Correlations in C3: Block Codes & Paradoxes

Keng August 2014

This is a summary of what we have so far, in regards to the generation of classes of blockcode and, when assigned with a causal structure (C3), the identification of classical paradoxes in these classes. (Skip to pg.2 for the algorithm.)

I. Consider $p-$parties, $n-$nary outcomes, with a list of k allowable outcomes, where $k \leq n^p$. This list can be written as a block code of $p-$columns and $k-$rows. Allowing for the three operations of relabeling parties, flipping(permuting) the $n-$nary outcomes, and reordering the outcome list, the problem of classifying equivalent outcomes is the same as classifying block codes. Obviously the block codes are list of outcomes totally independent of causal structures.

Matt's finding on the sequence of the number of classes of block code with the same size led to the paper *Enumeration, construction and random generation of block codes* by Harald Fripertinger. The paper gives these sequences for block codes up to any $p-$columns(parties), $k-$columns(outcomes) and $n-$nary alphabets(outcomes). The representation of classes can be generated at random by the *Dixon-Wilf Algorithm.*

In his reply, the author mentioned that these number sequences were algebraically computed, but there is currently no list of all the enumerated objects. He did mentioned that it is possible to compute the classes by using combinatorics under group actions for small block codes, but we would have to figure it out by ourselves. Maybe it could be done along the line of Burnsides lemma/orbits of group.

I'm not sure how useful/vital these results are since it is impossible to check all of them, but here's it anyway. The numbers of classes with $k = 0, 1, 2, 3, 4, \ldots$ outcomes are 1, 1, 3, 10, 34, 105, 321, 846, 1984 ... for $3-$nary outcomes, and 1, 1, 3, 10, 55, 254, 1643, 10164, 63488 ... for $4-$nary outcomes.

II. *The draft of my cube-rule algorithm for identifying paradoxes:*

There are three methods for identifying classical paradoxes as we currently know of: Rob's probabilistic argument, Tobias' diagrammatric argument, and my cube-ruletable argument. Apparently they are very similar (perhaps not equivalent); they just get more pictorial as we progress from Rob's to mine.

I tried to generalize my method to for $n-$nary block codes. The main idea is to reduce the problem to smaller similar sub-problems. However, I haven't been able to prove the completeness of the generalization, i.e. it can always identify a paradox, or find a classical model otherwise. It builds on some strong intuition and experience from experimenting with it.

There are two sides of the paradox to check here. Although for binary outcomes checking for either one is sufficient, for general $n-nary$ outcomes both must be checked for the sake of completeness. This is some sort of "sandwiching" procedure. The two sides of the paradox are:

(a) The *Too-Few* paradox: either we assume some of the outcomes in the class, and show that at least one of them cannot occur, or

(b) The *Too-Many* paradox: we show that by assuming all the outcomes in the given class, one must obtain some other additional outcomes.

(c) If none of them occur, then there exists a classical model.

Matt has an interesting comment regarding (b), it goes something like: "If we start out by assuming all the outcomes in the block code must occur with positive probabilities, then maybe we can try to derive some inequality to show that the probability of some additional, unwanted outcomes must be positive too."

Given a $n-$nary, $p-$columns(p=3 for us) and $k-$rows block code associated to a $C3$ causal structure, there are 2 cases to this algorithm:

1. *The basic case:* A party has one outcome value that occurs only once. Call this the "singlet" case.
2. *The extended case:* All parties have repeated outcome values (the non "singlet" case).

In each case there may or may not be a paradox; the algorithm shall identify it if there is one, or construct a classical model if there is none. Incidentally, for binary outcomes *Case* 2 does not yield a paradox. Also, *Case* 1 does not happen to the binary classes with more than 4 outcomes (because each party has repeat either 0 or 1 more than once). It seems related to the $n-$nary, as intuitively suggested by the algorithm I will now give.

The Algorithm (more like sketch of idea)

For a $n-$nary, $3-$columns and $k-$rows block code with a $C3$ causal structure, denote the parties A, B, C, and the sources $S_{AB}(q)$ etc. where the subscript AB is the target parties, and (q) is the label on the subset source of S_{AB} that yields a specific outcome ab. S_{AB} is the complete source set.

