Pool Testing for COVID-19: A Simulation Study

Arvind Raghavendran

July 30, 2021

Contents

Acknowledgment

I would like to thank my mentors, Dr.Abhinanda Sarkar and Dr.Sudeshna Adak for spending their valuable time in guiding me through this project. Sir's experience in the field of Statistics and Data Science has helped me understand the necessary theory and has greatly enhanced my interest in the field of data processing and analysis, and Ma'am provided the much needed biological and clinical constraints to be kept in mind. I would also like to thank KVPY and IISc for providing me the opportunity to work on a project during this period.

1 Abstract

This project explores the different methodologies by which pool testing can be done during the ongoing pandemic. We first look at some earlier forms of pool testing, and the necessity for non-adaptive pooling strategies. We then briefly study the well-received *Tapesty* by IIT-Bombay, and the use of compressed sensing as a way to obtain a solution for a pool system. We introduce a similar algorithm with the usage of block designs and BIBDs, and then provide a comparison between the efficiencies and the different BIBDs chosen.

2 Introduction

In 2020, the entire world was hit with the COVID-19 pandemic. Cases rapidly increased and doctors found it very hard to test the entire infected population. While the COVID-19 pandemic has brought tragedy and disruption, it has also provided a unique opportunity for mathematics to play an important and visible role in addressing a pressing issue facing our society.

By now, it's well understood that testing is a crucial component of any effective strategy to control the spread of the virus. Countries that have developed effective testing regimens have been able to, to a greater degree, resume normal activities, while those with inadequate testing have seen the virus persist at dangerously high levels. Developing an effective testing strategy means confronting some important challenges. One is producing and administering enough tests to form an accurate picture of the current state of the virus' spread. This means having an adequate number of trained health professionals to collect and process samples along with an adequate supply of reagents and testing machines. Furthermore, results must be available promptly. A person who is unknowingly infected can transmit the virus to many others in a week, so results need to be available in a period of days or even hours.

One way to address these challenges of limited resources and limited time is to combine samples from many patients into testing pools strategically rather than testing samples from each individual patient separately. It would enable the doctors use lesser number of kits to test more patients. While in theory it seems good, in practice there are a number of things to consider, for instance, how sensitive/specific the design is, how long does it take for the results to be out, does the prevalence matter, etc. Our aim is to end up with a design that would minimize the errors, whilst providing satisfactory results.

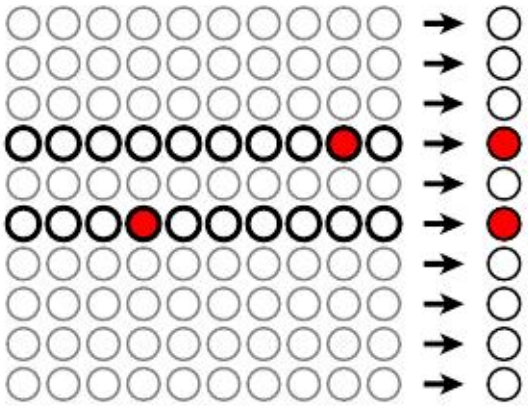
2.1 Earliest forms of pool testing

The concept of pool testing is not something that came up due to the pandemic. Since as early as the 90s, doctors have toyed around with the idea of testing in pools. It was a common practice during war to test the soldiers for any diseases, while they were being drafted. Robert Dorfman in 1943 [1], while testing soldiers for syphilis, found it very cumbersome to test them individually. So he designed and implemented this simple procedure:

Suppose N^2 soldiers were being drafted, then he would divide them into N pools of N soldiers each, and by assumption, the pools were distinct, i.e., a soldier was tested in exactly one pool.

The samples (in this case, blood) of each soldier in a pool were mixed, and then tested for syphilis. If the test was negative, that would mean that no one in the pool had syphilis, and they were all subsequently cleared. If the test was positive, then that would indicate that at least one soldier in that pool was positive. The easiest way to find out whom, would be to test each of the N soldiers in the pool individually.

Consider the case below, where a 100 patients were tested in 10 pools.



8 pools tested negative for syphilis, hence all 80 soldiers in those pools were cleared. For the 2 pools that did test positive, each of the 20 soldiers were tested individually, and then the positive patients were isolated. So, whereas individual testing

would lead us to performing 100 tests, pool-testing enabled us to get the same results using just 30 tests (10 pool + 30 individual). In other words, a 70% reduction in tests. One can, however, immediately notice the problem here. In the event that, however rare that may be, all the pools turn out to be positive, we have to perform 110 tests as opposed to just 100 if we had done it individually. But how rare is this event? Indeed, the answer to this question boils down to the positivity rate of the virus in the area. Let us try to show this mathematically.

Consider the arbitrary case of N^2 samples, pooled into N pools of N samples each. Let X be the number of tests done per pool. Then,

$$X = \begin{cases} 1 & \text{if pool test is negative} \\ N+1 & \text{if pool test is positive} \end{cases}$$

Suppose the prevalence of the disease in the area is p, where $p \in (0, 1)$. Then,

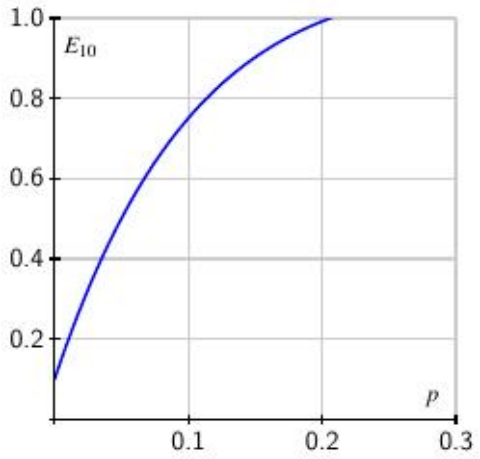
- The pool test is negative iff each sample is negative, i.e., w.p. $(1-p)^N$
- The pool test is positive iff at least one sample is positive, i.e., w.p. $1-(1-p)^N$

Now,

$$\begin{aligned} \mathrm{E}[X] &= 1 \times (1-p)^N + (N+1) \times (1-(1-p)^N) \\ &= 1 + N(1-(1-p)^N) \end{aligned}$$

So, the expected number of tests per person (E(p)) would be $\frac{\mathrm{E}[X]}{N} = \frac{1}{N} + (1-(1-p)^N)$

We wish that E(p) < 1, otherwise we are better off testing each sample individually. Note that, we are assuming the expected number of tests as a function of the positivity rate, since we wish to analyze that given such a pool-design, up to what positivity rate does it make sense to use the design. Suppose we plot E(p) vs p when N=10. The plot looks like the following:



As the graph shows, when $p > 0.2$, meaning there are more than 20 infections per 100 patients, E(p) < 1 and hence, we are better off testing each individual.

Let's define some terms that we would be using quite often from here on out.

- **Sensitivity** : The ability of a test to correctly identify patients with a disease, i.e., $\mathbb{P}$(Test is positive|Patient has the disease) $= \frac{n(\text{True Positive})}{n(\text{True Positive})+n(\text{False Negative})}$
- **Specificity** : The ability of a test to correctly identify patients without the disease, i.e., $\mathbb{P}$(Test is negative|Patient does not have the disease) $= \frac{n(\text{True Negative})}{n(\text{True Negative})+n(\text{False Positive})}$
- **False positive** : The test is positive but the person doesn't have the disease
- **False negative**: The test is negative but the person has the disease
- **True Positive Rate(TPR)** : $\frac{n(\text{True Positive})}{n(\text{True Positive})+n(\text{False Negative})}$
- **False Positive Rate(FPR)** : $\frac{n(\text{False Positive})}{n(\text{False Positive})+n(\text{True Negative})}$

One can see that indeed,

- Sensitivity = TPR
- Specificity = 1 - FPR

An ideal test would have a 100% sensitivity and a 100% specificity. In the above modeling of Dorfman's method, we assume that the test is ideal. However, using Dorfman's pooling method, there is a decrease in the sensitivity of the method. In other words, if S_e was the sensitivity of the original test, then Dorfman's method has a sensitivity of S_e^2. This is because for a sample to be declared positive, it must be positive, both, in the pool test and while testing individually. The specificity of the test, however, increases. If a sample is not infected, testing a second time increases the chance that we detect it as negative.

Several modifications of Dorfman's method were proposed, for instance having two sets of pools in the above sample grid, one row-wise and one column-wise. A sample would be positive iff it was tested positive in both the row-pool and column-pool associated to it. This was shown to improve the sensitivity of the method, albeit increasing E(p) as compared to the original method. However, since the value of E(p) is less for small values of p, this 2-D approach was more desirable.

2.2 Adaptive vs Non-Adaptive strategies

Adaptive strategies are those that perform an initial round of testing, and then use those results to determine how to proceed with a second round, and at the end of this round, the final results are produced. On the other hand, a non-adaptive method produces the final result in a single step. Which method is preferable, is up to us. It is immediate that an adaptive model takes at least twice the time taken by it's non-adaptive counterpart in the instance that there is at least one positive pool. But in general, if we are working with a test that takes hours to produce the result, for instance, in the case of an RT-qPCR test, then we would prefer a non-adaptive test ; but, if we are working with a test that produces results in minutes, like an antibody detection test, then we could model our algorithm using either strategies. Recall that in Dorfman's simple pooling strategy, the number of tests to be run, say per day, is itself a random variable. This is true for any adaptive model in general, since the second round of tests depend on the first round of tests, and that itself is a function of the prevalence p. On the other hand, no matter how long the test takes or how complex the design is, a non-adaptive test is always done once. Now if we think about this from a clinician's point of view, it's always good to know before hand how many tests are to be run, since he/she has to tally and keep in stock the appropriate amount of reagents, chemicals and equipment. For example, suppose a lab has the capacity to perform 10 tests per day, then the clinician can perform 10 non-adaptive tests and produce a definitive result, but the same cannot be said with an adaptive test, since there is no way one can accurately ensure 10 tests can be done, even if you know an estimate of the (then) current value of p. The second round of sample handling can increase the time to result and be laborious to perform since it requires the technician to wear PPE one more time, do another round of RNA extraction, and PCR. So from an experimental and practical point of view, a non-adaptive test is preferred over an adaptive one.

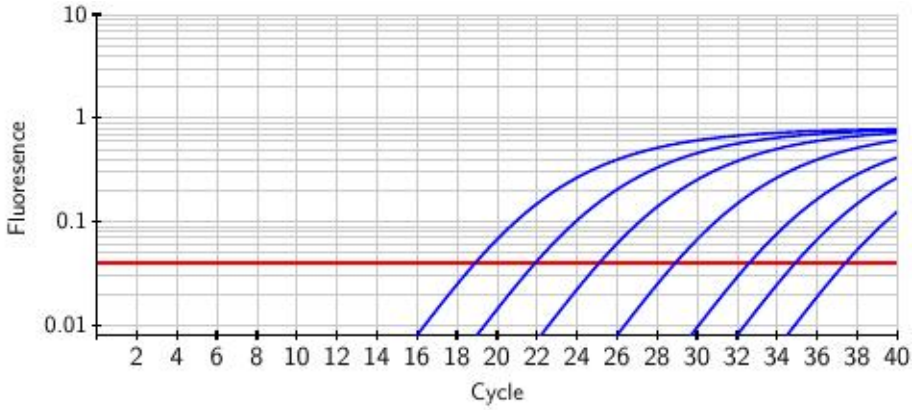
3 The RT-qPCR Test

The Reverse Transcription quantitative Polymerase Chain Reaction Test, or as we know it, the RT-qPCR test, is what is considered as the standard for COVID testing. It is currently the most widely used form of testing against the virus. Let us briefly look into how this test works, since it would help us understand what exactly we should consider while modeling a pool test.

