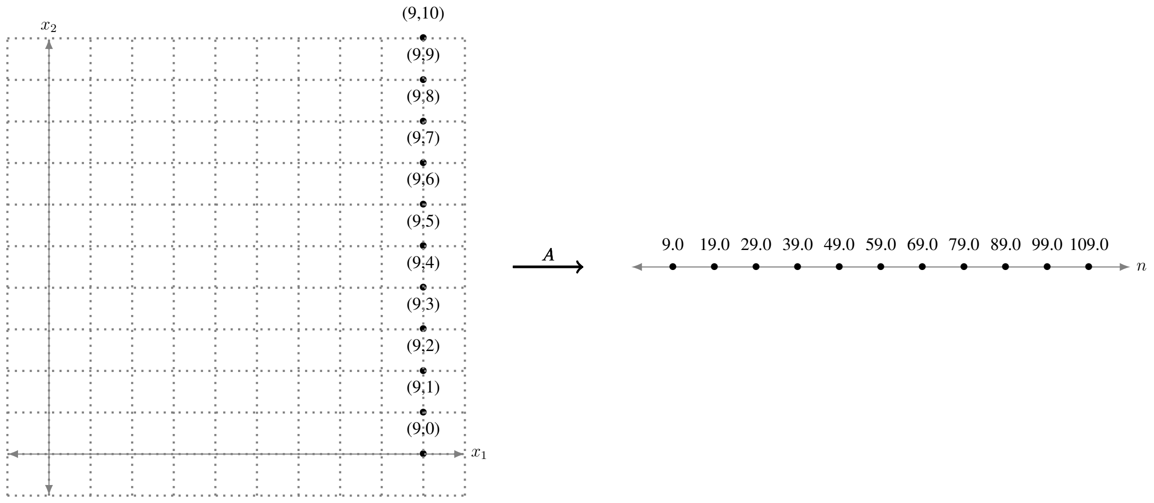
$$\text{Minimize } (x,y) \cdot (1,0) \text{ subject to } (1,10) \cdot (x,y) = n$$

where $(x,y) \in \mathbb{Z}^2$ and $x, y \geq 0$. Here $A = [1,10]$ and $c = (1,0)$. The corresponding family of linear relaxations are obtained by letting (x,y) be in $\mathbb{R}^2$ so that $x, y \geq 0$. Clearly, since the cost of y is 0, the optimal value of any of these linear relaxations is 0 and is obtained by letting $(x,y) =$

$(0, \frac{n}{10})$. On the other hand, the optimal value of any of the integer programs will always be the remainder of n when divided by 10. The largest this can be is 9. So gap$((1, 10), (1, 0)) = 9$. We can also find gap$((1, 10), (1, 0))$ by computing the Graver basis and finding the irreducible components of $M((1, 10), (1, 0))$. The Graver basis is just $\{[10, -1]\}$. So the ideal $M((1, 10), (1, 0))$ in the polynomial ring $\mathbb{R}[x_1, x_2]$ is given by $\langle x_1^{10} \rangle$. This is already irreducible. So we have just one component, $I(u, \tau)$ where $u = (9, 0)$ and $\tau = \{1\}$. We can find the gap value of the component by solving the auxiliary linear program

$$\text{Maximize } (9, 0) \cdot (1, 0) - (v_1, v_2) \cdot (1, 0) \text{ subject to } (1, 10) \cdot (v_1, v_2) = 9 \text{ and } v_1 \geq 0$$

where $(v_1, v_2) \in \mathbb{R}^2$. We can solve this using any of the common algorithms for linear programs. The optimal value is 9 and the optimal solution is $(0, \frac{9}{10})$. We can also find the standard pair for the one component, $(u, \tau) = ((9, 0), \{1\}) = \{(9, n) \mid n \in \mathbb{N}\}$. If we map (u, τ) via A then we get the set of right hand sides such that the optimal solution is in (u, τ).



