The 23rd ECMI Modelling Week
European Student Workshop
on Mathematical Modelling
in Industry and Commerce

Wrocław University of Technology, Wrocław, Poland, August 23-30, 2009

Report of project group 11 on

Improving the Efficiency of Allergy Tests

E C M I

The 23rd ECMI MODELLING WEEK
European Student Workshop on Mathematical Modeling in Industry and Commerce

Wrocław University of Technology
23rd - 30th August 2009

PROJECT TOPIC: **Improving the Efficiency of Prick-tests**

Blessing Uzor
Nathaniel Egwu
Peter Romeo Nyarko
Henrik Alsing Pedersen
Stephen Edward Moore
Ekeoma Rowland Ijioma
Jakub Tomczyk

Project supervisor: Christian Zschalig
Dresden, Germany

October 31, 2009

Contents

List of Figures

List of Tables

Abstract

Prick tests are common allergy tests. This research work is aimed at improving the efficiency of those tests by reducing the number of pricks made on a patients skin by applying mixtures of the considered allergenes. A model is formulated and Boolean lattice homomorphism is used to describe the relationship between the allergenes and the pricks after which the model is analyzed by scientific computation using Java. However, impressive solutions were obtained for $n = 14$ allergenes and $q = 10$ mixtures.

1 Introduction

Allergy tests are utilized to detect allergic reactions to certain substances (allergenes). A well known standard procedure is the prick-test (or scratch-test). Potential allergy inducing substances are applied onto the skin which is then pricked slightly to allow the allergenes to enter the upper dermis. After some time (approx. 20 min), the allergic reaction can be evaluated by means of the degree of redness and the size of the wheals of the affected region. The skin is pricked once for each allergene in this procedure, i.e. for n substances n pricks are needed.

In order to design a more elaborate treatment with fewer pricks, one can apply several substances, each consisting of a mix of allergens. The test will then have a positive result if the patient is allergic to at least one of its components. Actually, one has to regard potential reactions among the ingredients impacting the result, i.e. the reaction of the upper dermis. In our model we will exclude the consideration of these dependencies.

The aim is to develop a method to find the substances a patient is allergic to with as few pricks as possible. More precisely, $q < n$ compounds of allergens shall be analyzed in terms of their skin irritations. The reaction pattern may allow to detect the involved allergenes unambiguously. Obviously, one cannot expect to get a correct result for every combination of allergic substances since the number 2^q of possible outputs is less than the number 2^n of possible allergic reactions. However, the exact answer is important only for a small number $k < n$ of allergenes in many cases. If the number of allergic substances exceeds k then a poly-allergy is diagnosed which is treated in a different way less fitted to the included allergenes. We assume the number k to be about 5. The number of tests q is supposed to be as small as possible, but will increase while k is growing.

2 Mathematical Model of the Problem

2.1 Formulation of the Problem

We are given a set A of allergenes (i.e chemicals, foods, medicines, mold, plants, and pollen etc.) which depends on the type of test in question. This is so since we have different types of tests for different allergic reaction. Each prick can be modelled as a map $P_X : \mathcal{P}(A) \rightarrow \{0, 1\}$ with $X \in \mathcal{P}(A)$, where $\mathcal{P}(A)$ is the set of all possible subsets of allergenes. The function P_X defined by

$$P_X : y \mapsto \begin{cases} 1, & X \cap y \neq \emptyset \\ 0, & X \cap y = \emptyset \end{cases} \tag{1}$$

assigns a "yes" if a set of allergenes y contains a substance of the ones of X a patient is allergic to, otherwise it returns a "No".

Our aim is to find a set $Y = (y_1, \cdots, y_q) \subseteq \mathcal{P}(A)$ of size q containing mixtures of allergenes, such that the function $t_Y : \mathcal{P}(A) \rightarrow \{0, 1\}^q$, defined by

$$t_Y : X \mapsto (P_X(y_1), \cdots, P_X(y_q)) \tag{2}$$

called the prick-test-function, is *injective* for all X of cardinality at most k. More precisely, if $t_Y(X_1) = t_Y(X_2)$ holds, then $|X_1| > k$ and $|X_2| > k$, and we will call Y k-recognizable and t_Y k-injective. We will restrict to those Y that contain all elements of A (i.e for all $x \in A$ there exist $y \in Y : x \in y$) since otherwise $t_Y(x) = P(\emptyset) = (0, ..., 0)$, i.e. t_Y would be not even 0-injective. For an illustration of the k-injectivity have a look at Figure 1. In particular, the case depicted in the middle is not allowed. The prick-test-function assigns

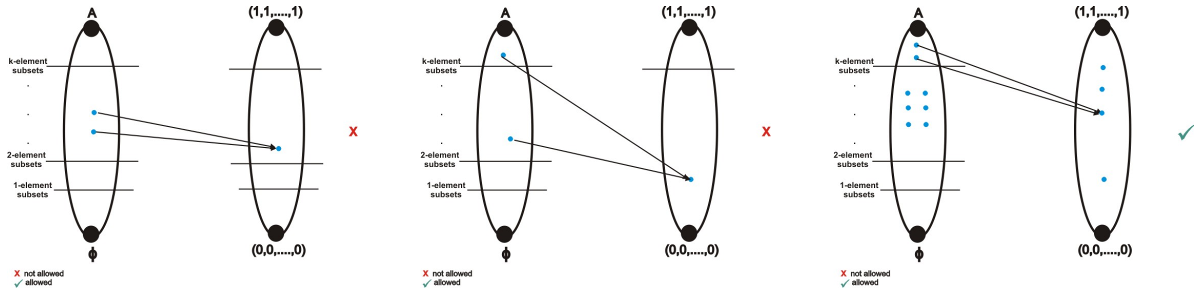


Figure 1: Illustration of the k-injectivity.

to the patients allergies X a tuple of zeros and ones, i.e. a pattern of the patients allergic reactions. Obviously, this function is composed of the q prick functions. In particular, we conclude that our test result is conclusive if we could arrive at exactly one possible conclusion of the allergic substances of the patient from the results of the test.

Furthermore, we mentioned that the prick-test-function shall be injective. Intuitively, this means that we want to unambiguously find out a patients allergies, if their number does not exceed k. Otherwise we will be able only to state that this number is greater than k without specifying the actual allergies.

2.2 Some basics of lattice theory

We will see in the next subsection that the function t_Y is a mapping between Boolean lattices. Hence, we want to recall here some basic notations regarding *lattices*. All content presented here is well known and can be found in lattice theory monographs like [1].