A sample is collected from a patient, often through a nasal swab, and dispersed in a liquid medium. The test begins by converting the virus' RNA molecules into complementary DNA through a process known as reverse transcription (RT). A sequence of amplification cycles, known as quantitative polymerase chain reaction (qPCR) then begins. Each cycle consists of three steps:

- The liquid is heated close to boiling so that the transcribed DNA denatures into two separate strands.
- Next, the liquid is cooled so that a primer, which has been added to the liquid, attaches to a DNA strand along a specific sequence of 100-200 nucleotides. This sequence characterizes the complementary DNA of the SARS-CoV-2 virus and is long enough to uniquely identify it. This guarantees that the test has a high sensitivity. Attached to the primer is a fluorescent marker.
- In a final warming phase, additional nucleotides attach to the primer to complete a copy of the complementary DNA molecule.

Through one of these cycles, the number of DNA molecules is essentially doubled. The RT-qPCR test takes the sample through 30-40 amplification cycles resulting in a significant increase in the number of DNA molecules, each of which has an attached fluorescent marker. After each cycle, we can measure the amount of fluorescence and translate it into a measure of the number of DNA molecules that have originated from the virus.The fluorescence, as it depends on the number of cycles, is shown below:



A sample with a relatively higher viral load will start showing significant fluorescence at an early stage. The curves below represent different samples and show how the measured fluorescence grows through the amplification cycles. Moving to the right, each curve is associated with a ten-fold decrease in the initial viral load of the sample. There is a well-established cut-off for the fluorescence value (denoted by the horizontal red line as above) above which one concludes that SARS-CoV-2 virus is present in the original sample, i.e., the patient was positive. The cycle at which this threshold is attained is called Cycle Threshold, or C_t value.So, while the test has a binary result, i.e., a Y/N answer to the question "Is the patient positive?", it also reports a quantitative answer in the form of the C_t value. C_t value is what is used, especially by *Tapestry*, to model existing pooling strategies.

One must remember that pooling an infected sample with other samples actually dilutes the viral load, the result being that the C_t value would be higher than what one would observe if the sample was tested individually. However, there is a limit up to which we can accept a C_t value. It has been experimentally found that, around cycle 40, noise starts to creep in. So, a C_t value above 40 is generally discarded, since it could well be a false positive. Studies by Bilder and Yelin et al investigated the practical limits of pooling samples in the RT-qPCR test and found that a single infected sample can be reliably detected in a pool of up to 32. The algorithms that we'll be investigating uses pool sizes well less than 32, so we can study them without the fear of controversial test results.

4 Compressed Sensing: A Study

Compressed sensing is a signal processing technique for efficiently acquiring and reconstructing a signal, by finding solutions to under-determined linear systems [5]. This is based on the principle that, through optimization, the sparsity of a signal can be exploited to recover it from far fewer samples than required by the Nyquist–Shannon sampling theorem, which says that if a real signal's highest frequency is less than half of the sampling rate, then the signal can be reconstructed perfectly by means of sinc interpolation. There are two conditions under which recovery is possible. The first one is sparsity, which requires the signal to be sparse in some domain. The second one is incoherence, which is applied through the isometric property, which is sufficient for sparse signals.

At first glance, compressed sensing might seem to violate the sampling theorem, because compressed sensing depends on the sparsity of the signal in question and not its highest frequency. This is a misconception, because the sampling theorem guarantees perfect reconstruction given sufficient, not necessary, conditions. A sampling method fundamentally different from classical fixed-rate sampling cannot *violate* the sampling theorem. Sparse signals with high frequency components can be highly under-sampled using compressed sensing compared to classical fixed-rate sampling.

4.1 Sparse Signals

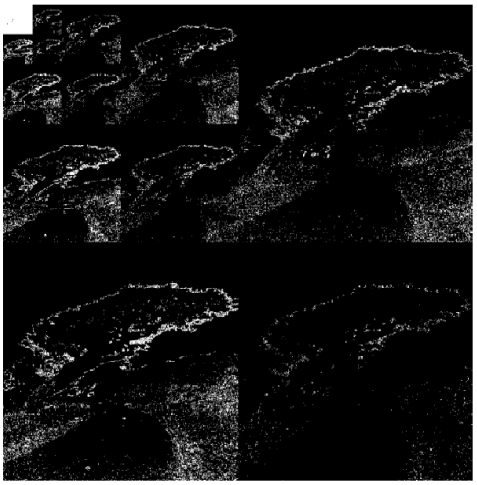
Let $\sum_k = \{x : ||x||_0 \leq k\}$, where $||x||_0$ is the 0^{th} norm of x, or the number of non-zero components of x. So, $\sum_k$ is the set of all k-sparse signals. Typically, we will be dealing with signals that are not themselves sparse, but which admit a sparse representation in some basis Φ. In this case we will still refer to x as being k-sparse, with the understanding that we can express x as $x = \Phi c$ where $||c||_0 \leq k$.

4.1.1 Geometry of Sparse Signals

Sparsity is a highly nonlinear model, since the choice of which dictionary elements are used can change from signal to signal. This can be seen by observing that given a pair of k-sparse signals, a linear combination of the two signals will in general no longer be k-sparse, since their supports may not coincide. The set of sparse signals $\sum_k$ does not form a linear space, instead, it is the union of all possible $\binom{N}{k}$ canonical subspaces. For example, suppose we want all 2-sparse vectors in $\mathbb{R}^3$. Then it is clearly $\sum_2 = \{(x, 0, 0)|x \in \mathbb{R}\} \cup \{(0, x, 0)|x \in \mathbb{R}\} \cup \{(0, 0, x)|x \in \mathbb{R}\}$ i.e., the union of $\binom{3}{2}$ spaces.

4.1.2 Practical Application

Before we proceed, let us look at a practical application of such sparse models in the form of image compression and image denoising. Most natural images are characterized by large smooth or textured regions and relatively few sharp edges. Signals with this structure are known to be very nearly sparse when represented using a multi-scale wavelet transform. The wavelet transform consists of recursively dividing the image into its low- and high-frequency components. The lowest frequency components provide a coarse scale approximation of the image, while the higher frequency components fill in the detail and resolve edges.



In the above image, large coefficients are represented by light pixels, while small coefficients are represented by dark pixels. Observe that most of the wavelet coefficients are close to zero. Hence, we can obtain a good approximation of the signal by setting the small coefficients to zero, or thresholding the coefficients, to obtain a k-sparse representation.

4.2 Sensing Matrices

Given a signal $x \in \mathbb{R}^n$, we consider measurement systems that acquire m linear measurements, i.e., retrieving the signal of length n, given m measurements. The measurement system, is mathematically a matrix. We can represent this process mathematically as,

$$y = Ax$$

where A is an $m \times n$ matrix and $y \in \mathbb{R}^m$. The matrix A represents a dimensionality reduction, i.e., it maps $\mathbb{R}^n$, where n is generally large, into $\mathbb{R}^n$, where m is typically much smaller than n. Note that in the standard CS framework we assume that the measurements are non-adaptive, meaning that the rows of A are fixed in advance and do not depend on the previously acquired measurements.

There are two main theoretical questions in CS. First, how should we design the sensing matrix A to ensure that it preserves the information in the signal x? Second, how can we recover the original signal x from measurements y? In the case where our data is sparse or compressible, we will see that we can design matrices A with $m \ll n$ that ensure that we will be able to recover the original signal accurately and efficiently using a variety of practical algorithms.

Before we answer the first question, let us answer the second one. We will now look at some recovery algorithms.

4.3 Recovery Algorithms

There are a variety of recovery algorithms that can be used to recover the signal. We will, in particular, be concerned with L_1-penalized optimization [6].

Given measurements y and the knowledge that our original signal x is sparse or compressible, we will attempt to recover x by solving an optimization problem of the form,

$$\hat{x} = \arg\min_z ||z||_0 \quad \text{subject to } z \in \mathcal{B}(y) \tag{1}$$

Here, $\mathcal{B}(y)$ is the set of all z that ensure that $\hat{x}$ is consistent with the measurements y. For example, if we are working with noiseless signals, then we set $\mathcal{B}(y)$ as $\{z|Az = y\}$, where A is our sensing matrix. If we are working with a small amount of bounded noise, like in the case of Gaussian noise, we could set $\mathcal{B}(y)$ as $\{z|||Az - y||_2 \leq \epsilon\}$. It is easy to see that in both cases, the optimization problem above finds the sparsest x that is consistent with the measurements of y.

Now observe that in the above, we are assuming that x itself is sparse. As discussed earlier, we could work with signals that are known to be sparse in some representation. Supose, Φ is the basis that sparsifies x, i.e., $x = \Phi c$. In this case, we could modify the approach to instead consider,

$$\hat{c} = \arg\min_z ||z||_0 \quad \text{subject to } z \in \mathcal{B}(y)$$

where, $\mathcal{B}(y)$ is either $\{z|A\Phi z = y\}$ or $\{z|||A\Phi z - y||_2 \leq \epsilon\}$. Note that, if we set A' as $A\Phi$, we would be dealing with the initial optimization problem. So, the actual problem depends not only on the design, but also on the signal itself. However, the computations are similar since if the signal isn't itself sparse, we can modify the design using the basis Φ and proceed similarly. Moreover, it can be shown that if A is a design that enables us to retrieve the signal, then any such A' will also do the same. So, we can simply deal with the case when $\Phi = I$.

However, there is a catch in doing so. The $||||_0$ function is a non-convex function, so the optimization problem isn't feasible; in fact, it is computationally NP-Hard. We can fix this by considering the $||||_1$ norm, since this is a convex function. Specifically,

$$\hat{x} = \arg\min_z ||z||_1 \quad \text{subject to } z \in \mathcal{B}(y) \tag{2}$$

Provided that $\mathcal{B}(y)$ is a convex set, the above optimization is feasible. When we assume a noiseless model, i.e., $\mathcal{B}(y) = \{z|||Az - y||_2 \leq \epsilon\}$, then the above can be stated as a linear program. When noise is there, the problem becomes a convex program with a conic constraint. These optimization problems could all be solved using general-purpose convex optimization software, there now also exist a tremendous variety of algorithms designed to explicitly solve these problems in the context of CS. The algorithm we are going to use is L_1 penalized non-linear optimization. We consider,

$$\hat{x} = \arg\min_z \frac{1}{2}||Az - y||_2^2 + \lambda||z||_1$$

Now, while it is clear that this change transforms a computationally intractable problem into a tractable one, it might not be obvious that the solutions will co-incide. While this can be shown mathematically, we will give a brief argument. It was shown that a bandlimited signal could be perfectly recovered in the presence of arbitrary corruptions on a small interval. The recovery method consisted of searching for the bandlimited signal that is closest to the observed signal in the L_1 norm. This can be viewed as validation of L_1 norm being well-suited to sparse errors.

For some particular choices of λ, the above problem boils down to the constrained version. The goal of the algorithm is to find these choices of λ. Now, a priori, these values are not known to us, and there are a lot of ways of finding them. We, in particular, will be using the AUC of the ROC curve generated by the signals, since we are dealing with binary signals. The exact process will be elaborated upon later, but we will summarize the above; solving equation (2) will provide a solution to equation (1) as well, and this is what we hope to obtain, a sparse recovery. We will review in brief about ROCs, sinc we would be using them quite often here on.

4.4 Receiver Operating Characteristic(ROC)

A receiver operating characteristic curve, or ROC curve, is a graphical plot that illustrates the diagnostic ability of a binary classifier system as its discrimination threshold is varied. Simply put, it can tell, potentially, how close the retrieved signal is to the original signal. The threshold is the value such that anything below it is declared as a 0 in the signal, and anything above it is declared a 1. The ROC curve is created by plotting the true positive rate (TPR) against the false positive rate (FPR) at various threshold settings. Recall that, the true-positive rate is also known as sensitivity and the false-positive rate

can be calculated as (1 - specificity). Sometimes a direct plot of Sensitivity vs Specificity is done, but the X axis is plotted in reverse to keep the shape of the curve the same. ROC analysis provides tools to select possibly optimal models and to discard sub-optimal ones independently from (and prior to specifying) the cost context or the class distribution.We can quantify an ROC curve using the Area Under the Curve, or, AUC. Since the best a model can do is a (Sensitivity,Specificity) of (100,100), we have that $0 \leq \text{AUC} \leq 1$; so the closer the AUC is to 1, the better it behaves. However, the behaviour of curves having the same AUC could be different; i.e., the best (Sensitivity,Specificity) pair that they can offer could be different. Often, the latter is preferred practically, than just the AUC, so it if of interest to look into the same. We will study about these in detail later on.