The method starts by picking a set of three sources that would result in some outcome set larger than the block code. This is always possible because at worst we can trivially pick the sources that result in all the n^p outcomes. Then, it tries to find: if the possibility of some subset of outcomes would ensure the impossibility of some other subset of outcomes: the *Too-Few* paradox; or else if some excessive outcomes not in the block code must happen: the *Too-Many* paradox; else it gives a classical model that is "sandwiched" nicely between the paradoxes.

Let a $n-$nary, $3-by-k$ block code be given; let a set of sources that yield an outcome set larger than this block code be given. Denote each individual outcomes as value $\in \mathcal{M} = \{1, 2, \ldots n\}$ (the binary case is when $n = 2$).

Case 1: the singlet case

(a) Call the *singlet* party Alice. So, A has an outcome $a = a_1 \in \mathcal{M}$ which occurs only once. Call the responsible subsets of sources $S_{AB}(1),\ S_{AC}(1)$.

For this instance, denote the outcomes of B, C respectively as $b = b_1,\ c = c_1 \in \mathcal{M}$, thus this outcome is $\{a_1b_1c_1\}$. It is easy to see that a_1 being singlet fixes b_1, c_1, so any outcomes b, c must be independent of S_{BC}. To recap, the responsible sources for singlet outcome $a_1b_1c_1$ are $S_{AB}(1),\ S_{AC}(1),\ S_{BC}$.

(b) Next, look elsewhere in the block code where $a \neq a_1$. One of the two ways is to change a source to one of its complements: $S_{AC}(2) \in (S_{AC}(1))^C$, so that $b = b_1$ still, while we get some a, c which may or may not be a_1, c_1. The paradox may arise now:

Scan the other rows.

i. If there is no b_1 anywhere else in the block code, i.e. b_1 is also a singlet, then a, c must be independent of S_{AC}. Thus, S_{AC}, S_{BC} completely determine $c = c_1$ as the only outcome for C.
If $c \neq c_1$ occurs anywhere else in the block code, then this is a contradiction, i.e. a paradox. Else, there exist a partial classical model for it.

ii. If b_1 occurs elsewhere with some $a \neq a_1$, call these a's $a_2, a_3, \ldots, a_{<k} \in \{a_i : 1 < i < k\}$. For each a_i, call $\{c_i\}$ the set of c's that occur with a_ib_1. Thus all the possible outcomes where $b = b_1$ is the set $\{a_ib_1\{c_i\} : 1 < i < k\}$, and the union of the sources that result in this set is $(S_{AC}(1))^C$.

*I use $\{a_ib_1\{c_i\} : 1 < i < k\}$ for $\bigcup_i a_ib_1\{c_i\}$

Now consider the union $a_1b_1c_1 \cup \{a_ib_1\{c_i\} : 1 < i < k\}$. The sources that result in this set are $S_{AC}(1) \cup (S_{AC}(1))^C = S_{AC}$ and S_{BC}. They determine the complete outcome set for C, which is $(c_1 \cup \{c_i : 1 < i < k\}) \subseteq \mathcal{M}$.
If there is $c \in (c_1 \cup \{c_i : 1 < i < k\})^C$ which occurs elsewhere in the block code, then this is a paradox. Else, there exists a partial classical model for it.

(c) If there is no paradox, proceed to repeat step (b) for S_{AB}, the other source of A. The same arguments follow by interchanging the roles of b, c: "change the other source to one of its complements: $S_{AB}(2) \in (S_{AB}(1))^C$, so that $c = c_1$ still, while we get some a, b which may or may not be $a_1, b_1 \ldots$"

Then, we would get $a_1b_1c_1 \cup \{a_j\{b_j\}c_1 : 1 < j < k\}$ with sources $S_{AB}(1) \cup (S_{AB}(1))^C = S_{AB}$ and S_{BC}, which determines the complete outcome set for B, $(b_1 \cup \{b_j : 1 < j < k\}) \subseteq \mathcal{M}$.

If there is $b \in (b_1 \cup \{b_j : 1 < j < k\})^C$ which occurs elsewhere in the block code, then this is a paradox. Else, there exists a partial classical model for it.

(d) If there is still no paradox, we proceed to search for one by looking at A. Previously we showed that the complete set of sources S_{AB}, S_{BC}, S_{AC} determines the complete outcome sets $\{b_j\}, \{c_i\}$ for B, C in the block code. The complete sources S_{AB}, S_{AC} would also yield the complete outcome set for A, which is $\bigcup_i a_i \cup \bigcup_j a_j$ from above.