▲

3 Contingency tables problem

Let us now turn our attention to using integer programs to bound cells of multi-dimensional contingency tables with the information from some set of margins. We can consider a n-dimensional contingency table to be given by a vector $d = \langle d_1, d_2, ..., d_n \rangle$ that specifies the number of levels in each dimension and a vector $y = \langle y_{11...1}, y_{11...2}, y_{11...3}, ..., y_{d_1 d_2 ... d_n} \rangle$ where $y_{i_1 i_2 ... i_n}$ corresponds to the cell entry in the i_jth level in the jth dimension. Each margin we release is just a lower dimensional contingency table and its values are simply higher dimensional analogues of row and column sums. We can think of a set of margins Δ as being specified by some collection of subsets of the set $\{1, 2, ..., n\}$ that specifies which relationships we release. We can assume that Δ is a simplicial complex since if we release the relationships between some set of variables we implicitly release the relationships between any subset of those variables. For example if we release the relationships between $\{1, 2, 3\}$, then we implicitly release the relationships between $\{1, 2\}$, $\{2, 3\}$, or even just $\{1\}$. This way of describing margins is called a *hierarchical model* [7].

Example Consider the 2-dimensional contingency table given by the following 2×5 table

Dim 2 / Dim 1	Level 1	Level 2	Level 3	Level 4	Level 5	Row sums
Level 1	20	30	10	15	10	85
Level 2	25	15	12	20	10	82
Col sums	45	45	22	35	20	

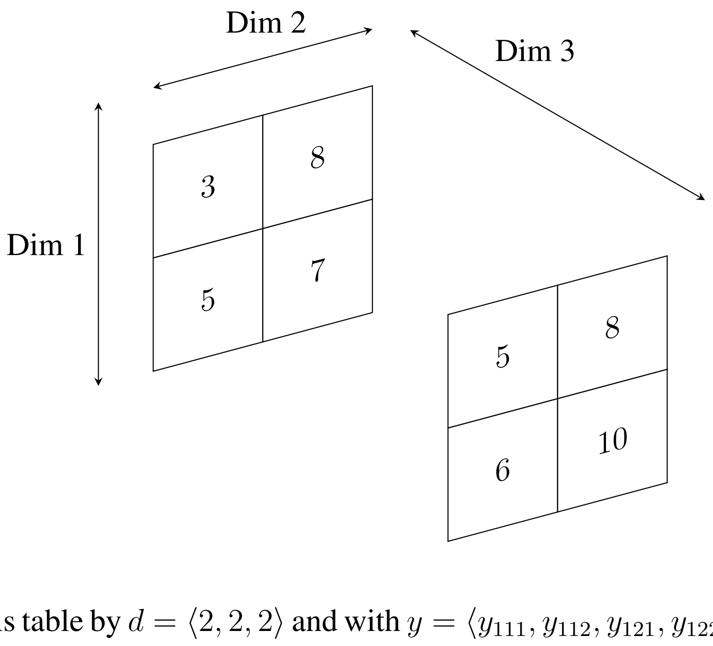
We can specify this table by $d = \langle 2, 5 \rangle$ and with $y = \langle y_{11}, y_{12}, y_{13}, y_{14}, y_{15}, y_{21}, y_{22}, y_{23}, y_{24}, y_{25} \rangle = \langle 20, 30, 10, 15, 10, 25, 15, 12, 20, 10 \rangle$. The margins obtained by row and column sums are specified by the set $\Delta = \{\{1\}, \{2\}\}$ and we get the 1-dimensional margin tables by summing over the variables not present. The set $\{2\}$ specifies the column sums and the set $\{1\}$ specifies the row sums. The problem we are interested in is the case when we know the margins and want to estimate or at least bound the cells. In the context of this example it would be like being given

Dim 2 / Dim 1	Level 1	Level 2	Level 3	Level 4	Level 5	Row sums
Level 1	y_{11}	y_{12}	y_{13}	y_{14}	y_{15}	85
Level 2	y_{21}	y_{22}	y_{23}	y_{24}	y_{25}	82
Col sums	45	45	22	35	20	

and having to bound some y_{ij}. ▲

Our goal is to rework this problem into an integer programming problem. To do this we will need a constraint matrix the encodes the relationship between the table and its margins. Given Δ we can construct a matrix A_Δ in the following way. For each maximal face F of Δ we construct $\prod_{i \in F} d_i$ many rows, one for each element of the margin. The columns are indexed by the elements of y. Each row has a 1 in the jth column if the jth element of y is part of the sum for the element of the margin that the row corresponds to and a 0 otherwise.