4.5 Choosing the matrices

Now we have one final question to answer, our first one; how do we choose our designs? In theory, we need the matrix to satisfy certain conditions, such as Restricted Isometry Property(RIP), Null Space Property(NSP), etc. However, we will choose our designs based on performance. Given a design, we first find out the λ of choice as discussed above. For this λ, we retrieve signals and compute the mean AUC over all such retrievals. Mathematically, an AUC of 70+ is deemed good, but from a clinical point of view, we need the AUC to be 97+. So any such design satisfying mean AUC above 97 could be potentially a clinical pooling strategy. There are, other factors of course, but we will discuss them later. There are 2 things to note here. First, while we cannot straightaway tell that the design satisfies above mentioned properties, the fact that it performs close to optimal retrieval means that it would theoretically satisfy the properties. Second, this choice of using an ROC curve to quantify the design is valid since we are going to be not only using binary signals, but also binary measurements of these signals. Simply speaking, all our vectors aren't going to be the actual C_t values, but instead, a simple Y/N (1/0) vector. Hence, it is valid.

From here on begins the core content of the discussion. We will model our pooling designs using BIBDs. Let us read about them.

5 *Tapestry* : An Overview

Tapestry is a pooling strategy proposed by IIT-Bombay [7]. *Tapestry* combines ideas from compressed sensing and combinatorial group testing with a novel noise model for RT-PCR used for generation of synthetic data. Unlike Boolean group testing algorithms, the input is a quantitative readout from each test and the output is a list of viral loads for each sample relative to the pool with the highest viral load. While other pooling techniques require a second confirmatory assay, *Tapestry* obtains individual sample-level results in a single round of testing, at clinically acceptable false positive or false negative rates. *Tapestry* uses two fundamental concepts: Compressed Sensing and Combinatorial Group Testing. We have read about the former, now let us read about the other. Note that *Tapestry* isn't a binary model, i.e., it uses the actual C_t values instead of just 0s and 1s. Hence, we cannot construct any ROC curves for the same directly.

5.1 Combinatorial Orthogonal Matching Pursuit (COMP)

Combinatorial Orthogonal Matching Pursuit (COMP) is a Boolean non-adaptive group testing method. Here one uses the simple idea that if a pool $\tilde{y}_j$ tests negative then any sample $\tilde{x}_i$ for which $A_{ji} = 1$ must be negative. Note that pools which test negative are regarded as noiseless observations. The other samples are all considered to be positive. This algorithm guarantees that there are no 'false negatives'. However it can produce a very large number of 'false positives'. For example, a sample $\tilde{x}_i$ will be falsely reported to be positive if every mixture $\tilde{y}_j$ it is part of, also contains at least one other genuinely positive sample. So, if enough tests aren't used, a lot of false positives would follow The COMP algorithm is largely insensitive to noise. Moreover a small variant of it can also produce a list of 'high confidence positives' after identifying the (sure) negatives. This happens when a positive mixture $\tilde{y}_j$ contains only one sample $\tilde{x}_i$ not counting the other samples which were declared sure negatives in the earlier step.

5.2 The Algorithm

Note that the CS stage following COMP is very important for the following reasons:

- COMP typically produces a large number of false positives. The CS algorithms help reduce the number of false positives as we shall see in later sections.
- COMP does not estimate viral loads, unlike CS algorithms.
- In fact, unlike CS algorithms, COMP treats the measurements in y as also being binary,thus discarding a lot of useful information

The *Tapestry* decoding algorithm involves a two-stage procedure.

In the first stage, we apply the COMP algorithm described above, to identify the sure negatives (if any) in $\tilde{x}$ to form a set $\mathcal{X}$. Let Y be the set of zero-valued measurements in $\tilde{y}$ (i.e., negative tests). Moreover, we define $\bar{\mathcal{X}}, \bar{\mathcal{Y}}$ as the complement sets of X, Y respectively. Also, let $y_{\bar{\mathcal{Y}}}$ be the vector of $m - |\mathcal{Y}|$ measurements which yielded a positive result. Let $x_{\bar{\mathcal{Y}}}$ be the vector of $n - |\mathcal{X}|$ samples, which does not include the $|\mathcal{X}|$ surely negative samples. Let $A_{\bar{\mathcal{X}},\bar{\mathcal{Y}}}$ be the sub-matrix of A, having size $(m - |\mathcal{Y}|)\bullet(n - |\mathcal{X}|)$, which excludes rows corresponding to zero-valued measurements in y and columns corresponding to negative elements in x. In the second stage, we apply a CS algorithm, specifically, L_1 penalized optimization to recover $\tilde{x}_{\bar{\mathcal{X}}}$ from $\tilde{y}_{\bar{\mathcal{Y}}}$, $A_{\bar{\mathcal{X}},\bar{\mathcal{Y}}}$.

However, the COMP algorithm prior to applying the CS algorithm is also very important for the following reasons:

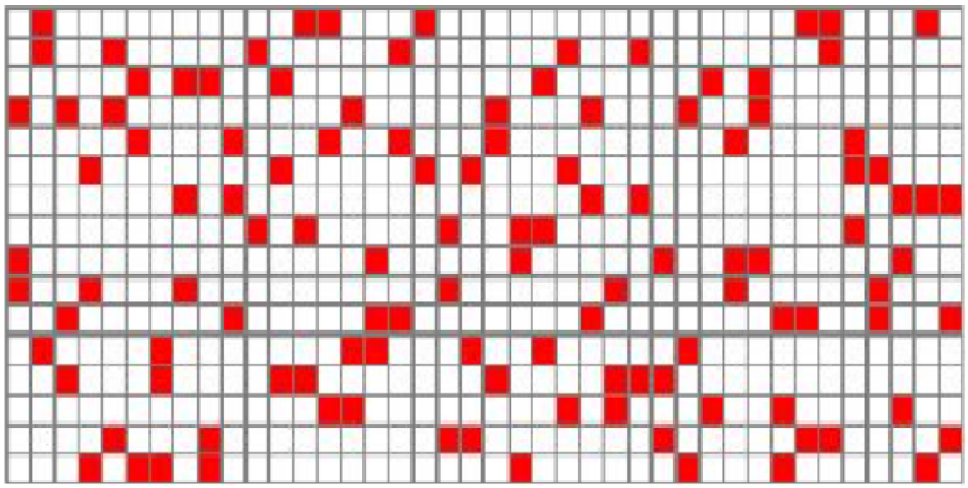
- Viral load in negative pools is exactly 0. COMP identifies the sure negatives in x from the negative measurements in y. Traditional CS algorithms do not take advantage of this information, since they assume all tests to be noisy. It is instead easier to discard the obvious negatives before applying the CS step.
- Since COMP identifies the sure negatives, therefore, it effectively reduces the size of the problem to be solved by the CS step from (m, n) to $(m - |\mathcal{Y}|, n - |\mathcal{X}|)$.

So, in short, we can summarise the algorithm as:

Input: n samples, $m \times n$ pooling matrix A

1. Perform pooling according to pooling matrix A and create m pooled samples
2. Run RT-PCR test on these m pooled samples and receive $m \times 1$ vector of cycle threshold values t
3. Compute $m \times 1$ vector of relative viral loads $\tilde{y}$ from t
4. Use COMP to filter out negative tests and sure negative samples. Compute submatrix $A_{\bar{\mathcal{X}},\bar{\mathcal{Y}}}$, $\tilde{y}_{\bar{\mathcal{Y}}}$ and list *HCP* of 'high-confidence positives' along with their viral loads
5. Use a CS decoder to recover relative viral loads $\tilde{x}_{\bar{\mathcal{X}}}$ from $\tilde{y}_{\bar{\mathcal{Y}}}$, $A_{\bar{\mathcal{X}},\bar{\mathcal{Y}}}$
6. Compute $n \times 1$ relative viral load vector $\tilde{x}$ by setting its entries from $\tilde{x}_{\bar{\mathcal{X}}}$, and setting remaining entries to 0.
7. return $\tilde{x}$, *HCP*

The designs suggested by *Tapestry* are built using Kirkman Triple Systems that satify the NSP and RIP as mentioned before. We will not study this in detail. We will close with an example of a design suggested by Tapestry:



6 Block Designs and BIBDs

6.1 Block Designs

Definiton: A design is a pair (X, A) s.t.

1. X is a set of elements called points
2. A is a multiset of non-empty subsets of X called blocks

Some more definitions[8]:

- **Repeated Blocks:** If two or more blocks are identical in A.
  The fact that the blocks needn't be distinct is why we allow A to be a multiset, rather than a set. Although the blocks are identical, they are not the same and are considered as 2 blocks.
- **Simple Design:** A design with no repeated blocks

6.2 Balanced Incomplete Block Designs(BIBDs)

Definition: Let v, k and λ be positive integers such that $v > k \geq 2$. A (v, k, λ)-BIBD is a design (X, A) that satisfies the following:

1. $|X| = v$
2. each block in A has exactly k points from X
3. every pair of distinct points in X is contained in exactly λ blocks

Property 3 is called as the "balance" property of a BIBD. Note that since we are assuming $k < v$, all the blocks of the design are incomplete, hence the name *incomplete* block design. Using some combinatorial arguments, we can deduce the following two expressions:

$$r = \frac{\lambda(v-1)}{k-1} \tag{3}$$

$$b = \frac{\lambda v(v-1)}{k(k-1)} \tag{4}$$

We can now describe a very important property that we will use soon.

Property: If $\exists\ (v, k, \lambda) \ni$ a (v, k, λ)-BIBD exists, then we have that it satisfies the two relations:

1. $\lambda(v-1) \equiv 0 \bmod (k-1)$
2. $\lambda v(v-1) \equiv 0 \bmod k(k-1)$

While this doesn't help us with identifying if a BIBD exists or not, it can help us in ruling out cases of (v, k, λ) triplets that cannot satisfy a BIBD.

6.3 Incidence Matrices

Strictly speaking, a BIBD is indeed a multiset of subsets of the set X. However, it is mathematically harder to process. If instead, we were to define this as a matrix, then we can build algorithms using this matrix, and therefore, using this BIBD.

Definition: Let (X, A) be a design where $X = \{x_1, ..., x_v\}$ and $A = \{A_1, ..., A_b\}$. The incidence matrix of (X, A) is the $v \times b$ {0,1} matrix $M = (m_{i,j})$ defined by the rule'

$$m_{i,j} = \begin{cases} 1 & \text{if} \quad x_i \in A_i \\ 0 & \text{if} \quad x_i \notin A_i \end{cases}$$

The incidence matrix, M, of a (v, b, r, k, λ)-BIBD satisfies the following properties:

1. every column of M contains exactly k 1s, i.e., column sum of M is k
2. every row of M contains exactly r 1s, i.e., row sum of M is r
3. two distinct rows of M both contain 1s in exactly λ columns

We can relate an incidence matrix to our notions of pool testing and samples.

- The number of rows are the number of pools in our design
- The number of columns are the number of patients whose sample have to be pooled
- The row sum is the pool size
- The column sum is the number of pools in which a person's sample is being tested
- $M_{ij} = 1$ iff the j^{th} person's sample is a part of the i^{th} pool

7 Experiment Methodology

Before we get into the details, there is a fundamental difference to be noted between this approach and any other approach so far. The widely used methods perform the pooled RT-qPCR tests, use the pool C_t values and reconstruct back the C_t value of the patients. We would like to perform, in some sense, an analogue to this, i.e., instead of the actual C_t values, could a similar reconstruction be done using a binary(Y/N) test result? In other words, if our pool tests only tell us that the pool is positive/negative, can we reconstruct the test results of the individual patients?

7.1 Functions

Let us define some functions that would help us in our working.

MakeBinary(v) converts a vector **v** into a binary vector, i.e., zeroes remain as zeroes, and any non-zero value is returned as 1. Intuitively speaking, **x** was a disease score vector for the patients, then MakeBinary(**v**) only tells us which patient is positive/negative. This is the basis for our experimental design.