There are two equivalent ways to declare a lattice, namely the *algebraic* and the *order theoretic* definition:

Definition 2.1 (algebraic definition) *An algebra $(L, \wedge, \vee)$ consisting of a set L and two binary relations $\vee$ ("join") and $\wedge$ ("meet") is called a* lattice *if $\vee$ and $\wedge$ are commutative, associative and fulfill the* absorption rules

$$\forall u, v \in L : u \wedge (u \vee v) = u = u \vee (u \wedge v).$$

An *upper bound* of a subset $X \subseteq P$ of an ordered set $(P, \leq)$ is an element y satisfying $x \leq y$ for all elements $x \in X$. Dually a lower bound is an element z with $z \leq x$ for all $x \in X$. We call $y = \bigvee X$ *least upper bound* if it is lesser or equal to every upper bound y' of X. Dually $z = \bigwedge X$ is a *greatest lower bound* of X if it is greater or equal to every lower bound z' of X. The order theoretic definition of a lattice is based on the existence of those bounds.

Definition 2.2 (order theoretic definition) *An ordered set $(L, \leq)$ is called a lattice if for every pair $u, v \in L$, the least upper bound $u \vee v$ (called* join*) and the greatest lower bound $u \wedge v$ (called* meet*) exist.*

Indeed, both definitions are equivalent: It is straightforward to see that by

$$\forall u, v \in L : u \leq v \iff u \wedge v = u \;(\iff u \vee v = v)$$

both formulations can be transferred into another.

A special class of lattices are the *Boolean lattices*. The main addition to the usual join and meet operations is a further unary operation called the *complement* that assigns to each lattice element its "negation" or "opposition":

Definition 2.3 *An algebra $(L, \wedge, \vee, {}^{-}, 0, 1)$ consisting of a distributive lattice $(L, \wedge, \vee)$, a unary relation ${}^{-}$ and two constants 0 and 1 satisfying:*

$$\begin{aligned} &\forall a \in L : a \wedge \overline{a} = 0, \qquad \forall a \in L : a \vee \overline{a} = 1 \text{ and} \\ &\forall a \in L : \overline{\overline{a}} = a. \end{aligned}$$

is called a Boolean lattice*.*

Remark 2.1 1. Algebras of sets *are prominent representatives of Boolean lattices; the* representation theorem of Stone *even states that every Boolean lattice is isomorphic to such an algebra of sets. In particular, the set $\mathcal{P}(M)$ of all subsets of a set M forms a Boolean algebra. More precisely, the algebra is given by $(\mathcal{P}(M), \cup, \cap, \overline{}, \emptyset, M)$, where $\cup$ and $\cap$ are the usual union and intersection operations and $\overline{}$ is the set complement. The order relation associated to such a Boolean lattice is the usual set inclusion $\subseteq$.*

2. *Another example of Boolean lattices is based on the set $\{0, 1\}^q$ of all q-tuples having only the entries 0 or 1. It is easy to see that*

$$B_q := (\{0, 1\}^q, \vee_{B_q}, \wedge_{B_q}, \overline{}_{B_q}, 0_{B_q}, 1_{B_q})$$

fulfills all required constraints, if we define for all $a, b \in \{0, 1\}^q$ with $a := (a_1, \ldots, a_q)$ and $b := (b_1, \ldots, b_q)$:

$$\begin{aligned} a \vee_{B_q} b &:= (a_1 \vee b_1, \ldots, a_q \vee b_q) \\ a \wedge_{B_q} b &:= (a_1 \wedge b_1, \ldots, a_q \wedge b_q) \\ \overline{a}_{B_q} &:= (\neg a_1, \ldots, \neg a_q) \\ 0 &:= (0, \ldots, 0) \\ 1 &:= (1, \ldots, 1) \end{aligned}$$

(Do not be confused by another occurrence of the symbols $\vee$, $\wedge$. Without index they indicate the common conjunction and disjunction operators used in predicate logic, while $\neg$ is the negation operator in this context.) In this setting, the order relation $\leq_{B_q}$ in the lattice is given by

$$a \leq_{B_q} b :\iff \forall i \in \{1, \ldots, q\} : a_i \leq b_i,$$

i.e. an element is smaller or equal than another, if it is smaller or equal in each component.

In a lattice or more general an ordered set $(P, \leq)$ two elements $u, v \in P$ are either *comparable*, i.e. $u \leq v$ or $v \leq u$ holds, or they are *incomparable*. This we will denote by $u \parallel v$.

Finally we recall *homomorphisms* on lattices. Due to the twofold approach to this structures, there are also two concepts of a homomorphism:

Definition 2.4 *A mapping $\varphi : L \to M$ between the lattices $(L, \leq_L)$ and $(M, \leq_M)$ is an* order homomorphism *if the following equivalence holds:*

$$\forall u, v \in L : u \leq_L v \iff \varphi(u) \leq_M \varphi(v).$$

Definition 2.5 *A mapping $\varphi : L \to M$ between the lattices $(L, \vee_L, \wedge_L)$ and $(M, \vee_M, \wedge_M)$ is a* lattice homomorphism *if the following equations hold:*

$$\forall u, v \in L : \varphi(u \vee_L v) = \varphi(u) \vee_M \varphi(v) \text{ and } \varphi(u \wedge_L v) = \varphi(u) \wedge_M \varphi(v)$$

A simple argumentation shows that every lattice homomorphism is an order homomorphism, too. The opposite is not true: There exist order homomorphisms between lattices not being lattice homomorphisms.

2.3 Properties of the Test Function t_Y

2.3.1 t_Y as a Mapping Between Boolean Lattices

We notice with Remark 2.1 that the function t_Y defines a map from the Boolean lattice $B_{|A|}$ into the Boolean lattice B_q. This is illustrated in the lattice diagram in Figure 1.