Example Consider the 3-dimensional contingency table given by



We can specify this table by $d = \langle 2, 2, 2 \rangle$ and with $y = \langle y_{111}, y_{112}, y_{121}, y_{122}, y_{211}, y_{212}, y_{221}, y_{222} \rangle = \langle 5, 3, 8, 8, 6, 5, 10, 7 \rangle$. Since $d_i = 2$ for all i, we call it a *binary table*. Suppose the set of released margins Δ has facets $\{1, 2\}$ and $\{1, 3\}$. The corresponding margins are given by the following

2-dimensional tables

Dim 1 \ Dim 2	Level 1	Level 2
Level 1	8	16
Level 2	11	17

Dim 1 \ Dim 3	Level 1	Level 2
Level 1	13	11
Level 2	16	12

The matrix A_Δ can be constructed in the way described above. The first half of the rows come from $\{1, 2\}$ and the second half come from $\{1, 3\}$.

$$A_\Delta = \begin{bmatrix} 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 \\ 1 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 1 \end{bmatrix}$$

Note that if we compute $A_\Delta y$ we get

$$\begin{bmatrix} 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 \\ 1 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 1 \end{bmatrix} \begin{bmatrix} 5 \\ 3 \\ 8 \\ 8 \\ 6 \\ 5 \\ 10 \\ 7 \end{bmatrix} = \begin{bmatrix} 8 \\ 16 \\ 11 \\ 17 \\ 11 \\ 13 \\ 16 \\ 12 \end{bmatrix} \begin{matrix} \left.\vphantom{\begin{matrix} 8 \\ 16 \\ 11 \\ 17 \end{matrix}}\right\} \{1,2\} \\ \left.\vphantom{\begin{matrix} 11 \\ 13 \\ 16 \\ 12 \end{matrix}}\right\} \{1,3\} \end{matrix}$$

which recovers our 2-dimensional margins. ▲

If we take $b = A_\Delta y$ them we can construct an integer program to bound cells of our contingency table in the following way. We can think of such right hand sides as collections of released tables of margins. Let e_j denote the vector that has a 1 in the jth coordinate and 0 elsewhere. Then the integer program

$$\text{Minimize } z \cdot e_j \text{ subject to } A_\Delta z = A_\Delta y \tag{3.1}$$

gives a lower bound on the jth element of our table. We know that 3.1 is feasible since y gives a solution, though y may not be optimal. Similarly the integer program

$$\text{Maximize } z \cdot e_j \text{ subject to } A_\Delta z = A_\Delta y \tag{3.2}$$

gives an upper bound on the jth element of our table. We can rewrite 3.2 as a minimize problem by just using $-e_j$ instead of e_j and taking the negation of the optimal value.

Example Let's return to our last example. Suppose that we are given the margins from before but we don't know the values of the original table. If we want an upper bound of y_{111} we can construct the integer program

$$\text{Maximize } z_{111} \text{ subject to } \begin{bmatrix} 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 \\ 1 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 1 \end{bmatrix} \begin{bmatrix} z_{111} \\ z_{112} \\ z_{121} \\ z_{122} \\ z_{211} \\ z_{212} \\ z_{221} \\ z_{222} \end{bmatrix} = \begin{bmatrix} 8 \\ 16 \\ 11 \\ 17 \\ 11 \\ 13 \\ 16 \\ 12 \end{bmatrix}$$

Solving this using any of the standard means yields the upper bound of 8 for y_{111}. ▲

Without loss of generality we will usually let $j = 1$ and just consider bounding $y_{11\ldots1}$. Our main focus is in finding the maximal difference between 3.1 and 3.2 and their linear relaxations as y varies over all vectors in $\mathbb{Z}^k$. This is the same as letting the right hand side vary over all elements of the image of $\mathbb{Z}^k$ via A_Δ. We will denote this maximal difference by gap(Δ).

We are interested in what kinds of gaps can arise and from what types of hierarchical models. Sullivant constructed in [10] a family of tables on $n \geq 4$ binary random variables and a specified collection of models such that the gap(Δ) is $2^{n-3} - 1$. However, O'Shea later showed in [7] that Sullivant's gaps are rare in that they come from standard pairs that are small in the following sense. We can think of the size of a standard pairs (u, τ) as the dimension of the slice of the image of A that the image of (u, τ) is contained in. This dimension can be easily found as the rank of the submatrix of A obtained by removing the columns indexed by τ. If the dimension is equal to the rank of A then we say that (u, τ) is *wide*. The standard pair from the example at the end of the integer programming section is an example of a wide pair since the smallest subspace it lives in is dimension 1. If the dimension is less the the rank of A then it makes up a very small part of all feasible right hand sides. If we assume a uniform distribution of feasible vectors then the probability of encountering a vectors from (u, τ) becomes zero. There are even reasons to believe that a uniform distribution is in many cases be overly generous. See [7] for further discussion. In the following section we give our results about some gaps that are rare in the this sense as well as some examples of ones that are not.