VectorEstimate(M,y,T,p,lambda) is the function that actually determines the sparse vector. Given the test vector **y**, pool matrix M, basis T, an initial guess p and a penalty coefficient λ(lambda), we use the predefined function SolveL1 in the R1magic package [2] that does L1 non-linear penalized optimization to recover the sparse vector x that satisfies the system of equations $M\mathbf{x} = \mathbf{y}$. In reality, the patient sample vector is sparse since the positivity rate of COVID-19 is low, atmost 5%, so the assumption of a sparse **x** is justified.

First1(x,*n*) divides the vector **x** into n sub-vectors. It returns $i - 1$, where i is the first subvector that contains a non-zero entry.

7.2 Defining the matrices

We are given a practical limit on the pool size. The machine used by us can test up to 8 pools, i.e., it can pipette and fill in 8 rows of samples at one time. It was also preferred that the number of samples we test per day, can be between 10-20. Using these constraints, and the corresponding modulo conditions that we saw, we can now get possible (v, k, λ) triplets, all corresponding to a (v, k, λ)-BIBD each.

Now while it's true that a (v, k, λ) triplet would give a corresponding (v, k, λ)-BIBD, we will slightly modify the iteration to instead return (v, b, k) triplets. The reason is that the function we use to define the BIBDs use (v, b, k) triplets as their inputs. Note that a single such (v, b, k) triplet would correspond to a unique (v, b, k)-BIBD, since given v, b, k, we can uniquely determine λ using the expression (2) as above.

```
## v= 5 b= 10 k= 2
## v= 5 b= 10 k= 3
## v= 5 b= 10 k= 4
## v= 6 b= 10 k= 3
## v= 3 b= 12 k= 2
## v= 4 b= 12 k= 2
## v= 4 b= 12 k= 3
## v= 6 b= 12 k= 3
## v= 6 b= 12 k= 4
## v= 6 b= 12 k= 5
## v= 7 b= 14 k= 3
## v= 7 b= 14 k= 4
## v= 7 b= 14 k= 6
## v= 8 b= 14 k= 4
## v= 3 b= 15 k= 2
## v= 5 b= 15 k= 2
```

```
## v= 5 b= 15 k= 3
## v= 5 b= 15 k= 4
## v= 6 b= 15 k= 2
## v= 6 b= 15 k= 4
## v= 4 b= 16 k= 3
## v= 8 b= 16 k= 7
## v= 3 b= 18 k= 2
## v= 4 b= 18 k= 2
## v= 6 b= 18 k= 2
## v= 6 b= 18 k= 3
## v= 6 b= 18 k= 4
## v= 6 b= 18 k= 5
## v= 4 b= 20 k= 3
## v= 5 b= 20 k= 2
## v= 5 b= 20 k= 3
## v= 5 b= 20 k= 4
## v= 6 b= 20 k= 3
```

We will use the predefined functions *find.BIB* and *isGYD* in the **crossdes** package in R. *find.BIB*(v, b, k) returns the b blocks of the design, i.e., a $v \times k$ matrix of subsets of 1,2,...,b, thereby indicating the row-wise positions of the 1s in the incidence matrix of the (v, b, k)-BIBD. *isGYD*(M), assumes the M to be the blocks of the design, and returns whether this design is balanced or not.

- If balanced, it returns "Balanced w.r.t rows" or "Balanced w.r.t columns".We would desire balance w.r.t rows since we work with the Row incidence matrix.
- If unbalanced, it returns "Neither balanced w.r.t rows or columns"

The *isGYD* has an additional parameter called "tables", which is by default, false. If set to true, it returns a dataframe with a lot of details about the BIBD, specifically,

- Rowsum, i.e., pool size
- Row incidence matrix
- Column incidence matrix
- Concurrence w.r.t rows
- Concurrence w.r.t columns

We can then just call the Row incidence matrix from above, and use it as the matrix of our design.

Note: From here on, the reference of a BIBD will always correspond to a (v, b, k)-BIBD. The reason is that although it is using λ that the formal definition is built, the notions of (v,b,k) are more relatable, since

- v is the number of pools
- b is the number of patients being tested, or the sample size
- k is the number of pools in which a patient's sample is being tested

Apart from this reasoning, as mentioned before, it corresponds well to our function parameters.

Now observe that of the 33 matrices so defined, 2 designs have 8 pools, i.e., 2 matrices have $v = 8$. We wish to test these matrices regarding the accuracy of reconstruction by plotting the ROC curve and the corresponding value of the AUC. There is no specific reason for the choice of $v = 8$, save for the fact that if these designs can be employed, we would actually be using the machine to its maximum capacity. For the sake of comparison, we have also tested the (5,10,2)-BIBD. The three chosen matrices are named as B144,B162 and B101 respectively.

7.3 Methodology

The process is similar for all 3 matrices, and is as follows (Assume we are working with an arbitrary (v, b, k)-BIBD):

1. We define x_0, a zero vector of length $N \times 1$, where N is chosen to be $10000b$.
2. We assume a certain prevalence before iterating. According to the IMA, any value between 1-5 % is valid. If the prevalence is higher, then we are better off testing individually and pool tests are not recommended.
3. We then randomly sample n integer from 1,...,N and then set x_0 as 1 at all these indices. Here, n is Np, where p is the prevalence (as a fraction). The modified x_0 is now our test vector, i.e., actual results of N patients
4. Then, we test these N values, b at a time. It follows a fixed iteration, and this is repeated 10000 times to cover all samples in x_0. The fundamental idea is to find a "good" penalty coefficient λ for a chosen b-subset of x_0, and use this λ to reconstruct the test vector for all the other b-subsets. The iteration is as follows:
   (a) We find the first b-subset of x_0 that has a non-zero entry, using *First1*(x_0). We call this b-subset as $\mathbf{x}$.
   (b) We compute $\mathbf{y}$, where $\mathbf{y} = \mathrm{M}\mathbf{x}$ and M is our BIBD. $\mathbf{y}$ is then made binary using *MakeBinary(**y**)*, as per our methodology.
   (c) Our criterion of a "good" λ is if the reconstruction shows an AUC $\geq 85\%$. Recall that this "good" lambda that we are talking about, is actually one of the (possibly) many lambdas that make the L_1 penalized optimization problem equivalent to the constained optimization problem. We take a 1000 points from 1 to 7 as λ, and then for each of these points, discard λ if the reconstruction using *VectorEstimate(**M**, **y**, **T**, p, λ)* shows an AUC $<$ 85. In doing so, we obtain a (possible) vector of all such "good" λ.
   (d) We set l to be the maximum of all such λ.
   (e) Using the l above, and barring $\mathbf{x}$, we then recreate the test vectors for all the other b-subsets of x_0 using *VectorEstimate* as we did in the case of $\mathbf{x}$, and concatenate the vectors into a $N \times 1$ vector, say x_{01}. Now one might argue if that there are multiple choices of "good" λ, was it valid to use the maximum of these lambdas for all b-subsets? The answer to this can be seen if we repeat the iteration with all possible λs and see the distribution of the AUCs. It can be seen that l achieves the mode AUC for all the designs.
5. Now the value of First1(x_0) need not be 0 all the time, so, we rearrange x_{01} to correspond to the b-subsets it estimated, and this vector is called x_1.
6. So given the test vector x_0, we have recreated it as x_1. All that's left is to run a test for the goodness of fit.

7.4 Analysis of the designs

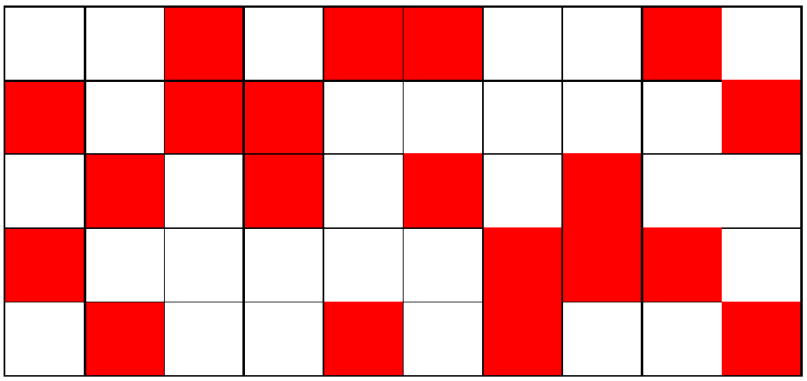
We now perform the above for the 3 chosen designs. Before any computations,we will first show how the matrix looks like as a pool design. A red square in the matrix indicates a 1, and a white square indicates a 0. Along with the AUC of the design, we also provide a 95%-CI for the AUC and the best sensitivity/specificities that the curve can offer, for different positivity rate. All the functions mentioned below are taken from the *pROC* package [3] in R.

- The CI says that if the particular design is used, then there is a 95% chance that the AUC of the design is in the CI, at that prevalence. We will use the *ci.auc()* function in R to compute the same.
- *ci.se()* in R provides a 95% CI for the sensitivities, given different specificities. For example, if the test requires the sensitivity be as high as possible, but the specificity can be atmost 90%, then we can obtain the corresponding CI of the sensitivity using *ci.se()*. For the sake of demonstration, we will report the 95% CI of the sensitivity at a 100% specificity. We do the same for specificity as well using *ci.sp()*
- Another value of interest is the maximum (S_e,S_p) that the curve can offer. Recall that an ROC curve is plotted by taking several thresholds and computing the FPR and TPR at those thresholds. We use *coords*(curve,"local maximas") to get the best thresholds for the curve. Among all such thresholds, we take the threshold for which the geometric mean [4] of the (S_e,S_p) pair is highest. That point would correspond to the best (S_e,S_p) offered by the curve, in other words, the point on the curve closest to the left most corner.

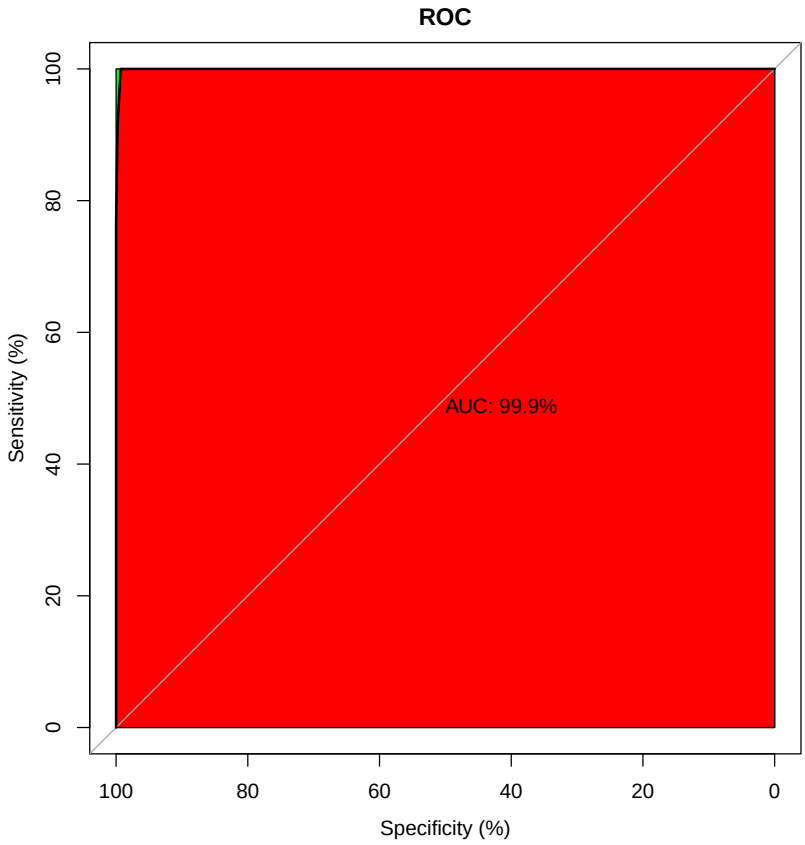
We pick the positivity rates (p) to be 0.01%,0.1%,1%,2%,3%,4%,5% and 10%. The above data is presented as a table.