Now, if there is $a \in (\bigcup_i a_i \cup \bigcup_j a_j)^C$ which occurs elsewhere in the block code, then this is a paradox. Or else, we obtain the complete indepdent sets of individual outcomes for A, B, C; denote these sets $\bigcup_h \{a_h\}, \bigcup_j \{b_j\}, \bigcup_i \{c_i\}$.

Any paradoxes produced in the steps above is the *Too Few* paradox. If none occurs, we check for the *Too-Many* paradox next.

(e) Now we proceed to the final step. We started with the assumption that the complete set of sources S_{AB}, S_{BC}, S_{AC} yields an outcome set larger than the given block code; and so far there is no *Too-Few* paradox, thus there is no lack of outcomes. However, we may get more outcomes than we want.

Since the steps $(b), (c)$ are independent, the complete set of outcomes by taking the tensor product among elements of individual outcome sets $\bigcup_h \{a_h\}, \bigcup_j \{b_j\}, \bigcup_i \{c_i\}$ can easily be larger than the given block code.

Therefore, the last step is to take the tensor product among $\bigcup_h \{a_h\}, \bigcup_j \{b_j\}, \bigcup_i \{c_i\}$ to get the set $\{a_h b_j c_i : h, i, j < k\}$. If the generated set is bigger than the block code, and there must exists some outcomes not in the block code, and this is a *Too-Many* paradox.

Or else, there is no *Too-Few* and no *Too-Many* paradox. This implies that the generated outcome set coincides perfectly with the block code, thus there exists a classical model. The algorithm for the *singlet* case end here.

Case 2: the extended case

It would take me longer to rigorously formulate this, but it's an extension/modification of the *singlet* case algorithm.

The key idea is to consider the changes needed:

$$a_1 \text{ occurs once} \mapsto a_1 \text{ occurs multiple times,}$$
$$S_{BC} \mapsto S_{BC}(1), S_{BC}(2), \ldots, S_{BC}(q)$$

The multiple occurences of a_1 requires us to distribute S_{BC} to several pairs of $a_1 bc$, e.g. $S_{BC}(1) \to a_1 b_1 c_1$, $S_{BC}(2) \to a_1 b_2 c_2 \cdots$ etc. This would give us many cases to consider. However, to find paradoxes, there are some ways of reducing redundancy, such as using the same instances of S_{AB}, S_{AC} for all of the $S_{BC}(i)$ above. The intuition comes from from the "wall, filter" concepts of the cube-ruletable.

To be continue, maybe...

Let's start. I tried to do it in the greatest generality, i.e. the source sets can be $\mathbb{R}$ intervals. The rule-table would then be replaced by a map

$$\mathbb{R}_i \times \mathbb{R}_j \mapsto \mathcal{M}$$

where subscripts are source labelings, and $\mathcal{M} = \{1, 2, 3, \ldots, n\}$ is the finite list of possible individual outcomes.

It is easy to see that the singlet case has generality equivalent to this. However, the arguments breakdown for the non-singlet case; the problem arises because the rectangle interval $\mathbb{R}_i \times \mathbb{R}_j$ can be divided into finite partitions, where each partition can have infinitely many disjoint subsets (like how one can partition the reals into rationals and irrationals).

I talked to Matt about the importance of this assumption. It shouldn't be a big concern since we have been implicitly assuming the sources and outcomes being finite, countable sets. Dennis' result on the completeness of the classical set using n=7-nary sources justifies the assumption too.

Therefore I shall proceed with the assumption that the sources and outcomes are both finite countable sets.

Case 1: the non-singlet case

(a) In the given block code with the sources that yield a possibly larger outcome set, assume that there is no singlet case. Therefore, each outcome value on each party must occur at least once.

Pick one of the party outcome values that occurs the least times. Call this value $a = a_1$ of Alice. Then, there is $t-$instances of a_1, where $2 \leq t \leq n^{p-1}$, each occuring with some different pair of bc. Call these instances

$$a_1b_1c_1,\ a_1b_2c_2,\ a_1b_3c_3,\ \ldots,\ a_1b_tc_t$$

and the corresponding sources $S_{AB}(i), S_{BC}(i), S_{AC}(i) : 1 \leq i \leq t$.