4 Results

Our method and goal were straightforward: Implement the techniques described in the previous sections to find gap(Δ) and exhaustively compute it for all models with certain small dimensions. We coded our implementation in the `SageMath` mathematics software package [2] and used `4ti2`, which is a software package for algebraic and combinatorial problems, to find the Graver basis of the matrices [1] and used `Macaulay2` to find the irreducible components [4]. The full code can be found in the appendix.

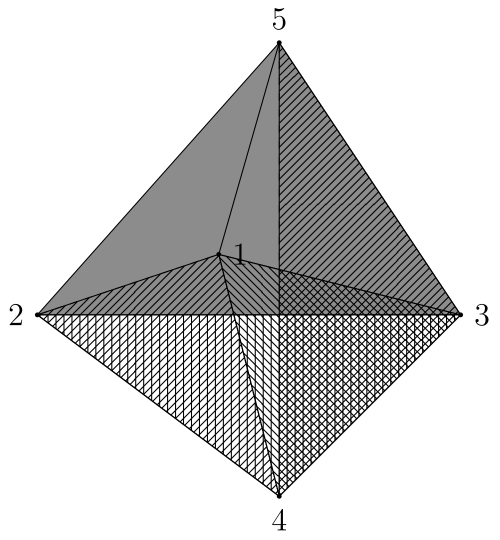
Proposition 4.1 *There does not exist a model* Δ *on* $\{1, 2, 3, 4\}$ *and a* $2 \times 2 \times 2 \times 2$ *table or a* $3 \times 2 \times 2 \times 2$ *table with margins* b *such that gap*$(A_\Delta, b) \geq 1$ *and* b *comes from a wide standard pair.*

This was shown by exhaustively computing gap(Δ) for all such models and comparing the necessary ranks to see if any of the standard pairs are wide. There are only two models up to relabeling for $2 \times 2 \times 2 \times 2$ tables that even have nonzero gaps. None of the components are both wide and have a gap greater than or equal to 1. The models are listed in the appendix along with the irreducible components where the nonzero gaps occur. There are more nonzero gaps for $3 \times 2 \times 2 \times 2$ tables but there are still no components that are both wide and have a gap greater than or equal to 1.

Proposition 4.2 *There do exist models* Δ *on* $\{1, 2, 3, 4, 5\}$ *and* $2 \times 2 \times 2 \times 2 \times 2$ *tables with margins* b *such that gap*$(A_\Delta, b) \geq 1$ *and* b *comes from a wide standard pair.*

We showed this by computationally finding examples where this occurs.

Example Consider the model Δ with facets $\{1, 2, 3, 5\}$, $\{3, 4, 5\}$, $\{1, 2, 4\}$, $\{1, 3, 4\}$, and $\{2, 3, 4\}$ for a margin of a $2 \times 2 \times 2 \times 2 \times 2$ table. We can visualize Δ as a simplicial complex in the following way:



Here the filled in area is the 3-dimensional face and the shaded areas are the four 2-dimensional faces. We can find A_Δ and it is given by