7.4.1 Design 1: (10,5,2)-BIBD aka B101

Representation of the matrix:



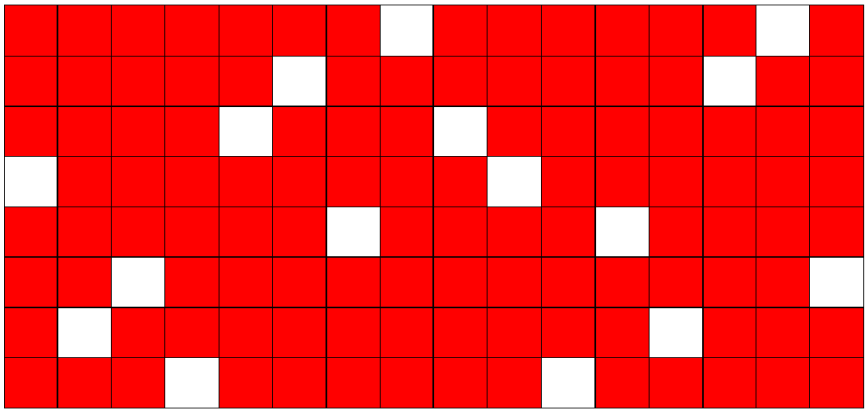
The ROC curve at $p = 3\%$ looks like the following:



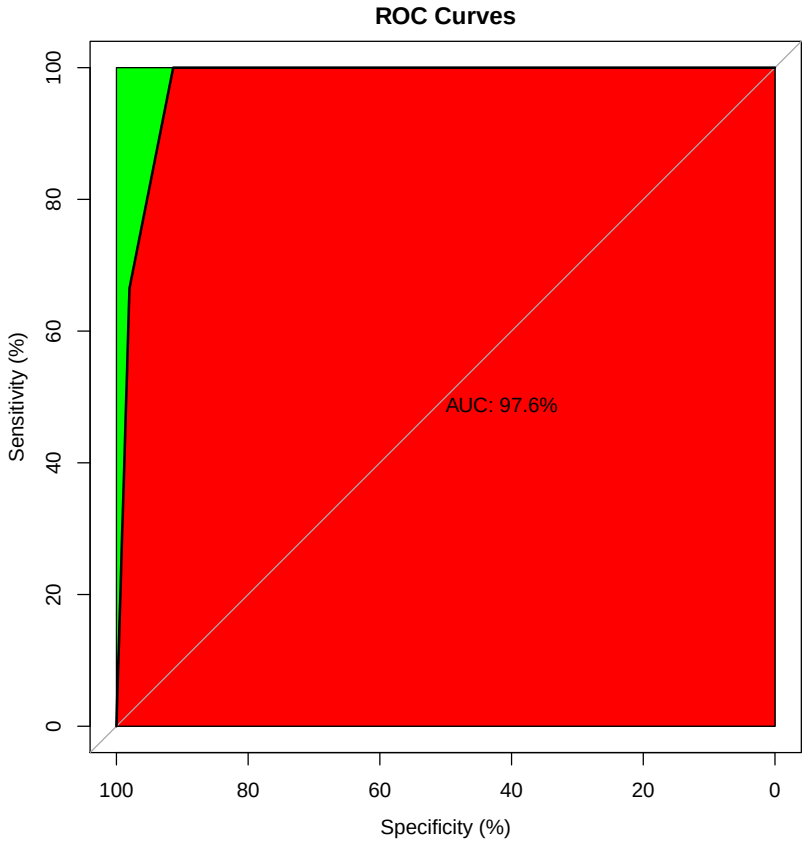
$p \rightarrow$	0.01%	0.1%	1%	2%	3%	4%
AUC(in%)	95	99.5	99.94	99.98	99.93	99.85
95% CI for AUC(in%)	(85.19,100)	(98.51,100)	(99.84,100)	(99.97,99.98)	(99.93,99.94)	(99.84, 99.87)
95% CI for S_e(in%) at 100% S_p	(0,100)	(0,100)	(0,99.92)	(86.05,88.95)	(81.47,84.1)	(22.45,79.33)
95% CI for S_p(in%) at 100% S_e	(0,100)	(55,100)	(82.7,95.1)	(99.59,99.66)	(99.15,99.26)	(0.01,98.72)
Best (S_e,S_p) in curve (in%)	(90,99.99)	(99,99.99)	(99.9,99.08)	(100,99.62)	(99.98,98.66)	(28.22,97.76)

7.4.2 Design 2: (16,8,7)-BIBD aka B162

Representation of the matrix:



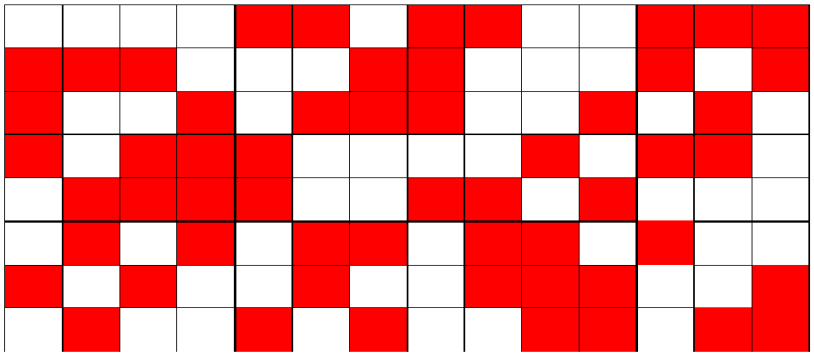
The ROC curve at $p = 3\%$ looks like the following:



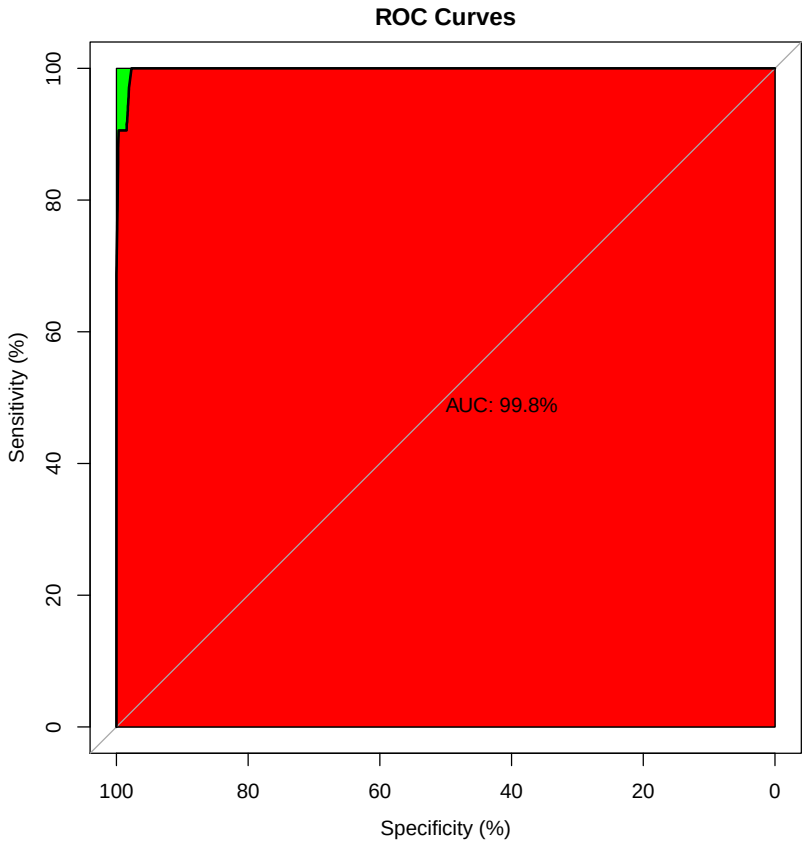
$p \rightarrow$	0.01%	0.1%	1%	2%	3%	4%
AUC(in %)	96.87	99.63	99.41	98.67	97.38	95.85
95% CI for AUC(in %)	(90.73,100)	(99.01,100)	(99.33,99.49)	(98.6,98.73)	(97.29,97.46)	(95.73,96.9)
95% CI for S_e(in %) at 100% S_p	(0,18.75)	{0}	{0}	{0}	{0}	{0}
95% CI for S_p(in %) at 100% S_e	(0,99.99)	(0,99.91)	(0,98.28)	(0,95.38)	(0,90.96)	(86.13,86.46)
Best (S_e,S_p) in curve (in %)	(93.75,99.99)	(99.38,99.90)	(99.94,98.24)	(99.97,95.30)	(99.98,90.85)	(100,86.3)

7.4.3 Design 3: (14,8,4)-BIBD aka B144

Representation of the matrix:



The ROC curve at $p = 3\%$ looks like the following:



$p \rightarrow$	0.01%	0.1%	1%	2%	3%	4%
AUC(in %)	96.42	99.64	99.94	99.9	99.79	99.6
95% CI for AUC(in %)	(89.41,100)	(98.51,100)	(98.86,100)	(99.86,99.95)	(99.76,99.83)	(99.56,99.63)
95% CI for S_e(in %) at 100% S_p	(0,100)	(46.43,100)	(54.43,89)	(37.36,79.57)	(13,71.31)	(40.63,62.25)
95% CI for S_p(in %) at 100% S_e	(0,100)	(0,100)	(0.04,99.38)	(0.16,98.69)	(0.36,97.8)	(0.61,96.78)
Best (S_e,S_p) in curve (in %)	(92.86,100)	(99,29.100)	(99.93,99.35)	(99.96,98.64)	(99.98,97.74)	(99.98,96.71)

7.5 Choosing Better Designs

7.5.1 Real-Time Constraints

The above 3 designs were just a sample of how one could theoretically build pooling systems and compare their efficiencies. Practically, there are a number of things we'd like to consider before choosing the appropriate designs.

Any test, be it the RT-qPCR or just an antibody detection test is sensitive to the concentration of the virus. For example, recall the working process of the RT-qPCR test; it reports back the number of cycles it takes for the viral load to multiply and cross the threshold level. Hypothetically, suppose that the test can detect up to 100 ppm, i.e., if the viral load is less than 100ppm, then the C_t would be more than 40, and subsequently the sample is declared negative. While pooling reduces the number of tests done per person, it can bring about false negatives if designed inaccurately. For instance, consider the design B162. It consisted of 8 pools having 14 samples each, i.e., 14 loads were combined into 1 sample and tested. Pooling is normally done when the positivity rate is less than 3%, so there is a good chance that if the pool consists of just one positive sample and 13 negative samples, the pooled sample would test negative due to less concentration of viral load. So, a good test would have pools of size no more than 7. In terms of the matrix design, we wish to have the row sum, $r \leq 7$. The choice of 7 being the threshold was taken from the performance of B144, since it was a design having 8 pools of size 7 each, and gave very good results for positivity rates up to 4%.

We argued about the optimality of the row sum, but what about the column sum? Recall that the column sum k was the number of pools a patients sample was divided into. It would not be a good idea to divide the original sample into a lot of pools. The simple reason is that even if the patient was originally positive, if we would be sending very less viral load to each pool, then the effect of dilution, as discussed above, would be much stronger. This is especially in the case of patients who are borderline positive. Once again, judging from the performance of B144, we wish that each person's sample goes to no more than 4 pools, i.e., $k \leq 4$.

As argued earlier, the cap for the number of pools was set at 8, which gave us 2 matrices; B162 and B144. However, it needn't always be 8. It would be useful to find designs having number of pools (v) of size 4 or 16 as well. This is simply because these designs can be run with multiple 8-pool machines; in the case of 4 we can run 2 different sets of pools with one machine, and for 16 we can arrange for 2 machines and configure the pipetting to be done as per the whole design. Therefore, we'd like the number of pools $v \in 4, 8, 16$.

Summarizing,

1. $v \in 4, 8, 16$
2. $k \leq 4$
3. $r \leq 7$

A minor improvement could be suggested for the number of patients being pooled (b) as well. It is easier for the lab technician to comprehend and pool a "good" number of patients, say 5,10,20,25,etc. as compared to 14,16 or 12. Though the latter may be mathematically good designs, it is essential to make the design look practically intuitive. Pooling 20 patients into 6 pools is something easier to keep track of and do, rather than pooling 23 patients into 6 pools, from a practical point of view. However, this isn't a very strict necessity; not as strict as the 3 above.