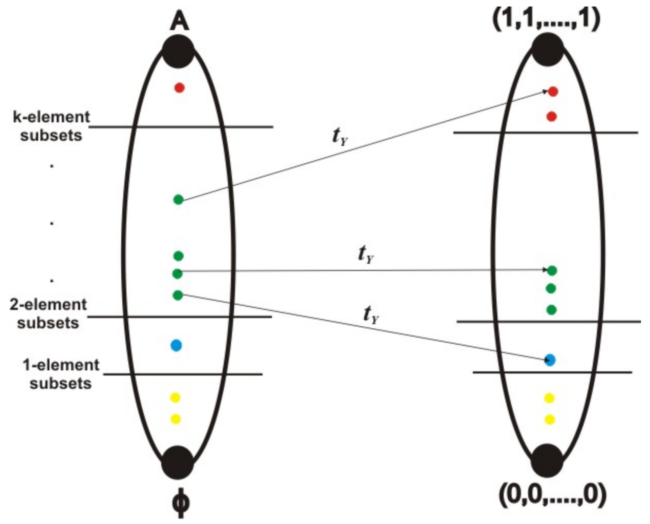


Figure 2: Lattice representations of $B_{|A|}$ and B_q

In the diagram in Figure 2, we have used different colors to represent the notion of partial order in the sets. There exist greatest elements A and $(1, 1, ..., 1)$ as well as a least elements $\emptyset$ and $(0, 0, ..., 0)$ of the sets. Now we want to prove that t_Y is even an order homomorphism and a $\vee$-homomorphism, two facts that will be very helpful for the algorithmic part. Thereby, t_Y is an order homomorphism i.e. an order-preserving map from the partially ordered set $\mathcal{P}(A)$ into the partially ordered set $\{0, 1\}^q$ if

$$\forall X_1, X_2 \in \mathcal{P}(A) : X_1 \geq X_2 \implies t_Y(X_1) \geq t_Y(X_2).$$

Proposition 2.2 *Let $\mathcal{P}(A)$ be a Boolean lattice and let $B_q = \{0,1\}^q$ be another Boolean lattice defined as in Remark 2.1. Then the map $t_Y : \mathcal{P}(A) \to Q$ as defined in Equation 2 is an order homomorphism.*

Proof:

$$X_1 \subseteq X_2 \overset{(*)}{\Longleftrightarrow}, \forall y \in Y : X_1 \cap y \subseteq X_2 \cap y \forall y \in Y : P_{X_1}(y) \leq P_{X_2}(y) \Leftrightarrow: t_Y(X_1) \leq t_Y(X_2)$$

where (*) holds since for all $m \in X_1$ we find a $y \in Y$ such that $m \in y$ (see definition of t_Y), i.e. $m \in X_2 \cap y$, yielding $m \in X_2$. □

Proposition 2.3 *The function t_Y as defined in the last proposition, is a $\vee$-homomorphism.*

Proof: We have to show that $t_Y(X_1 \cup X_2) = t_Y(X_1) \vee t_Y(X_2)$ holds for all $X_1, X_2 \subseteq A$. Using the definition of P_X in Equation 1, we know

$$P_X(y_i) = \begin{cases} 1, & X \cap y_i \neq \emptyset \\ 0, & X \cap y_i = \emptyset \end{cases} \tag{3}$$

From the definition of t_Y in Equation 2, we know that

$$t_Y(X_1) = (P_{X_1}(y_1), ..., P_{X_1}(y_q)) \text{ and } t_Y(X_2) = (P_{X_2}(y_1), ..., P_{X_2}(y_q)) \tag{4}$$

Now we claim that

$$\forall i = 1, ..., q : P_{X_1}(y_i) \vee P_{X_2}(y_i) = P_{X_1 \cup X_2}(y_i).$$

Assume $P_{X_1}(y_i) \vee P_{X_2}(y_i) = 1$ for some i. By Equation 1 this is equivalent to $X_1 \cap y_i \neq \emptyset$ or $X_2 \cap y_i \neq \emptyset$. That is, $(X_1 \cap y_i) \cup (X_2 \cap y_i) = (X_1 \cup X_2) \cap y_i \neq \emptyset$ which is, again by Equation 1, $P_{X_1 \cup X_2}(y_i) = 1$. This proves our claim. This means that $t_Y(X_1 \cup X_2)$ and $t_Y(X_1) \vee t_Y(X_2)$ agree in each component, i.e. they are equal. □

In the following we will discover two further properties that the mapping t_Y is satisfying. We will employ them in Section 4.

Corollary 2.4 (injectivity constraint) *The mapping t_Y is k-injective, if and only if*

$$\forall X_1, X_2 \in \mathcal{P}(A) : |X_1| \leq k, |X_2| \leq k \Rightarrow t_Y(X_1) \parallel t_Y(X_2) \tag{5}$$

Proof: "$\Leftarrow$": Assume $t_Y(X_1) = t_Y(X_2)$ for $X_1, X_2 \in \mathcal{P}(A)$, then w.l.o.g $|X_2| > k$ otherwise Equation 5 will not be fulfilled. Let $|X_1| \leq k$ then $t_Y(X_1) \geq t_Y(X_3) \quad \forall X_3 \leq X_2, |X_3| \leq k$ in contradiction to 5. Hence, $|X_1| > k, |X_2| > k$.
"$\Rightarrow$": Let $|X_1| \leq k, |X_2| \leq k$. W.l.o.g assume $t_Y(X_1) \geq t_Y(X_2)$. Then $t_Y(X_1) = t_Y(X_1 \cup X_2)$ with $|X_1| \leq k$. This contradicts the k-injectivity of t_Y. □

Corollary 2.5 (The strong injectivity constraint) *The mapping t_Y is k-injective, if and only if $\forall s \in A, X \in \mathcal{P}(A) : |X| \leq k \implies$*

$$t_Y(s) \parallel t_Y(X) \tag{6}$$

Proof: " $\Leftarrow$ ": Assume $|X_1| \leq k, |X_2| \leq k$, and w.l.og. that $t_Y(X_1) \leq t_Y(X_2)$, then $\forall$ allergene $s \in X_1$ and $s \notin X_2$, we have that $t_Y(s) \leq t_Y(X_1)$, and this implies that

$$t_Y(s) \leq t_Y(X_2)$$

which contradicts 6
" $\Rightarrow$ ": Same as in Corollary 2.4 with $X_2 = \{s\}$. □

Corollary 2.6 *The mapping t_Y is specified by the set of images of* 1*-element subsets of A.*

Proof: This is an immediate conclusion of Proposition 2.3. The details of the proof are left to the reader. □

By the last corollary, we know that we do not have to define the set Y "rowwise", but we can "columnwise" define the set of images $\{t_Y(\{a\} \mid a \in A\}$. This is, what we will do in the algorithm (see Section 4). Consider Figure 3 for a visualization of that fact. There you see (reading rowwise) a set of mixtures Y consisting of the components $Y_1, \ldots, Y_9$. Columnwise, you find the images of the one-element arguments of the respective function t_Y, namely $t_Y(\{a_1\}), \ldots, t_Y(\{a_{12}\})$. For example, the first column states $t_Y(\{a_1\}) = (0,0,0,0,0,0,1,1,1)$ since a_1 is contained in the mixtures Y_7, Y_8 and Y_9. Indeed, these images determine the whole function t_Y due to Proposition 2.3; the image of a subset of A equals to the componentwise join of the respective columns. The function t_Y given by that table was found by the algorithm described later. It is 2-injective.