(b) First, consider the outcome $a_1b_1c_1$ with sources $S_{AB}(1), S_{BC}(1), S_{AC}(1)$

1. Prove that sources independence is redundant: i.e. shd have just one big $\{S_1\} \times \{S_2\}$ map listing. This is essentially the essence of the singlet case: singlet-property of the sources, for any number of resultant outcomes.

If there is some independence of any sources, then it's like forming many parallel walls extending from b (cores arg for c): same arg as before. Proof of it?? Arg shd be similar, just try to reduce to sub-singlet cases. and merge.

Thus have to consider the stacking of walls since the previous case is stupid. (so vary only SBC)

note to self: Quantum version of the arg.

Solving the problem of matrix classification

Keng August 2014

Fripertinger '96 discusses the enumeration of block codes of size $p-$columns, $k-$rows and $n-$nary(alphabet). The paper gives a tabulation of the number of the classes for each block code size given these three parameters.

In our recent conversation, Fripertinger mentioned his results on the explicit representation of each of the classes using vector coefficients. For a class of block code, each row is a vector from the base n space $\{0, 1, \ldots, n, n+1, \ldots, n+n, \ldots n^p - 1\}$. The block code is represented as a list of vectors in the space which occurs in it.

e.g. base 2, 3 parties (columns), 3-outcomes (rows):

$$\{000, 001, 010, 011, 100, 101, 110, 111\}$$

$$\textit{class:}\ \{000, 001, 010, 111\} = \{1, 2, 3, 8\}$$

Call this the (vector) coefficient representation. Fripertinger has the collection of these coefficients for classes up to $n = 7$.

A side problem that comes up in the study of classical paradoxes is the testing for the class membership of a block code, or, equivalently, the classification of matrices. This by itself is a more general mathematical problem that has not been solved. A more detailed illustration of the problem is:

Given two matrices of equal size and $n-$nary entries, determine whether or not they are equivalent, i.e. they belong to the same class, under the following transformations/operations:

1. column-swapping
2. row-swapping
3. letter permutation

An easier sub-problem is when we disregard the third operation.

Algorithm to classify block codes/matrices

Now I present an algorithm to solve the general problem as well as the sub-problem of matrix classification under the aforesaid operations. The algorithm will arrange a matrix into the *bundle form*, which I shall illustrate later. To justify the algorithm, I will also prove that every class has a different *bundle form*.

The *bundle form* is a canonical form of matrices under the operations described earlier. The bundling of a matrix is done in recursive manner, with these properties:

1. A column has bundles. Each of these bundles is a collection of the rows in the same column that have the same entry value. Down the column, the size of the bundles gets smaller.

2. Fix a column. For each bundle, this range of rows and all the columns on the right is a sub-matrix. Property 1 holds true for this sub-matrix, and it holds true recursively for all the bundles, and the bundles in the sub-matrices spanned by them.

3. This results in a partitioning/refinement of a matrix recursively to the right until it cannot be refine anymore, i.e. the final column's bundle sizes are one. Note that no sub-bundle can be bigger than its super-bundle.

That is the description of the *bundle form*.

Proof. It is sufficient to show that any two distinct classes have different bundled forms; this is obvious. Take two classes and transform them into their *canonical bundled form*. Since the three operations/transformations does not change the class representation, these two classes remain invariant. Since they are different, the corresponding bundled forms must be different too. Therefore the mapping from classes to bundles forms is injective (although obviously not surjective). □

note: Want each class to have a unique bundle form, i.e. class and bundle form are bijective. Unique bundle form: big to small: down each column, and right per adjacent columns. Place by the biggest you can get next. But is it going to be the same as Fripertinger's form?

Proof should be easy: diff class under ops shd have diff form. that's okay.

can a same class have many forms? if the bundle form is guaranteed to be unique (well ordered). Then bundled form = class, not bijective. but it's alright.

Retrieve the algorithm from chalk pics. Define sameness bundles.

Sketch of bundled-form algorithm

Firstly, it would be tremendously helpful to have a useful notation. A block code is given with three parameters: $n-$nary entries, $p-$columns, $k-$rows where k

1. It'd be tremendously useful to have some descriptive notation. Again, we are given a block code of size p k n