As computed earlier, we will once again list out possible (v, b, k) triplets satisfying the above.

```
## v= 4 b= 6 k= 2 r= 3[1] "\n"
## v= 4 b= 8 k= 3 r= 6[1] "\n"
## v= 4 b= 12 k= 2 r= 6[1] "\n"
## v= 8 b= 14 k= 4 r= 7[1] "\n"
## v= 16 b= 20 k= 4 r= 5[1] "\n"
## v= 16 b= 24 k= 2 r= 3[1] "\n"
## v= 16 b= 24 k= 4 r= 6[1] "\n"
## v= 8 b= 28 k= 2 r= 7[1] "\n"
## v= 16 b= 28 k= 4 r= 7[1] "\n"
## v= 16 b= 32 k= 3 r= 6[1] "\n"
## v= 16 b= 48 k= 2 r= 6[1] "\n"
```

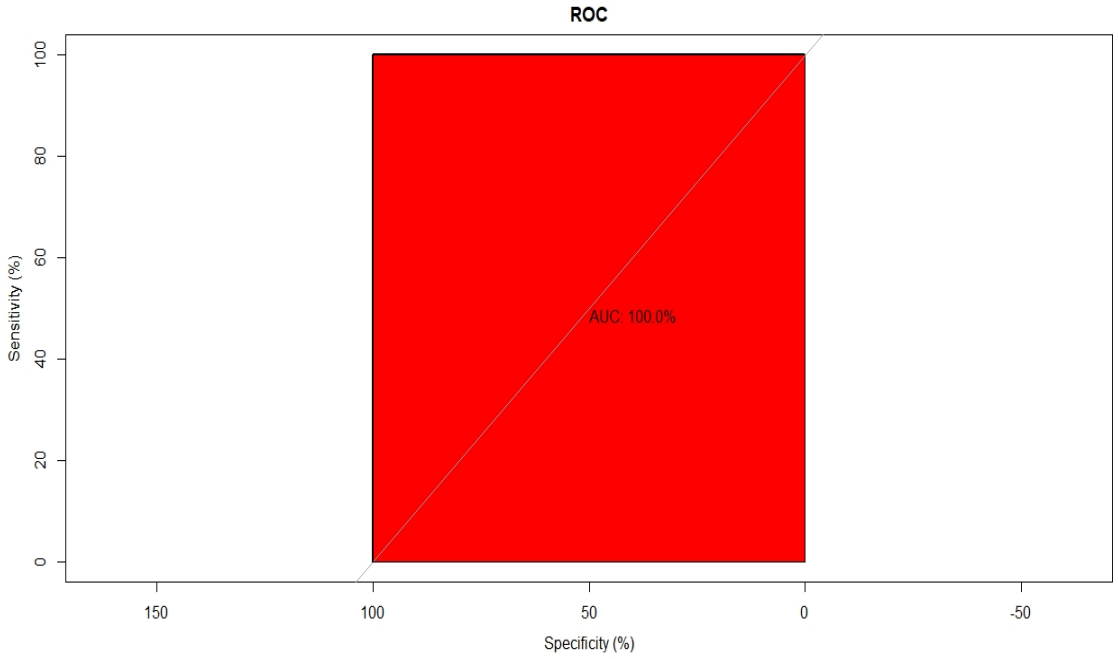
The corresponding designs were also generated as matrices and stored. They were called B61,B81,B122,B144,B206,B241,B242,B281,B282,B321 and B481 respectively. B144 defined here is the same matrix defined earlier.

7.5.2 Analyzing the designs

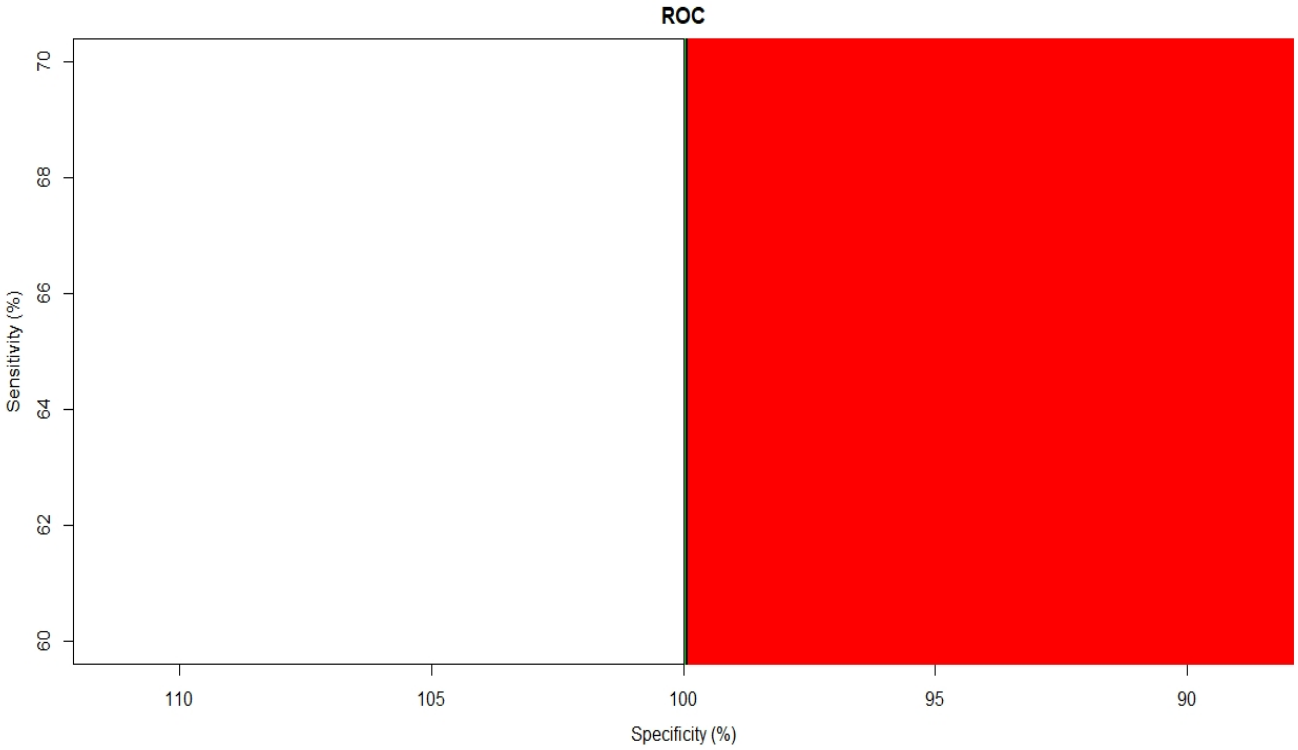
We have thus obtained 11 designs than satisfy the practical constraints. For each of the designs, we will use the same methodology as before and recreate the table, but instead of a CI around the AUC, we will report the mean value of AUC (taken over 100 different test vectors) as an estimate of the original AUC.

From here on, we would be more interested in the best (S_e, S_p) rather than a CI around the S_e when S_p is 100% and vice versa. The reason is the nature of the ROC curve. The ROC curve, if you recall, is taken by choosing different thresholds to classify the 0s and 1s, and plotting the corresponding (S_e, S_p) values. This curve is not a "smooth" curve all the time, it is a concatenation of lines obtained between thresholds, and can behave quite differently at the end points.

Consider the following scenario. We used the (4,8,3)-BIBD aka B81 as our pool design, and it was tested at 3% positivity rate. A particular test vector gave the following ROC curve:



It looks like a promising design with an AUC of 99.95% and if one were to guess the sensitivity at 100% specificity, it would be close to 95-99%, if not 100%. This, however, is not the case. If we magnify the Y-axis between 60-70%, we would observe the following:

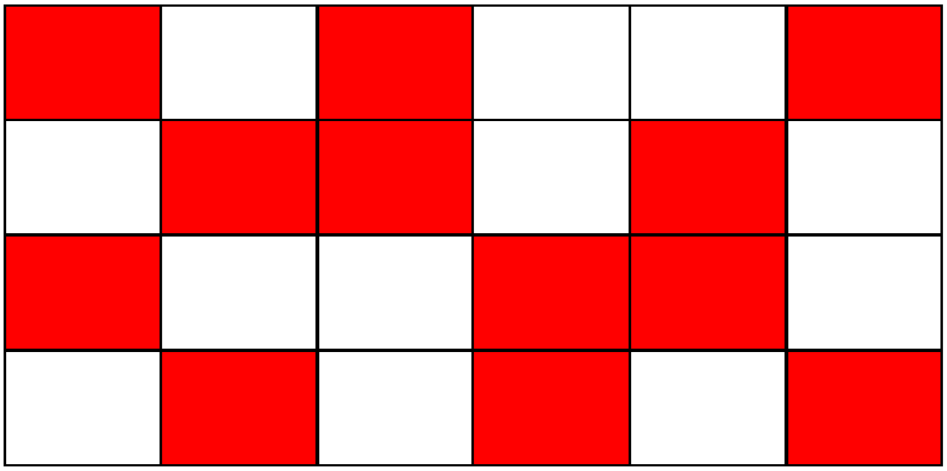


The green area is a part of the Max-AUC-curve, i.e., an ROC having AUC=100%. We are well aware that our curve isn't a Max-AUC curve; it has an AUC of 99.95%. What matters to us is where that 0.05% is lost; is it the sensitivity, or the specificity, or both? In this case, clearly the loss is more in the sensitivity than the specificity, as shown by the magnified plot. In fact, if one were to run *ci.se()* on this curve at 100% specificity, we would see that the resultant CI is 0, indicating that at 100% specificity, the curve cannot identify the 0s from the 1s and will report all of them as 0s. However, if we were to sacrifice a little specificity, say around 0.01%, then we can see that the curve behaves well. In fact, if we run *ci.se()* on this curve at 99.99% specificity, the resultant CI is 100. An analogous situation can happen at 100% sensitivity as well. Therefore, this behavior of the ROC curve at the local maximas on either axes can be misleading towards the design, and hence, it is a good idea to report the non-trivial local maxima, i.e., the left-most point of the curve. That would give a better picture about the design. If the design is good, this point would be close to (100,100) meaning that the loss incurred in the AUC is both due to a loss in sensitivity and specificity. Reporting this left-most point of the curve, is as good as reporting the curve itself, so we won't need to picture the curve as done earlier.