t_Y	a_1	a_2	a_3	a_4	a_5	a_6	a_7	a_8	a_9	a_{10}	a_{11}	a_{12}
Y_1									×	×	×	×
Y_2						×	×	×				×
Y_3				×	×			×			×	
Y_4			×		×		×			×		
Y_5		×			×	×			×			
Y_6		×	×	×								×
Y_7	×			×			×		×			
Y_8	×		×			×					×	
Y_9	×	×						×		×		

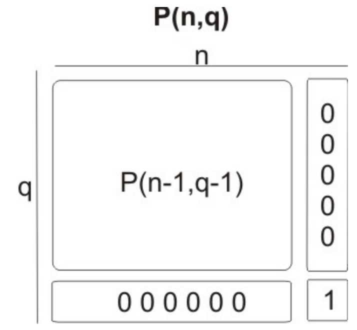
Figure 3: Rowwise/columnwise definition of Y and t_Y.

3 Upper & Lower Bounds for the Number of Mixtures

We consider in this section a heuristic approach to determining the minimum number of prick tests to do.

3.1 Upper bounds for the number of mixtures, q[n]

Generally, the problem of finding a k-injective function t_Y for n substances and q tests is denoted by $P_k(n,q)$. We call $q_k[n]$ the least number of tests we require to perform given n substances and q-allergenes of a given patient.
It is intuitively clear that $q_k[n]$ cannot be smaller than $q_k[n-1]$. If a given problem with n substances cannot be solved with q number of tests then it certainly will not help to try and squeeze another substance in. The formal proof we leave to the reader.
The $q_k[n]$ is no larger than $q_k[n-1]+1$. If $P_k(n-1,q-1)$ is solvable, then we are able to include another substance as shown below, which technically means conducting one additional test. It can be easily shown that $P_k(n,q)$ is being solvable then.



This concludes that $q_k[n]$ is either $q_k[n-1]$ or $q_k[n-1]+1$.

In the following, we outline a heuristical approach giving a k-injective function t_Y for the cases $k=1$, $k=2$ and $k=3$. These are not the best solutions, of course but supply an upper bound for an optimal solution that, as far as we think, can not be found analytically. We call this approach the square method as the determination of the set Y can be intuitively described by arranging the analyzed substances $a \in A$ in a square.

Proofs for the correctness of the described constructions are left to the reader.

3.2 Lower bounds for the minimum q[n]

Finally, we want to emphasize that $k+1$-consistency is harder to solve than k-consistency because it just adds more constraints, so solutions for $q_k[n]$ is a lower bound for $q_{k+1}[n]$.

3.3 Solution for k-injectivity

We present a short description to a heuristic approach to obtaining solutions for k-injectivity. However, we will restrict to the cases where $k=1,2$.

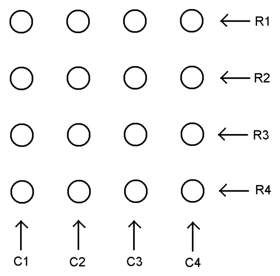
A heuristic solution for 1-injectivity

Arrange the allergenes in a matrix. All rows constitute disjoint parts of a partition R. Similar all columns constitute disjoint parts of another partition C. It is easily seen that these two partitions are 1-connected. Thus, collecting each row and each column in Y, the test function t_Y will be 1-injective.

A heuristic solution for 2-injectivity

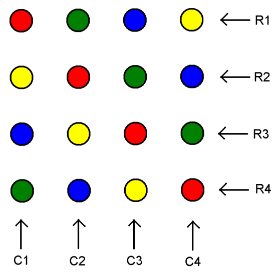
First, if $k = 1$, then Y consists of the rows R_i and columns C_i of the appropriate $m \times m$-square. It can be easily shown that the function t_Y defined by this scheme is indeed 1-injective. We notice that the cardinality of Y is $2m$ in this case. Hence, we have the upper bound:

$$n = m^2, k = 1, q = 2m$$



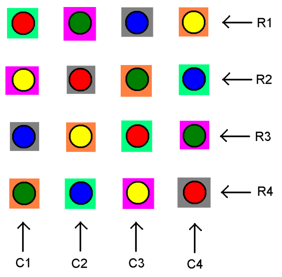
In an analog way we can add the m diagonals D_i (depicted with different colors in the picture) as further compounds and gain a function t_Y being 2-injective. The cardinality of Y is $3m$ then. Hence, we have the upper bound:

$$n = m^2, k = 2, q = 3m.$$



Finally, for the case $k = 3$ we have to add the other diagonals as well. Here, things are getting more complicated, we find out that this is sufficient for odd m. if m is even, the resulting t_Y is not 3-injective yet, s.t. we have to blow up the square to one of size $(m + 1) \cdot (m + 1)$. Hence, the cardinality of Y is $4m$ or $4(m + 1)$ respectively, i.e.

$$n = m^2, k = 3, q = 4m \quad (2 \not| m) \text{ or } q = 4m + 4 \quad (2|m).$$



Arrange the allergenes in a square matrix, adding placeholders for non-existing allergenes if necessary. Let R be the row-partition and C the column-partition. Now imagine making diagonals that continue from the bottom if it reaches the top, always moving in a up-right direction. It can not only be shown that these diagonals are disjoint, but also that the set of diagonals is 1-connected with R and C. Thus, collecting each row, column and diagonal in Y, the test function t_Y will be 2-injective.

4 The Algorithm

4.1 The decimal representation of binary vectors

As stated in Corollary 2.6, it is possible to find a solution columnwise, i.e. by finding the images $\{t_Y(\{a\} \mid a \in A\}$. These images are elements of $\{0, 1\}^q$. We will denote them as binary vectors in the following.