$$\begin{bmatrix}
1&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&1&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0\\
0&1&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&1&0&0&0&0&0&0&0&0&0&0&0&0&0&0\\
0&0&1&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&1&0&0&0&0&0&0&0&0&0&0&0&0&0\\
0&0&0&1&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&1&0&0&0&0&0&0&0&0&0&0&0&0\\
0&0&0&0&1&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&1&0&0&0&0&0&0&0&0&0&0&0\\
0&0&0&0&0&1&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&1&0&0&0&0&0&0&0&0&0&0\\
0&0&0&0&0&0&1&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&1&0&0&0&0&0&0&0&0&0\\
0&0&0&0&0&0&0&1&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&1&0&0&0&0&0&0&0&0\\
0&0&0&0&0&0&0&0&1&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&1&0&0&0&0&0&0&0\\
0&0&0&0&0&0&0&0&0&1&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&1&0&0&0&0&0&0\\
0&0&0&0&0&0&0&0&0&0&1&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&1&0&0&0&0&0\\
0&0&0&0&0&0&0&0&0&0&0&1&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&1&0&0&0&0\\
0&0&0&0&0&0&0&0&0&0&0&0&1&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&1&0&0&0\\
0&0&0&0&0&0&0&0&0&0&0&0&0&1&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&1&0&0\\
0&0&0&0&0&0&0&0&0&0&0&0&0&0&1&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&1&0\\
0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&1&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&1\\
1&0&1&0&1&0&1&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0\\
0&1&0&1&0&1&0&1&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0\\
0&0&0&0&0&0&0&0&1&0&1&0&1&0&1&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0\\
0&0&0&0&0&0&0&0&0&1&0&1&0&1&0&1&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0\\
0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&1&0&1&0&1&0&1&0&0&0&0&0&0&0&0&0\\
0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&1&0&1&0&1&0&1&0&0&0&0&0&0&0&0\\
0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&1&0&1&0&1&0&1&0\\
0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&1&0&1&0&1&0&1\\
1&1&0&0&0&0&0&0&1&1&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0\\
0&0&1&1&0&0&0&0&0&0&1&1&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0\\
0&0&0&0&1&1&0&0&0&0&0&0&1&1&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0\\
0&0&0&0&0&0&1&1&0&0&0&0&0&0&1&1&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0\\
0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&1&1&0&0&0&0&0&0&1&1&0&0&0&0&0&0\\
0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&1&1&0&0&0&0&0&0&1&1&0&0&0&0\\
0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&1&1&0&0&0&0&0&0&1&1&0&0\\
0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&1&1&0&0&0&0&0&0&1&1\\
1&1&0&0&1&1&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0\\
0&0&1&1&0&0&1&1&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0\\
0&0&0&0&0&0&0&0&1&1&0&0&1&1&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0\\
0&0&0&0&0&0&0&0&0&0&1&1&0&0&1&1&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0\\
0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&1&1&0&0&1&1&0&0&0&0&0&0&0&0&0&0\\
0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&1&1&0&0&1&1&0&0&0&0&0&0&0&0\\
0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&1&1&0&0&1&1&0&0\\
0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&1&1&0&0&1&1\\
1&1&1&1&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0\\
0&0&0&0&1&1&1&1&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0\\
0&0&0&0&0&0&0&0&1&1&1&1&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0\\
0&0&0&0&0&0&0&0&0&0&0&0&1&1&1&1&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0\\
0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&1&1&1&1&0&0&0&0&0&0&0&0&0&0&0&0\\
0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&1&1&1&1&0&0&0&0&0&0&0&0\\
0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&1&1&1&1&0&0&0&0\\
0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&1&1&1&1
\end{bmatrix}$$

which has 48 many rows and 32 many columns and rank 25. We can compute $M(A_\Delta, e_1)$ and find its irreducible components. There are 136 components and 8 of them have a gap equal to 1 and have standard pairs that are dimension 25. One of these components is $\langle x_3, x_5, x_{11}, x_{13}, x_{22}, x_{24}, x_{30}^2 \rangle$ which has a gap of 1. We can also compute $M(A_\Delta, -e_1)$ which also has 136 components and also has 8 with a gap greater than or equal to 1 and standard pairs that are dimension 25. ▲

It is important to note that we are not claiming the these examples have commonly occurring tables with large gap values. Remember that the gap of a component is the maximum of all possible gaps arising from right hand sides from the standard pair corresponding to the component. We make no assertion that all or even most of right hand sides from the standard pair have large gaps. These examples are just not rare in the sense of Sullivant's gaps.

We hope to pursue the question that the previous remark raises in future work. We would also like to have a complete classification of all models of margins from $2 \times 2 \times 2 \times 2 \times 2$ tables.

A $2 \times 2 \times 2 \times 2$ Examples

$\Delta = \{\{1,2,3\},\{1,4\},\{2,4\},\{3,4\}\}$					
$M(A_\Delta, e_1)$			$M(A_\Delta, -e_1)$		
Gap	Component	Wide	Gap	Component	Wide
0.5	$\langle x_7, x_9, x_{10}, x_{12}^2 \rangle$	Yes	0.5	$\langle x_1^2, x_2, x_4, x_{15} \rangle$	Yes
0.5	$\langle x_7, x_9, x_{10}^2, x_{12} \rangle$	Yes	0.5	$\langle x_1, x_2^2, x_4, x_{15} \rangle$	Yes
0.5	$\langle x_7, x_9, ^2 x_{10}, x_{12} \rangle$	Yes	0.5	$\langle x_1, x_2, x_4^2, x_{15} \rangle$	Yes
0.5	$\langle x_7^2, x_9, x_{10}, x_{12} \rangle$	Yes	0.5	$\langle x_1, x_2, x_4, x_{15}^2 \rangle$	Yes
1.0	$\langle x_0^2, x_3, x_5, x_6, x_{11}, x_{13}, x_{14} \rangle$	No	1.0	$\langle x_3, x_5, x_6, x_8^2, x_{11}, x_{13}, x_{14} \rangle$	No