7.5.3 Design 1: (4,6,2)-BIBD aka B61

- Number of pools : 4
- Number of times a patient's sample is divided : 2
- Pool size : 3
- Number of patients being pooled : 6

Representation of the matrix:

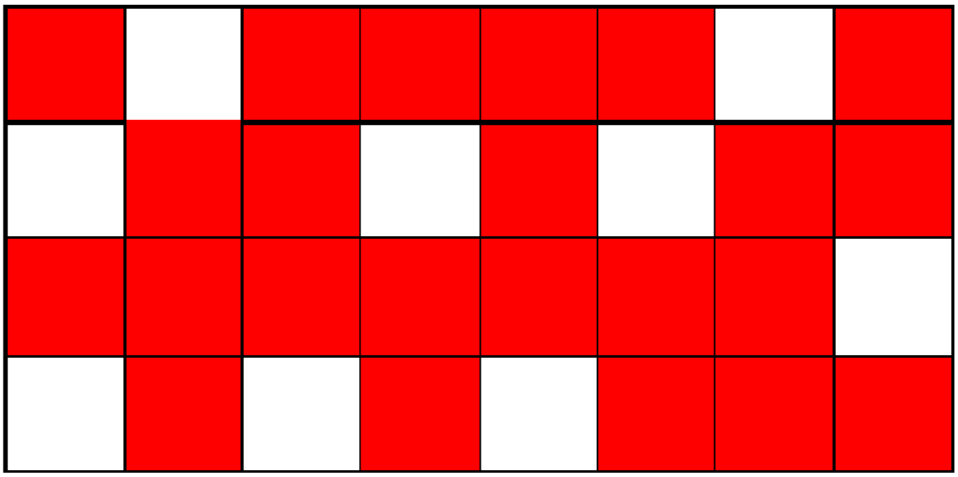


$p \rightarrow$	0.01%	0.1%	0.5%	1%	2%	3%
Mean AUC(in %)	100	100	99.9999	99.9992	99.9947	97.98
Best (S_e,S_p) in curve (in %)	(100,99.992)	(100,100)	(100,99.991)	(100,99.96)	(100,99.845)	(98,97.666)

7.5.4 Design 1: (4,8,3)-BIBD aka B81

- Number of pools : 4
- Number of times a patient's sample is divided : 3
- Pool size : 6
- Number of patients being pooled : 8

Representation of the matrix:

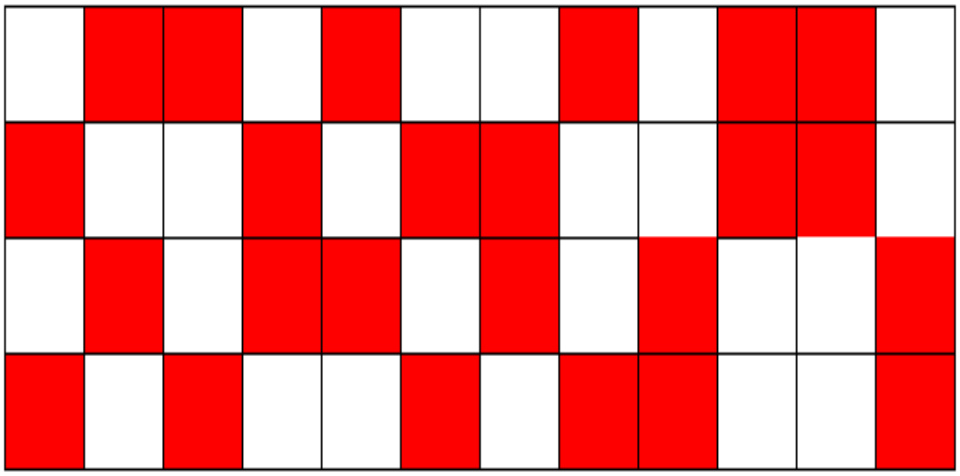


$p \rightarrow$	0.01%	0.1%	0.5%	1%	2%	3%
Mean AUC(in %)	99.95	99.95	98.755	98.011	97.505	92.608
Best (S_e,S_p) in curve (in %)	(100,99.991)	(100,99.899)	(99,98.482)	(99,98.482)	(98.5,96.122)	(94,90.499)

7.5.5 Design 1: (4,12,2)-BIBD aka B122

- Number of pools : 4
- Number of times a patient's sample is divided : 2
- Pool size : 6
- Number of patients being pooled : 12

Representation of the matrix:

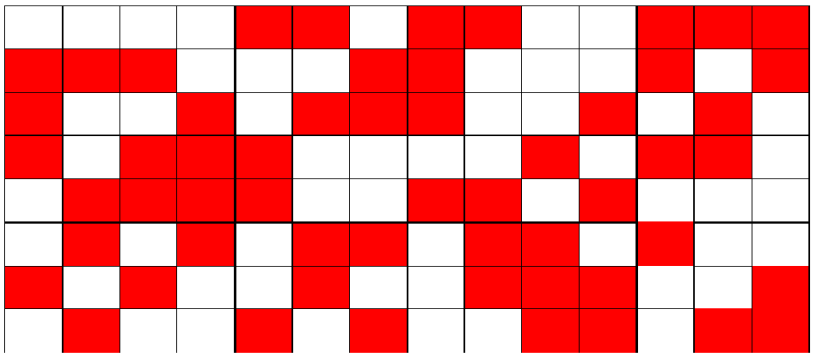


$p \rightarrow$	0.01%	0.1%	0.5%	1%	2%	3%
Mean AUC(in %)	99.995	99.95	99.75	98.934	97.766	96.82
Best (S_e,S_p) in curve (in %)	(100,99.989)	(100,99.898)	(100,99.46)	(99.432,98.292)	(98.78,96.25)	(98.38,94.24)

7.5.6 Design 1: (8,14,4)-BIBD aka B144

- Number of pools : 8
- Number of times a patient's sample is divided : 4
- Pool size : 7
- Number of patients being pooled : 14

Representation of the matrix:

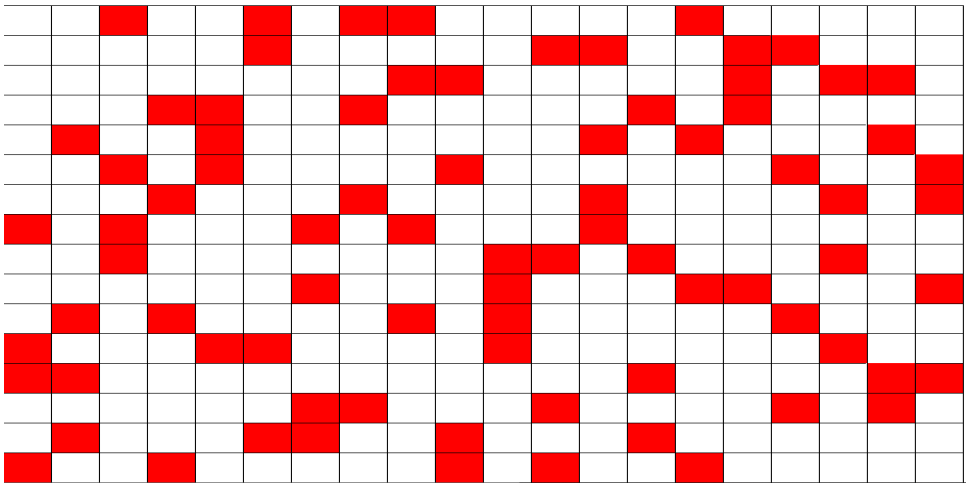


$p \rightarrow$	0.01%	0.1%	0.5%	1%	2%	3%
Mean AUC(in %)	100	99.999	99.994	99.97	99.87	99.71
Best (S_e,S_p) in curve (in %)	(100,100)	(100,99.99)	(99.977,99.599)	(100,99.11)	(100,98.11)	(100,97.091)

7.5.7 Design 1: (16,20,4)-BIBD aka B206

- Number of pools : 16
- Number of times a patient's sample is divided : 4
- Pool size : 5
- Number of patients being pooled : 20

Representation of the matrix:

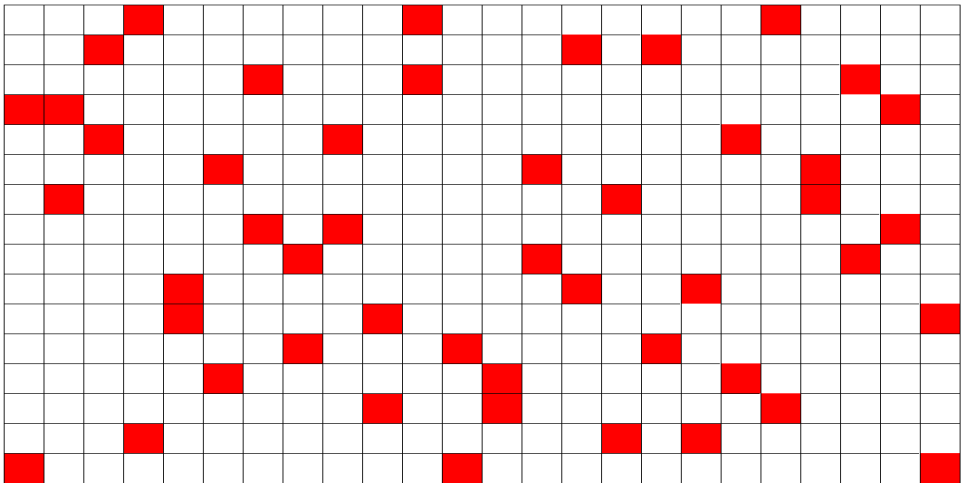


$p \rightarrow$	0.01%	0.1%	0.5%	1%	2%	3%
Mean AUC(in %)	100	100	100	99.999	99.999	99.998
Best (S_e,S_p) in curve (in %)	(100,100)	(100,100)	(100,99.999)	(99.999, 99.995)	(99.96,99.949)	(99.83,99.84)

7.5.8 Design 1: (16,24,2)-BIBD aka B241

- Number of pools : 16
- Number of times a patient's sample is divided : 2
- Pool size : 3
- Number of patients being pooled : 24

Representation of the matrix:

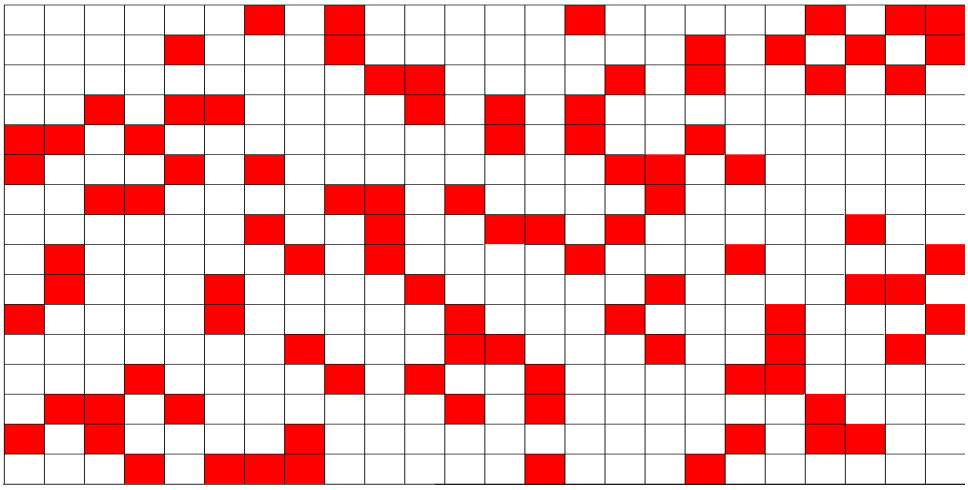


$p \rightarrow$	0.01%	0.1%	0.5%	1%	2%	3%
Mean AUC(in %)	100	100	100	99.999	99.996	99.988
Best (S_e,S_p) in curve (in %)	(100,100)	(100,99.99)	(100,99.996)	(99.983,99.968)	(99.872,99.833)	(99.68,99.61)

7.5.9 Design 1: (16,24,4)-BIBD aka B242

- Number of pools : 16
- Number of times a patient's sample is divided : 4
- Pool size : 6
- Number of patients being pooled : 24

Representation of the matrix:

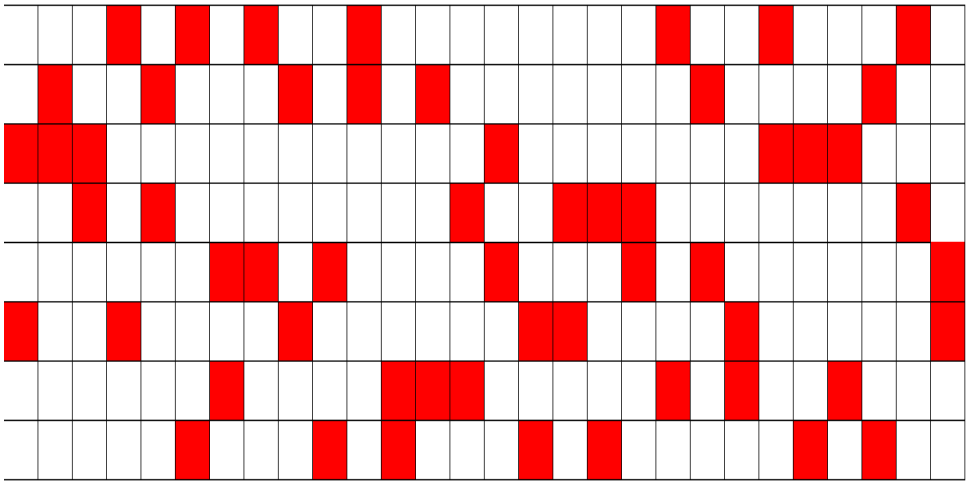


$p \rightarrow$	0.01%	0.1%	0.5%	1%	2%	3%
Mean AUC(in %)	100	100	99.99	99.999	99.996	99.98
Best (S_e,S_p) in curve (in %)	(100,100)	(100,99.999)	(100,99.996)	(99.98,99.962)	(99.85,99.77)	(99.61,99.45)