Reading the binary vector as a binary number, we are able to map every single combination as a distinct integer number. In our convention, we read the top element of the binary vector as the smallest digit of the binary number, and the bottom element as the largest. Here are some examples where we use dec: binary vector → integer, and vec : integer → binary vector.

$$dec[(:)] = binary\ number \rightarrow integer$$

$$vec[integer] = vec[binary\ number] \rightarrow [(:)]$$

Remark
By way of notation, we have used [(:)] to denote a "binary vector"

This conversion follows that when the binary vector has q elements, the binary string will have q digits and this will be mapped to integers between 0 and 2^{q-1}. This interval has length 2^q, which is also the number of combinations for the binary vector.The decimal representation is valuable because we can use it as the index to an array of possibilities, if that is what is needed.In Java and C/C++, the following holds

$$dec[X_1 \vee X_2]\ is\ the\ same\ as\ dec[X_1] \mid dec[X_2]$$

$$dec[X_1 \wedge X_2]\ is\ the\ same\ as\ dec[X_1]\&\ dec[X_2]$$

4.2 Comparability and Incomparability

As stated in Remark 2.1, two binary vectors $t_Y(\{a_1\})$ and $t_Y(\{a_2\})$ are comparable if one is componentwise smaller or equal than the other. We want to illustrate this by two examples, see Table 1. On the left, $C_1 \leq C_2$ holds since C_1 is elementwise smaller or equal than C_2. On the right, $C_3 \parallel C_4$ holds because of lines 2 and 4.

C_1	C_2
1	1
1	0
0	0
.	.
.	.
.	.
0	0

C_3	C_4
1	1
1	0
1	0
0	1
.	.
.	.
.	.
0	1

Table 1: Example of comparable and incomparable columns.

4.3 How to find Comparable binary vectors

In this subsection we want to introduce a method to quickly find all the binary vectors (i.e. elements of $\{0,1\}^q$) being greater or smaller than a given binary vector.

In order to iterate over all possible binary vectors greater than or equal to the binary vector y, one should start from $X_1 = y$, and continue by

$$X(n) = vec[dec[X(n-1)] + 1] \vee y$$

which as integers is

$$X[n] = (X[n-1]+1)|y$$

In order to iterate over all possible binary vectors smaller than or equal to the binary vector y, one should start from $X_1 = y$, and continue by

$$X[n] = vec[dec[X(n-1)]-1] \wedge y$$

which as integers is

$$X[n] = (X[n-1]-1)\&y$$

4.3.1 Validity of the Comparability test

Suppose X is a vector of binary numbers that is comparable to a vector y, in the sense that

$$X \geq y \tag{7}$$

We represent X as sequence of $1's$ followed by a 0 and then some arbitrary combinations of $0's$ and $1's$ which we denote by b^X, and whose ordering is not important. That is,

$$X = |1|1|1|1|0|b^X|b^X|b^X|...$$

Starting with the least possible increment of X:

$$X+1 = |0|0|0|0|1|b^X|b^X|b^X|...$$

We make an elementwise comparison of the vector X with those of y. This means that we take each binary element of the vector, X, independent of the other elements.
Now, let b denote each binary bit in X, different from the $b^X{}'s$ such that

$$b \geq b^y$$

after the increment in X. This holds for all $b^y \in y$.

Next, we swap corresponding $0's$ in X with the $b^y{}'s$ from y, so far as the $b^y \neq 0$, otherwise the change is trivial. By (5), the condition

$$b^X \geq b^y \;\; \forall b^X \in X$$

is already satisfied due to comparability. Hence, the bits, b^X, do not have to be changed.
Therefore, we arrive at a least possible increment given as:

$$(X+1) \cup y = |b^y|b^y|b^y|b^y|1|b^X|b^X|b^X|...$$

These, we implement using an operation we call next() through the following procedure:

$$X[0] = y; \tag{8}$$
$$X[n] = next(X[n-1]), \;\; \text{for n=1,2,3,...} \tag{9}$$

Let $X[0]$ be the least element greater than or equal to y i.e.

$$X[0] \geq y \tag{10}$$

Since next() is incrementing X by at least one, and does not involve decreasing operations, the sequence is monotonically increasing:

$$X[0] \leq X[1] \leq X[2] \leq ... \leq X[n-1] \leq X[n] \tag{11}$$

Since next() uses the least possible increment to get to the next element comparable (i.e. greater than or equal) to y,

$$\nexists\ X^* \in X\ \ s.t.\ \ X[n] < X^* < X[n+1]\ \text{ that is comparable to y.} \tag{12}$$

Because of 10 and 11, we know that we are iterating over distinct elements greater than or equal to y. Because of 12, we also know that we do not skip any elements, and that the sequence is complete. This proves the correctness of the statement.

4.4 How to find Incomparable binary vectors

In order to iterate over all possible binary vectors incomparable to the binary vector y, one should partition the elements of y in such a way that $M_0(y)$ contains all rows where y is 0 and $M_1(y)$ contains all rows where y is 1. Now we require that the rows of an arbitrary incomparable binary vector X has at least a 1 in the rows of $M_0(y)$ and at least a 0 in the rows of $M_1(y)$.

Method I-The marking process
In this method, whenever we want to ensure that some binary vector is consistent with the rest of the columns, we check if it has been marked by any of the previous columns. This checking can be done extremely fast using the decimal representations of binary vectors as indexes to a marking list. We also have to check if some union $(c \cup y)$, for y having at most $(k-1)$ elements, are marked. If it was only marked by y earlier we would have

$$t_Y(c \cup y) \geq t_Y(y)$$

which would not be a violation since y is a subset of $c \cup y$. The question is whether it has been marked such that

$$t_Y(c \cup y) \geq t_Y(z)$$

This would not be allowed. In order to distinguish this, we have different marking list for different columns.Some set y would be marked in the list of all substances it contains since y would be comparable to these. Now we only have to check if $t_Y(c \cup y)$ is marked in a list for a substance not in the set y. When we have found a consistent binary vector for a column, we create the marking list for the new column c. Everything that makes the consistency relation come true for some X_1 and X_2 where c is included, should be marked. For y being a set of at most $(k-1)$ elements to ensure k-consistency, we want all binary vectors X, satisfying either one of the following conditions, to be marked in the list for column c.

$$t_Y(X) \geq t_Y(c \cup y)\ \ or$$

$$t_Y(c \cup y) \geq t_Y(X)$$

As we have seen in the Boolean lattice, this however would mark many of the same vectors twice.