$\Delta = \{\{1,2\},\{1,3\},\{1,4\},\{2,3\},\{2,4\},\{3,4\}\}$					
$M(A_\Delta, -e_1)$			$M(A_\Delta, e_1)$		
Gap	Component	Wide	Gap	Component	Wide
0.6667	$\langle x_1, x_2, x_4, x_8, x_{15}^3 \rangle$	Yes	1.0	$\langle x_0^2, x_3, x_5, x_6, x_7^2, x_9, x_{10}, x_{11}^2, x_{12}, x_{13}^2, x_{14}^2 \rangle$	No
0.6667	$\langle x_1, x_2, x_4, x_8^2, x_{15}^2 \rangle$	Yes	1.0	$\langle x_0^2, x_3^2, x_5^2, x_6, x_7, x_9^2, x_{10}, x_{11}, x_{12}, x_{13}, x_{14}^2 \rangle$	No
0.6667	$\langle x_1, x_2, x_4^2, x_8, x_{15}^2 \rangle$	Yes	1.0	$\langle x_0^2, x_3^2, x_5, x_6^2, x_7, x_9, x_{10}^2, x_{11}, x_{12}, x_{13}^2, x_{14} \rangle$	No
0.6667	$\langle x_1, x_2^2, x_4, x_8, x_{15}^2 \rangle$	Yes	1.0	$\langle x_0^2, x_3, x_5^2, x_6^2, x_7, x_9, x_{10}, x_{11}^2, x_{12}^2, x_{13}, x_{14} \rangle$	No
0.6667	$\langle x_1^2, x_2, x_4, x_8, x_{15}^2 \rangle$	Yes	1.0	$\langle x_0^2, x_3, x_5, x_6, x_7^2, x_9^2, x_{10}^2, x_{11}, x_{12}^2, x_{13}, x_{14} \rangle$	No
1.6667	$\langle x_1^2, x_2^2, x_3, x_4^2, x_5, x_6, x_7, x_8^2, x_9, x_{10}, x_{11}, x_{12}, x_{13}, x_{14}, x_{15}^2 \rangle$	No			
0.6667	$\langle x_1, x_2, x_4, x_8^3, x_{15} \rangle$	Yes			
0.6667	$\langle x_1, x_2, x_4^2, x_8^2, x_{15} \rangle$	Yes			
0.6667	$\langle x_1, x_2^2, x_4, x_8^2, x_{15} \rangle$	Yes			
0.6667	$\langle x_1^2, x_2, x_4, x_8^2, x_{15} \rangle$	Yes			
0.6667	$\langle x_1, x_2, x_4^3, x_8, x_{15} \rangle$	Yes			
0.6667	$\langle x_1, x_2^2, x_4^2, x_8, x_{15} \rangle$	Yes			
0.6667	$\langle x_1^2, x_2, x_4^2, x_8, x_{15} \rangle$	Yes			
0.6667	$\langle x_1, x_2^3, x_4, x_8, x_{15} \rangle$	Yes			
0.6667	$\langle x_1^2, x_2^2, x_4, x_8, x_{15} \rangle$	Yes			
0.6667	$\langle x_1^3, x_2, x_4, x_8, x_{15} \rangle$	Yes			

