Faculty of
Mathematics
and Informatics

VILNIUS UNIVERSITY
FACULTY OF MATHEMATICS AND INFORMATICS
MODELLING AND DATA ANALYSIS
MASTER'S STUDY PROGRAMME

OPTIMAL GROUP TESTING WHEN PREVALENCE IS VARYING

Master's thesis

Author: Lina Meškonytė
VU email address: lina.meskonyte@mif.stud.vu.lt
Supervisor: Associate professor Viktor Skorniakov

Vilnius

2022

Abstract

Group (pooled) testing is an alternative to individual (one-by-one) testing that can be used to efficiently (faster and cheaper) test a large set of objects (specimens/samples/individuals/etc.) when samples are pooled in right size groups. The common approach widely investigated in the literature requires to know the exact prevalence rate. However, it is not clear what to do when it is not known.

We aim to answer this question by numerically computing the group size which minimises the expected number of tests when there is no prior knowledge of prevalence (except assuming the majority of large tested population is negative/not infected). The optimisation problem is solved using minimax and average squared error loss functions - the optimal group size is calculated for a group of existing probabilistic pooled testing procedures. The research does not cover and could be extended to higher dimensional methods as well as combinatorial procedures, furthermore, not only the average (expected number of tests), but also variance could be taken into account.

Keywords: Group testing, unknown prevalence, optimisation problem.

Contents

Acknowledgement

The author is thankful for the high performance computing resources provided by the Information Technology Research Center of Vilnius University.

Notations

Throughout the thesis there are some key notations to remember:

$\mathbb{E}_A(N,p)$	expected total number of tests for a given N and p under procedure A
$\lfloor x \rfloor$ ($\lceil x \rceil$)	the largest (smallest) integer which is smaller (larger) than or equal to x (for x > 0)
N	group size
$N_{opt}(p)$	optimal group size
p	prevalence (probability of being positive)
p_U	Ungar's universal cut-off point (UCP), $p_U = \frac{3-\sqrt{5}}{2} \approx 0.381966$
S_e	sensitivity of the test (the probability an individual is categorised as positive given the individual is truly positive)
S_p	specificity of the test (the probability an individual is categorised as negative given the individual is truly negative)
$t_A(N,p)$	expected number of tests per person/specimen for procedure A in a group size of N, $t_A(N,p) = \frac{\mathbb{E}_A(N,p)}{N}$

1 Introduction

By definition, pooled testing means splitting up a task of identifying specific items or individuals into tests on grouped objects, instead of processing each one. If people or items are tested in right size groups – both time and money can be saved. Group testing has many applications, including medicine, engineering, biology and more. There are various schemes of group testing (GT), some are quite simple, such as the first ones described, e.g. by Dorfman in 1943, some are more complex. The mentioned GT scheme is executed in such a way: (a) having N blood samples, pool them and test the pooled sample; (b) in case pooled sample tests negative, finish; otherwise, retest each each patient individually (see fig. 1 for illustration). The purpose of using a GT scheme is to do as little testing as possible to identify all positive samples. The question is then - what should be the group size to achieve optimal scenario. First, expected number of test per person is calculated for fixed group size and prevalence. For Dorman's procedure the total number of tests is 1 with probability q^N ($q = 1 - p$) and $N + 1$ with probability $1 - q^N$. The expected number of tests per person is then defined as the total expected number of tests divided by the total number of tested persons:

$$t_D(N, p) = \begin{cases} 1 - q^N + \frac{1}{N}, & \text{for } N \geq 2; \\ 1, & \text{for } N = 1. \end{cases}$$

Next, for main schemes, including Dorfman's, optimisation problem is already solved for known p, thus the step is just to use the expression. For other schemes optimal group size can be found by checking all possible group sizes and taking the one with which $E(N, p)$ (which here and in the sequel stands for the total expected number of tests) is the smallest. In other words, if we know exact prevalence, we know optimal group size to use for grouped testing.

Group testing is not a new field, as mentioned above, there are lots of schemes that can be used to choose group sizes. The downfall usually is that one needs to know the probability to get infected in the whole population which quite often is not known specifically. The purpose of this paper is to provide a guidance that goes through existing schemes and suggests the group sizes for the cases when the prevalence is unknown.