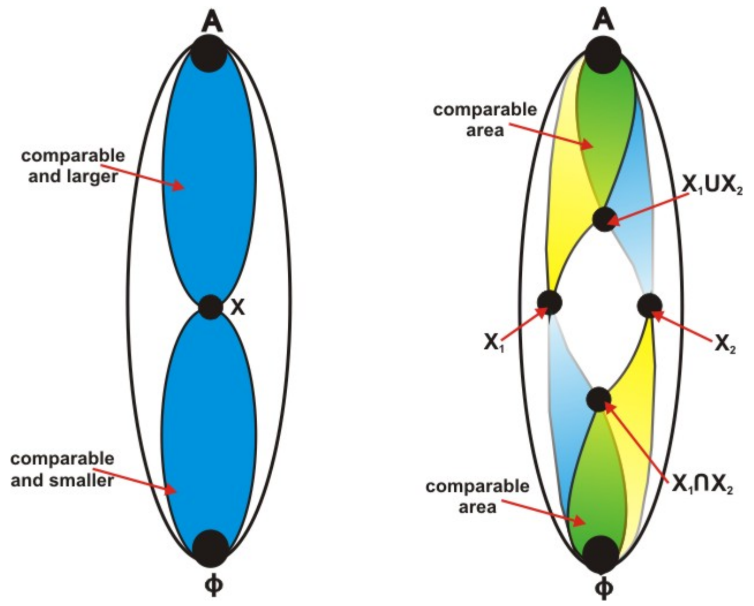


Figure 4: Boolean lattice of a set and two sets with their union, showing comparable areas.

From this, we can see that we need only to mark every X that satisfies either of

$$t_Y(X) \geq t_Y(c)$$

$$t_Y(c \cup y) \geq t_Y(X)$$

Of course, here we would still be marking some binary vectors twice but at least we have improved on the method. We have already shown how to iterate through all comparable binary vectors. What remains is to use these methods to mark everything larger than c, and everything smaller than $c \cup y$, for all possible y sets with at most $(k-1)$-elements. Later columns that do not lead to a solution will be defined, we just have to remove every mark from that columns list, and reinsert marks for the next possible value.

Method II-The Staircase Structure

In method I, we were marking everything that was comparable and checking if some binary vector for the next column had been marked already. This is a sort of having a list of comparable vectors and check for equality. The idea for method II, is having a list of essential vectors and check for comparability. For k-consistency, the essential binary vectors are those which represent the possible unions of at most k-columns. These unions represents the sets of allergies we would like to detect with the tests. This is the same as the set of allergies for which we wanted the test-function to be injective. So what we have is a column c, and then we are trying to iterate over all possibilities for X, to test is the condition for consistency holds

$$t_Y(c) \geq t_Y(X)$$

$$t_Y(X) \geq t_Y(c)$$

Of course this is the same as asking the following question, for which we have already described a method to provide the answer. $t_Y(c)$ comparable to $t_Y(X)$
Each column has its own list where it stores the essential vectors X, representing a union where the column itself is included. When arranged as below, this gives a staircase looking structure
The 1-consistency and 2-consistency
See the illustrations in Table 2.

c_1	c_2	c_3	c_4	...
c_1	c_2	c_3	c_4	...

	c_1	c_2	c_3	c_4	...
c_1	c_1	$c_2 \cup c_1$	$c_3 \cup c_1$	$c_4 \cup c_1$	...
c_2		c_2	$c_3 \cup c_1$	$c_4 \cup c_1$	...
c_3			c_3	$c_4 \cup c_1$	...
c_4			c_3	c_4	...
c_5					...

Table 2: Representation of the structure of 1-consistency and 2-consistency

The 3-consistency
This would look like a 3-dimensional table with the columns c_1, c_2, c_3, etc. out of all three axis. Every cell would be the union of the three columns specified on the axis. Due to symmetry, half of the table could be removed leaving a 3-dimensional staircase shape. in checking if a column c is consistent with previous columns, we run over all cells in this table and check for comparability. If accepted, we add another step to the staircase which contains the unions which includes the accepted column. If this value is later rejected because no solution was found, we remove the step and build it up with the next possible value.

Construction of k-injective test functions

A partition $P = P_1, P_2, ..., P_n$ of the allergenes A, has the following properties:

1. The union of all parts, covers all of the allergenes: $P_1 \vee P_2 \vee ... \vee P_n = A$
2. All parts are disjoint: $P_i \cap P_j = \emptyset$, whenever i is not equal to j.

We say that two partitions R and C are 1-connected, if any part of R has exactly one allergene in common with any part of C. That is:

$$|R_i \cap C_j| = 1 \text{ for all i and j}$$

Define Y to be a set of $(k+1)$ partitions that are all pairwise 1-connected. The test function t_Y on any single allergene, will output exactly one reaction for each partition. That is:

$$|t_Y(s)| = k + 1$$

Furthermore, since each allergene is situated in different parts of the partitions, the specific $k+1$ parts that react is different for each allergene. Thus, no set X, not containing s, can be found such that

$$t_Y(X) \leq t_Y(s)$$

Also, in order to construct a set X, not containing s, such that $t_Y(X) \geq t_Y(s)$, we would have to get a reaction in each the $k+1$ partitions. The partitions were 1-connected, and so did not have any other substance but s in common. Thus, this construction must satisfy $|X| \geq k+1$. According to Corollary 2.5, this test function is k-injective.

4.5 Detecting Inconsistency

Checking whether a newly added binary vector as a column is consistent with the previous ones (i.e. whether the resulting function t_Y is still k-injective), one only has to check that $t_Y(X_1) \parallel t_Y(X_2)$ holds for all X_1, X_2 (being subsets of the already included columns) of cardinality equal or less than k due to Corollary 2.4. This is by far quicker than checking for k-injectivity as given in subsection 2.1.

4.6 The Pseudocode

The method works by choosing the exact prick-tests we would like the substances to be a part of, and then see if the choice is consistent with the choices of the previous substances. We starts by trying to include the first substance.

```
define column(i)
{
    for all column values
    {
    % check column values for consistency
    if consistent column values then
    define column(i+1)
    else exit
    }
define column(i-1)
}
```

Consistency means that k-element subsets are not comparable, i.e. for $k=2$, if

$$t_Y(\{c_i, c_j\}) \geq t_Y(\{c_m, c_n\}) \Rightarrow \; t_Y(\{c_i, c_j\}) = t_Y(\{c_i c_j, c_m c_n\})$$

4.6.1 Consistent with previous substances

Being consistent with previous substances means the choice of which prick-tests to be a part of should balance with the choices of the previously assigned substances in such a way that they will produce conclusive test results for any patient with at most k-allergies. As already discussed,we realized that we can detect inconsistency by checking if the following relationship exists. X_1 and X_2 having at most k-elements and X_2 not being a subset of X_1 such that

$$t_Y(X_1) \geq t_Y(X_2)$$

This could also be formulated for $t_Y(X_1)$ but not necessarily for $t_Y(X_3)$.