B Code

```
def makeMarginMatrix(C, d):
    p = len(d)*[1]
    for i in range(1,len(d)):
        p[i]= p[i-1]*d[i-1]
    rng = range(p[len(d)-1]*d[len(d)-1])
    index = [[floor(n/p[i])%d[i] for i in range(len(d))] for n in rng]
    M=[]
    for F in C:
        flat = [d[i]^int(i in F) for i in range(len(d))]
        k = len(d)*[1]
        for i in range(1,len(d)):
            k[i]= k[i-1]*flat[i-1]
        margins=[]
        for n in range(k[len(d)-1]*flat[len(d)-1]):
            margins.append([floor(n/k[i])%flat[i] for i in range(len(d))])
        for v in margins:
            M.append([int(all(index[n][i]==v[i] for i in F)) for n in rng ])
    return matrix(M)

def toBinomial(v, R):
    p = [max(i, 0) for i in v]
    n = [abs(min(i, 0)) for i in v]
    return R.monomial(*p) - R.monomial(*n)

def initFormsIdeal(I, R, w):
    IFI=[]
    for f in I.gens():
        exp = f.exponents()
        modexp = [[e[i]*w[i] for i in range(len(e))] for e in exp]
        if sum(modexp[0])>sum(modexp[1]):
            IFI.append(R.monomial(*exp[0]))
        else:
            if sum(modexp[0])<sum(modexp[1]):
                IFI.append(R.monomial(*exp[1]))
            else:
                IFI.append(f)
    return ideal(*IFI)

def makeHomogen(P, H, v):
    mons = v.monomials()
    if len(mons)==1:
        return v.coefficient(mons[0])* H.monomial(*list(mons[0].degrees())+
         P.ngens()*[0])
    else:
        return v.coefficient(mons[0])* H.monomial(*list(mons[0].degrees())+
          list(mons[1].degrees())) + v.coefficient(mons[1])*
          H.monomial(*list(mons[1].degrees())+list(mons[0].degrees()))

def  largestMonomailIdeal(I, P):
    n = P.ngens()
    H = PolynomialRing(QQ, 2*n, names = list(P.variable_names())
     + ['y'+str(i) for i in range(n)])
    homoIdGens = [makeHomogen(P, H, v) for v in list(I.gens())]

    newId = ideal(homoIdGens)
    f = H.monomial(*(2*n)*[1])
    newId.saturation(ideal(f))

    mons = []
    for v in newId.gens():
        if len(v.exponents())==1:
            mons.append(P(v))
    return ideal(*mons)

def irrDecomMono(I):
    return [macaulay2.ideal(f).to_sage() for f in
     macaulay2.irreducibleDecomposition(macaulay2.monomialIdeal(I))]

def makeIP(M,b,c):
    p = MixedIntegerLinearProgram(maximization=False, solver = "GLPK")
    w = p.new_variable(integer=True, nonnegative=True)
    for i in range(len(M.rows())):
        q = M.rows()[i]
        p.add_constraint(p.sum([w[n]*q[n] for n in range(len(q))]) == b[i])
    p.set_objective(p.sum([w[n]*c[n] for n in range(len(c))]))
    return p

def makeLP(M,b,c):
    p = MixedIntegerLinearProgram(maximization=False, solver = "GLPK")
    w = p.new_variable(integer=False, nonnegative=True)
    for i in range(len(M.rows())):
        q = M.rows()[i]
        p.add_constraint(p.sum([w[n]*q[n] for n in range(len(q))]) == b[i])
    p.set_objective(p.sum([w[n]*c[n] for n in range(len(c))]))
    return p

def auxiliaryLP(A,c,u0):
    u = [max(j-1,0) for j in u0]
    b = A*column_matrix([u])
    p = MixedIntegerLinearProgram(maximization=True, solver = "GLPK")
    w = p.new_variable(integer=False, nonnegative=False)
    for i in range(len(M.rows())):
        q = M.rows()[i]
        p.add_constraint(p.sum([w[n]*q[n] for n in range(len(q))]) == b[i][0])
    p.set_objective(list(matrix([c])*column_matrix([u]))[0][0]
     - p.sum([w[n]*c[n] for n in range(len(c))]))

    sup = []
    for i in range(len(u)):
        if u0[i]>0:
            sup.append(i)

    for i in range(len(u)):
        if i in sup:
            p.set_min(w[i], 0)

    return p

def monIdealVector(I, R):
    g = I.gens()
    explist = []
    for f in g:
        explist.extend(f.exponents())
    u0 = [sum([v[i] for v in explist]) for i in range(R.ngens())]
    return u0

def maxGap(A, c):
    from sage.interfaces.four_ti_2 import four_ti_2
    G = four_ti_2.graver(mat=A)
    if G.nrows()==0:
        return 0
    n = A.ncols()
    P = PolynomialRing(QQ, n, names = ['x'+str(i) for i in range(n)])
    I = ideal(*[toBinomial(r, P) for r in G.rows()])

    IFI = initFormsIdeal(I, P, c)

    BMI = largestMonomailIdeal(IFI, P)

    ID = irrDecomMono(BMI)

    return max([auxiliaryLP(A,c,monIdealVector(ic, P)).solve() for ic in ID])

def SpernerFamilies(n):
    A = [tuple(i) for i in powerset(range(n))]
    from sage.combinat.posets.posets import FinitePoset
    B = []
    for i in A:
        for j in A:
            if set(i).issubset(set(j)):
                B.append([i,j])
    g = DiGraph()
    g.add_vertices(A)
    g.add_edges(B)
    P = FinitePoset(g)
    IsoList = []
    for S in P.antichains_iterator():
        if all(not IncidenceStructure(S).is_isomorphic(K) for K in IsoList):
     IsoList.append(IncidenceStructure(S))
    sFamilies = [i.blocks() for i in IsoList]
    return sFamilies

def fat(M, l):
    s = [sign(i) for i in l]
    Mt = M.transpose()
    cols = Mt.rows()
    subcols=[]
    for n in range(len(l)):
        if s[n] == 0:
            subcols.append(cols[n])
    SM = column_matrix(subcols)
    return M.rank() <= SM.rank()

def gapData(A):
    n = A.ncols()
    c = n*[0]
    c[0] = 1
    e = n*[0]
    e[0] = -1
    gap = 0
    from sage.interfaces.four_ti_2 import four_ti_2
    G = four_ti_2.graver(mat=A)
    if G.nrows()==0:
        return gap
    P = PolynomialRing(QQ, n, names = ['x'+str(i) for i in range(n)])
    I = ideal(*[toBinomial(r, P) for r in G.rows()])

    IFIc = initFormsIdeal(I, P, c)
    IFIe = initFormsIdeal(I, P, e)

    BMIc = largestMonomailIdeal(IFIc, P)
    BMIe = largestMonomailIdeal(IFIe, P)

    IDc = irrDecomMono(BMIc)
    IDe = irrDecomMono(BMIe)
    for ic in IDc:
        k = auxiliaryLP(A,c,monIdealVector(ic, P)).solve()
        if k>gap:
            gap = k
        if k > 0 :# and fat(A, monIdealVector(ic, P)):
            print "+ gap:", k,
            print "i.c:",
            for g in ic.gens():
                print g,
            print fat(A, monIdealVector(ic, P))

    for ic in IDe:
        k = auxiliaryLP(A,e,monIdealVector(ic, P)).solve()
        if k>gap:
            gap = k
        if k > 0 :# and fat(A, monIdealVector(ic, P)):
            print "- gap:", k,
            print "i.c:",
            for g in ic.gens():
                print g,
            print fat(A, monIdealVector(ic, P))
    return gap



#Main

n = 4
d = [2,2,2,2]
count = 0
SF = SpernerFamilies(n)
for i in range(len(SF)):
S = SF[i]
Y = SimplicialComplex(S)
M = makeMarginMatrix(S, d)
     if M.nrows()>0:
         g = gapData(M)
         if g>= 1:
             print count, S
             print
    count = count +1
print "Finished."
```