7.5.10 Design 1: (8,28,2)-BIBD aka B281

- Number of pools : 8
- Number of times a patient's sample is divided : 2
- Pool size : 7
- Number of patients being pooled : 28

Representation of the matrix:

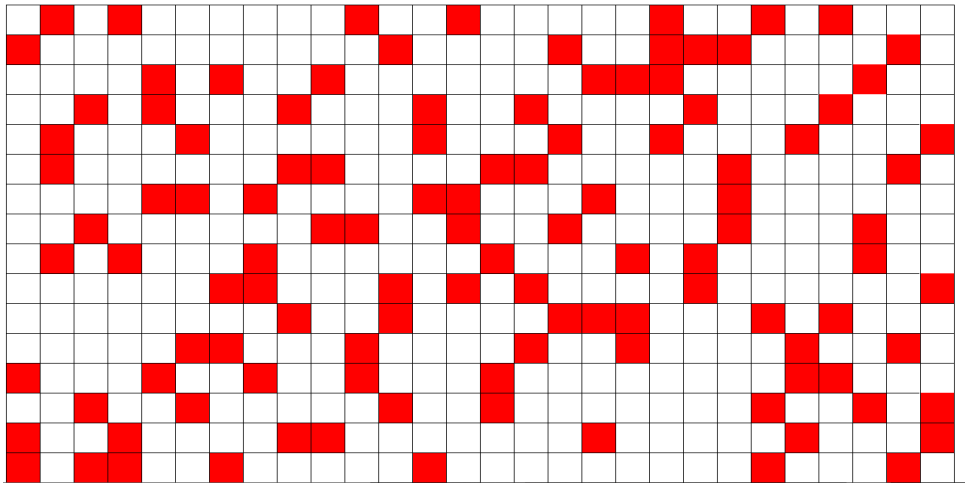


$p \rightarrow$	0.01%	0.1%	0.5%	1%	2%	3%
Mean AUC(in %)	99.999	99.99	99.999	99.972	99.81	99.43
Best (S_e,S_p) in curve (in %)	(100,99.99)	(100,99.997)	(100,99.987)	(100,99.665)	(100,98.7)	(100,97.22)

7.5.11 Design 1: (16,28,4)-BIBD aka B282

- Number of pools : 16
- Number of times a patient's sample is divided : 4
- Pool size : 7
- Number of patients being pooled : 28

Representation of the matrix:

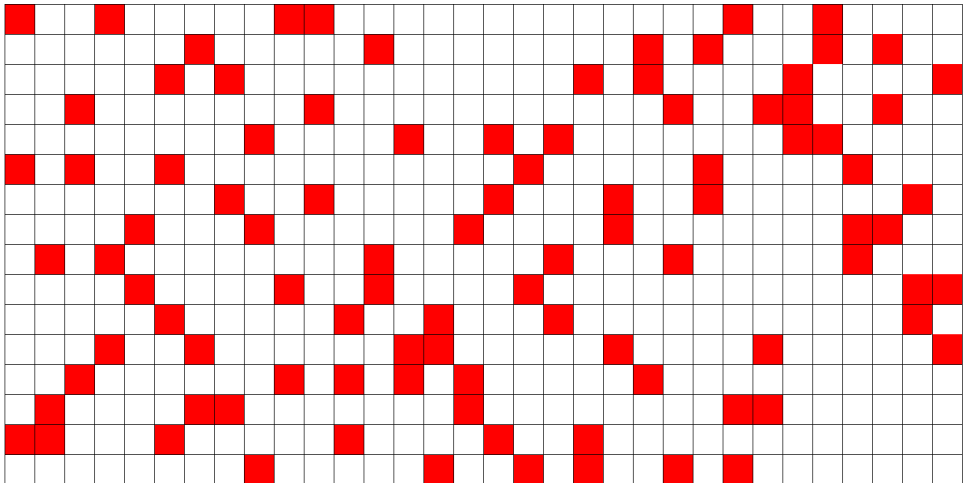


$p \rightarrow$	0.01%	0.1%	0.5%	1%	2%	3%
Mean AUC(in %)	100	100	99.999	99.998	99.982	99.93
Best (S_e,S_p) in curve (in %)	(100,100)	(100,99.999)	(99.99,999.989.35)	(99.91,99.88)	(99.55,99.486)	(99.032,98.75)

7.5.12 Design 1: (16,32,3)-BIBD aka B321

- Number of pools : 16
- Number of times a patient's sample is divided : 3
- Pool size : 6
- Number of patients being pooled : 32

Representation of the matrix:



$p \rightarrow$	0.01%	0.1%	0.5%	1%	2%	3%
Mean AUC(in %)	100	100	100	99.999	99.993	99.965
Best (S_e,S_p) in curve (in %)	(100,100)	(100,100)	(100,99.985)	(99.98,99.93)	(99.82,99.664)	(99.438,99.199)

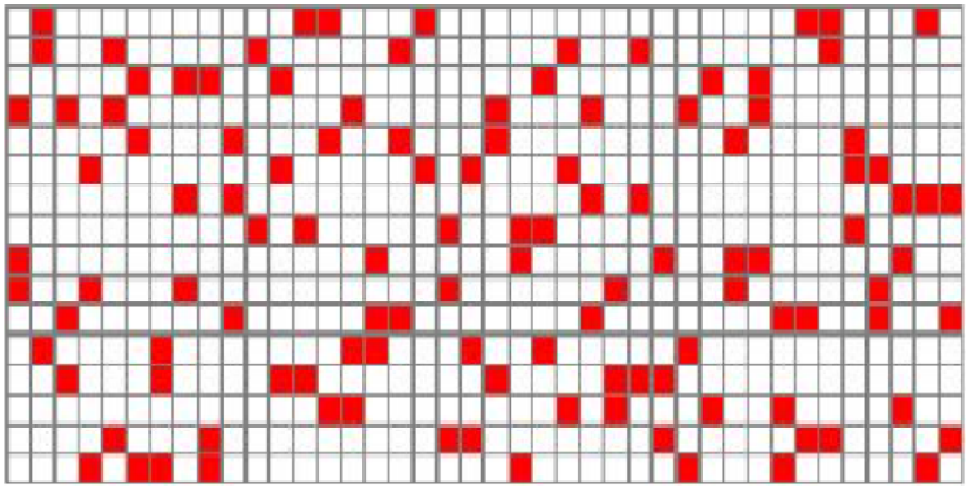
7.5.13 Working of the Tapestry matrix using our methodology

Tapestry had devised innovative pooling strategies using combinatorial Group Testing. We will use the 16×40 matrix and see how it fares with our algorithm.

The asymmetric nature of the design simply points to the fact that Tapestry haven't used any BIBDs in their algorithm, but came up with a different procedure to select designs.

- Number of pools : 16
- Number of times a patient's sample is divided : 3 in most cases. 2 people have their samples split into 2
- Pool size : 10 pools have size 7, 4 have size 8 and 2 have size 9
- Number of patients being pooled : 40

Representation of the matrix:



$p \rightarrow$	0.01%	0.1%	0.5%	1%	2%	3%
Mean AUC(in %)	100	99.99	99.999	99.993	99.94	99.79
Best (S_e,S_p) in curve (in %)	(100,100)	(100,99.999)	(99.95,99.92)	(99.8,99.72)	(99.25,98.95)	(98.65,97.71)

7.6 Comparison of a select few

The detailed analysis of each of the designs is helpful, as we can isolate the ones that perform better than the rest and could be a good practical implementation. After consultation with the mentors, 4 designs are of interest, not just for their performance, but also for their clinical utilities as discussed before They are:

1. (5,10,2) -BIBD aka B101
2. (4,12,2) -BIBD aka B122
3. (8,14,4) -BIBD aka B144
4. (16,24,4)-BIBD aka B242

We will tabulate our data into one table and compare the results. We also add Tapestry for reference, since it is well established as a pooling algorithm.

Design →	B101	B122	B144	B242	Tapestry
Number of pools	5	4	8	16	16
Pool Size	4	6	7	6	{7,8,9}
Number of times sample is divided	2	2	4	4	{2,3}
Number of people being pooled	10	12	14	24	40
Mean AUC(in %)					
p = 0.01%	100	99.995	100	100	100
p = 0.1%	99.99	99.95	99.999	100	100
p = 0.5%	99.99	99.75	99.994	99.99	100
p = 1%	99.998	98.934	99.97	99.999	99.999
p = 2%	99.981	99.77	99.87	99.996	99.993
p = 3%	99.35	96.82	99.71	99.98	99.965
p = 4%	99.86	94.39	99.455	99.95	99.893
p = 5%	97.75	92.11	97.155	99.89	99.762
Best (S_e,S_p) in curve (in %)					
p = 0.01%	(100,100)	(100,99.989)	(100,100)	(100,100)	(100,100)
p = 0.1%	(100,99.999)	(100,99.898)	(100,99.99)	(100,99.999)	(100,100)
p = 0.5%	(100,99.98)	(100,99.46)	(99.977,99.599)	(100,99.996)	(100,99.985)
p = 1%	(100,99.914)	(99.432,98.292)	(100,99.11)	(99.98,99.962)	(99.98,99.93)
p = 2%	(100,99.66)	(98.78,96.25)	(100,98.11)	(99.85,99.77)	(99.82,99.664)
p = 3%	(100,99.23)	(98.38,94.24)	(100,97.091)	(99.61,99.45)	(99.438,99.199)
p = 4%	(100,98.67)	(96.5,90.74)	(100,95.87)	(99.188,98.956)	(98.931,98.51)
p = 5%	(98,96.1)	(94.77,87.37)	(98,92.74)	(98.65,98.6)	(98.166,97.78)

Conclusion

We will wrap up our discussion with the protocol to be followed, if the above designs are to be used.

1. A wet lab technician collects samples of patients and accordingly chooses the designs, preferably from the 4 designs compared in the end since they performed the best.
2. The vector y of pool tests is then provided as input into the corresponding code snippet associated for that design. Note that other information such as the matrix and other parameters of the supporting functions are already entered.
3. The associated ROC curve, estimate $\tilde{x}$ and the best (Sensitivity,Specificity) points are provided as an output. The person chooses the threshold, either using the best point, or as per the clinician's demand, and estimates the binary vector by declaring all values below the threshold to be 0 and above to be 1.‘

References

[1] Austin,D,(2010,October),Pooling strategies for COVID-19 testing,*American Mathematical Society*,`http://www.ams.org/publicoutreach/feature-column/fc-2020-10`

[2] Suzen,M,(2016,August 29),Package ‘R1magic’,*The Comprehensive R Archive Network*,`https://cran.r-project.org/web/packages/R1magic/R1magic.pdf`

[3] Robin,et.al,(2021,January 13),Package ‘pROC’,*The Comprehensive R Archive Network*,`https://cran.r-project.org/web/packages/pROC/pROC.pdf`

[4] Brownlee,J,(2020,February 10),A Gentle Introduction to Threshold-Moving for Imbalanced Classification,*Machine Learning Mastery*,`https://machinelearningmastery.com/threshold-moving-for-imbalanced-classification/`

[5] Wikipedia,(2021,June 24),*Compressed sensing*,`https://en.wikipedia.org/wiki/Compressed_sensing`

[6] Davenport, M., Duarte, M., Eldar, Y., & Kutyniok, G, *Compressed Sensing: Theory and Applications*,Introduction to compressed sensing.(2012,November),1-64,DOI:10.1017/CBO9780511794308.002

[7] Ghosh,et.al,*A Compressed Sensing Approach to Pooled RT-PCR Testing for COVID-19 Detection*,**2**,(2021,April),248-264,DOI=10.1109/OJSP.2021.3075913

[8] Douglas R. Stinson,*Combinatorial Designs: Constructions and Analysis*,Springer-Verlag,New York.2004