4.7 Possible ways of including a substance

Here, we consider various ways a substance could be included in a test.

Exploiting the need for exclusion
A substance cannot be included in all tests if we want to be able to recognize other substances simultaneously. If a column X_1 is all $1's$, any other column for example X_2 would have to respect the relation:

$$t_Y(X_1) \geq t_Y(X_2)$$

From our definitions, the columns X_1 and X_2 would then be inconsistent with each other.

Exploiting the need for inclusion
A substance cannot be excluded from all tests if we want to be able to recognize any reactions to it. If a column X_1 is all $0's$, any other column for example X_2 would have to respect the relation:

$$t_Y(X_2) \geq t_Y(X_1)$$

We would then conclude that the columns X_1 and X_2 are inconsistent with each other as defined.

Exploiting ordering of substances
We do not distinguish between the individual substances and so the columns of the test-matrix can be switched around without changing the solution. However, instead of going through all solutions and permutations, we consider once in each distinct solution. As a result, we are allowed to invent some kind of total ordering of the columns that will remove the permutations and reduce the search space. We have chosen an ordering based on the integer representation and requires

$$dec[c_1] < dec[c_2] < dec[c_3] < ...$$

We made the order-relations strictly increasing since equality between the columns would result in inconsistency.

Exploiting ordering of tests
As with the ordering of substances we do not distinguish between the individual tests and so the rows of the test-matrix can be switched around without changing the solution. This means that all permutations of columns with the same number of test-inclusions will be the same but it does not. The columns are simply bounded too closely together by the already established rules for this to be possible.

Ensuring consistency
For the first column,we have no rules that could break consistency as consistency checks are performed by looking backwards in the method. After the first column has been defined, the tests will no longer be the same as some test will differ in the substances they include.
However, inside of the two partitions $M_0(c_1)$ and $M_1(c_1)$, the tests will remain indistinguishable. As more columns are defined, the symmetry-groups will become very complicated to keep hold of, and we have,thus, chosen to limit our focus to the two first columns.

Searching for all non-symmetric solutions
We have already defined that each column should be decimally greater than the previous. For each symmetry-group, we choose exactly that permutation which has the lowest decimal value, otherwise we would throw away an important part of the search space.
The first column is not affected by the consistency-rule. Here, we only have to make sure that we choose the permutation with the lowest decimal value. All columns with the same number of test-inclusions, that is, all binary vectors with the same number of $1's$, are symmetric. This accounts for the reason to iterate the possibilities for the first column by adding $1's$ to the top of the vector as illustrated below.

$$\begin{array}{cccc}
1 & 1 & 1 & 1 \\
0 & 1 & 1 & 1 \\
0 & 0 & 1 & 1 \\
0 & 0 & 0 & 1 \\
0 & 0 & 0 & 0
\end{array}$$

For the second column, we now see that the groups $M_1(c_1)$ and $M_0(c_1)$ are easily separated in each of the possibilities for the first column. Moreover, for the second column, consistency simply means that it should be incomparable to the first column,thus it should have at least one exclusion from the tests of $M_1(c_1)$ and at least one inclusion from the tests of $M_0(c_1)$. The second rule is obeyed by locating the test-inclusions ($1's$) at the top of the binary vector in each of the symmetry-group.

These possibilities could then, for example, be iterated row-wise. The two generating methods for the first two columns contribute to reduce the number of redundant solutions within the search space by a large factor.

Exploiting the number of substances
Our goal is to device some sort of upper bound to the number of substances that can possibly be included from a partial solution. Whenever this upper bound falls below the number of substances we are aiming to reach, we immediately reject this partial solution and try other possibilities for previously assigned columns.

Exploiting solvability of subproblems
When we were trying to produce results, we found the minimum number of tests $q[n]$ for some number of substances n before trying to find $q[n+1]$. Thus, we had the privilege of knowing which subproblems were solvable and which were not. Here are three cases which can not be solved unless the subproblem can be solved.

Case I: This is the simplest case where we have a test with only one substance and where

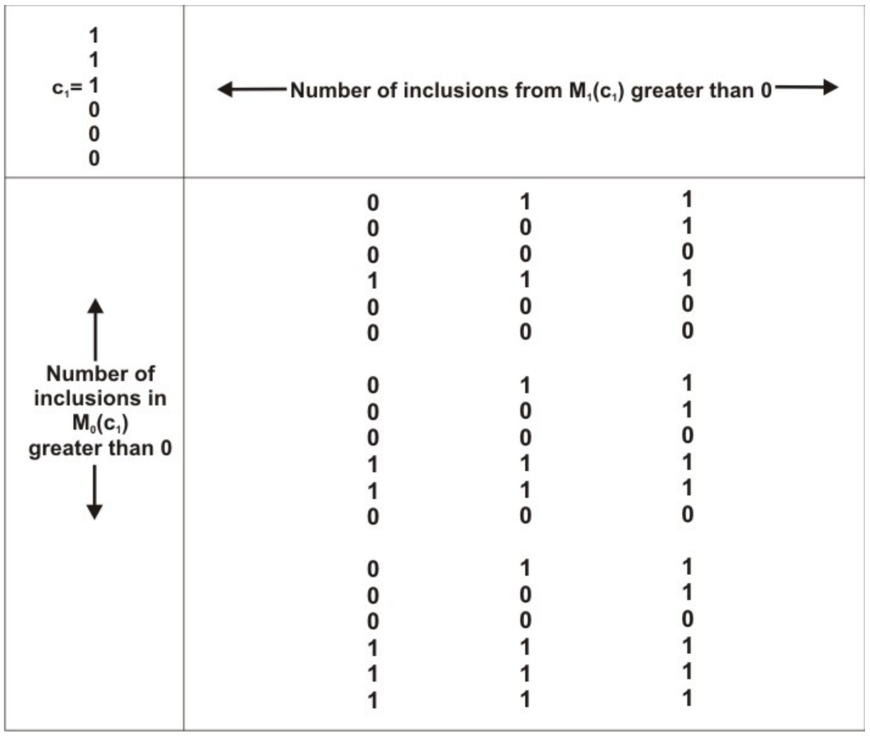
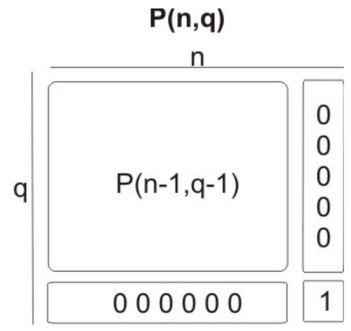