References

[1] 4ti2 team. 4ti2—a software package for algebraic, geometric and combinatorial problems on linear spaces.

[2] The Sage Developers. *SageMath, the Sage Mathematics Software System (Version 6.9)*, 2016. `http://www.sagemath.org`.

[3] Adrian Dobra and Stephen E. Fienberg. Bounds for cell entries in contingency tables given marginal totals and decomposable graphs. *Proc. Natl. Acad. Sci. USA*, 97(22):11885–11892 (electronic), 2000.

[4] Daniel R. Grayson and Michael E. Stillman. Macaulay2, a software system for research in algebraic geometry.

[5] Serkan Hoşten and Bernd Sturmfels. Computing the integer programming gap. *Combinatorica*, 27(3):367–382, 2007.

[6] Serkan Hoşten and Rekha R. Thomas. Standard pairs and group relaxations in integer programming. *J. Pure Appl. Algebra*, 139(1-3):133–157, 1999. Effective methods in algebraic geometry (Saint-Malo, 1998).

[7] Edwin O'Shea. On the occurrence of large gaps in small contingency tables. 2009.

[8] Mutsumi Saito, Bernd Sturmfels, and Nobuki Takayama. *Gröbner deformations of hypergeometric differential equations*, volume 6 of *Algorithms and Computation in Mathematics*. Springer-Verlag, Berlin, 2000.

[9] Alexander Schrijver. *Theory of linear and integer programming*. Wiley-Interscience Series in Discrete Mathematics. John Wiley & Sons, Ltd., Chichester, 1986. A Wiley-Interscience Publication.

[10] Seth Sullivant. Small contingency tables with large gaps. *SIAM J. Discrete Math.*, 18(4):787–793 (electronic), 2005.