Figure 5: Everything incomparable to c_1 and lowest decimal representative from symmetry-group

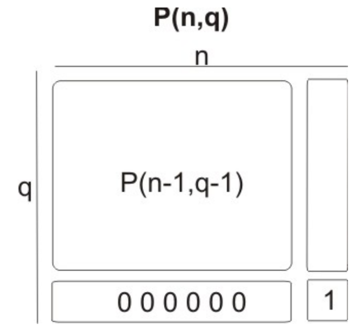
that substance is only included in the test. In this case, the subproblem is entirely independent and we have that solvability of $P(n,q) \Leftrightarrow$ solvability of $P(n-1,q-1)$

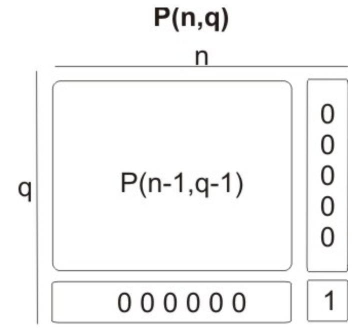


Case II: Here, the substance is allowed to be included in more tests but since we already have a test with nothing but this substance, we already know if we are allergic to it. Its inclusion in the other tests are therefore completely unnecessary and actually superfluous the test results. That is why this case is harder to solve than case I and we can say that solvability of $P(n,q) \Rightarrow$ solvability of $P(n-1,q-1)$

Case III: Here, the test is allowed to contain more substances but since consistency tells us that all columns must be incomparable, the test excludes all other substances, leading back to case II. Again, we can state that, solvability of $P(n,q) \Rightarrow$ solvability of $P(n-1,q-1)$

Now we know that if the subproblem $P(n-1,q-1)$ could not be solved, then all tests





should contain more than one substance, and all substances should be included in more than one test.

5 Results

Test procedure
We only made a program to ensure 2-consistency.i.e for $k = 2$
Since $q_2[n]$ is either $q_2[n-1]$ or $q_2[n-1]+1$, we can always get a conclusion from trying to solve the problem $P_2(n, q_2[n-1])$. If it is solvable, it is the lowest possible value and therefore minimum. If it is not solvable, then the only other choice is $q_2[n-1]+1$, which must be the minimum. Knowing the minimum number of tests for some problem, we just tried to include another substance. The solution to this problem will be conclusive as just shown.
We know $P_2(1,0)$ is not solvable, so we start with $P_2(1,1)$.
Now in general we have to consider two cases:

- If $P_2(n,q)$ is solvable, q is optimal. Try $P_2(n+1,q)$.
- If $P_2(n,q)$ is not solvable, $q+1$ is optimal. Try $P_2(n+1,q+1)$.

In this way, we keep adding another substance and try to solve with the number of tests that were optimal for the problem beforehand. The final results for 2-consistency we found so far is given by Table 3. Plotting the tabulated result, we gain the graph given in Figure 6 for the number of substances against minimum number of tests. The next problem $P_2(15,10)$ did not finish after approximately 10 hours.

Possible Improvement In our implementation, we had a fixed number of tests, and stopped

Number of substances	Minimum number of tests
1	1
2	2
3	3
4	4
5	5
6	6
7	7
8	8
9	9
10	9
11	9
12	9
13	10
14	11

Table 3: Results obtained for different number of substances.

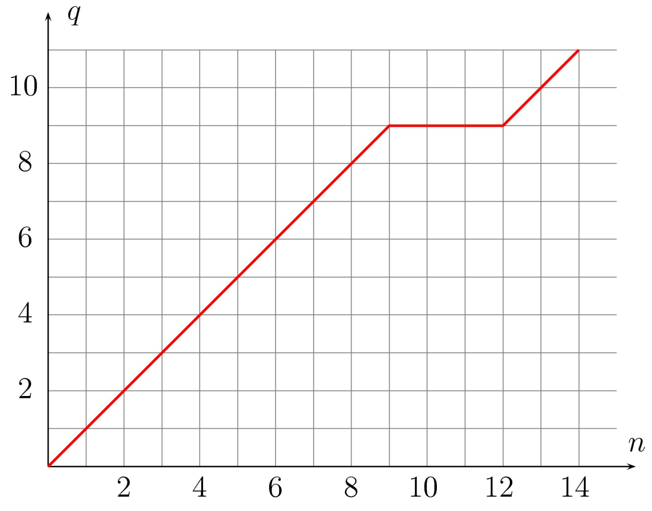


Figure 6: A graph visualizing the algorithms results.

as soon as we had found a solution that included all of the substances we were testing for. Otherwise, it would search the entire combinatorial space and omit to return a solution. Two possible extensions would be:

1. If the method failed to find a solution for a particular number of tests, this number should increase by one, and then the method should continue the search among those combinations that took advantage of the extra test.
2. If we found a solution for a particular number of substances, this solution should just be compared with the solution currently including the largest amount of substances, and the method should afterwards just continue to try and include more substances.

The first extension will allow the method to continue working on its own until it has found a solution including the requested amount of substances. The second extension will allow the method to find the largest amount of substances that can be included with the specified number of tests. Both of them are valuable improvements if more computational effort is put into it.

6 Conclusion/Future work

At the beginning, we showed that we can conduct n tests for given n substances to obtain 1-consistency i.e. a result that is conclusive for a single allergic substance. This turns out only to be a trivial case of the problem at hand, as it became more challenging in providing solutions for higher consistencies.
However, using the mathematical model formulated through the use of boolean lattices we were able to reduce the number of tests needed to be performed to achieve 2-consistency, given n number of allergic substances. We obtained results up to 14 substances for a minimum number of 10 tests (there are $2^{10\times 14} \approx 10^{42}$ possible tests). This also explains the fact that we have a lower-bound for conducting higher tests.Through the heuristic model, we were able to confirm the validity of our model for solving 2-consistency.
We recommend that the mathematical formulation should be reconsidered to impose more constraints on the test function so as to reduce the search space, and hence the computational cost of the algorithm. Moreover, we also recommend further work to finding optimal solution for higher consistency .i.e. $k > 2$, and improving on the methods for finding upper and lower bounds.

References

[1] B. Davey and H. Priestley, *Introduction to Lattices and Order*, Cambridge Univ. Press, second edition, 2002.