The work carried out includes some schemes for gold-standard tests and a few with the presence of test error - Dorfman's procedure (D), Extension of Dorfman's procedure (D'), Sterrett's procedure (S), Halving procedure (H), Hierarchical algorithm (D2), Squared array (A2), Generalisation of 3L extension to multiple dimensions (ML). All these GT schemes are described in section 2. Similar attempts were made for some more procedures, but due to lack of resources those were left for the future work. Besides preparing the mentioned guideline, some additional results are gathered through the work, e.g. for the gold standard tests one can use the summarised plot (in fig. 9) of p (prevalence) to $t(N_{opt}, p)$ (expected number of tests per person in an optimal group size) to see which procedures would be more efficient (which curve is the lowest) in specific p intervals. Furthermore, some procedures become less efficient than the individual group testing at some point, that would be visible in the plot when the curve of specific procedure reaches 1, i.e. $t(N_{opt}, p) = 1$.

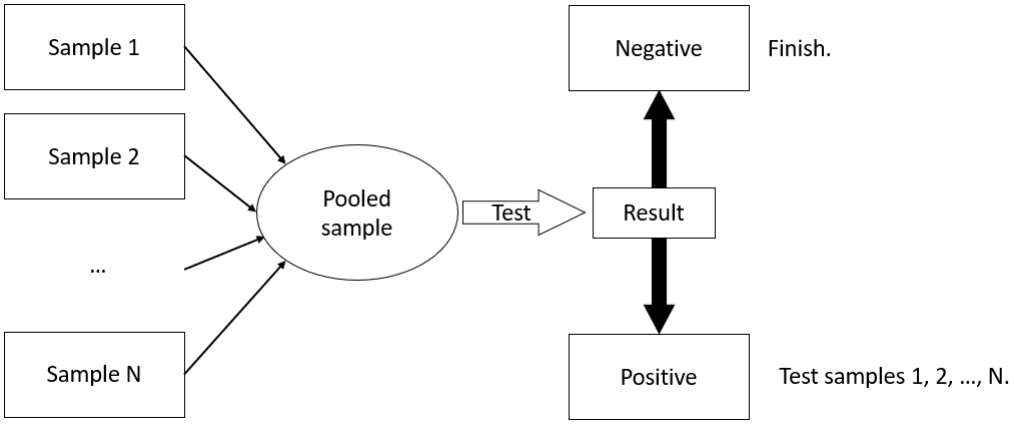

Figure 1: Dorfman scheme

The remaining structure of this paper is organized into 4 parts. The literature review outlines the detailed descriptions and methodology of procedures mentioned above, also covers 2 loss functions used in work. Assumptions, approach and theoretical steps on how the problem is solved are described in scientific research part. The main section - practical application part - includes all calculations, simulations and results. The summary of which is then given in conclusions section.

2 Literature overview and methodology

Firstly, it is helpful to have 2 main metrics in mind - $t(N,p)$ and $N_{opt}(p)$. Let N be the size of a group of patients on which we plan to try applying a group testing procedure X and let T_N is denote the total number of tests applied for the identification of the subset of all infected (it is a random number). The main metric used in GT tasks is the expected number of tests per person as mentioned in introduction: $t(N,p) = \frac{\mathbb{E}\, T_N}{N}$. The question to answer is what should be the value of N so that we would test optimally, i.e. for fixed p, the $t(N,p)$ value to be the smallest. The right side of equation is dependant both on N and p and often can be expressed analytically for a particular scheme. If p is known or can be quite accurately estimated from the sample data, then the problem can be simplified to finding the minimum of a continuous function $N \mapsto t(N,p)$ where $N \in (1,\infty)$. That value of N will then be called the optimal group size and denoted as $N_{opt} = N_{opt}(p)$ or $N_{opt}^{(X)} = N_{opt}^{(X)}(p)$ to emphasise specific procedure. Note that for individual testing $t(N,p) = 1$.

Secondly, in order to keep the main expressions on all procedures gathered in one place for easy access later on, they are listed prior to the review of investigated articles. Therefore, it is recommended for the reader to glance over them and then skip to the articles described below (section 2.4) where then all of the listed methodology is referenced step by step.

2.1 Procedures for gold-standard tests

In short, each section below describes a different GT procedure, when the gold standard tests are assumed and binomial assumptions hold. First, the steps are described on how the method would be carried out, then the formula of the expected number of tests per person is listed for given procedure and finally, if available, the analytical expression of optimal group size for the known p.

2.1.1 Dorfman's procedure (D)

Test pooled sample, if it tests negative - finish, if positive - test the samples in the pool individually.

$$t_D(N,p) = \begin{cases} 1 - q^N + \frac{1}{N}, & \text{for } N \geq 2; \\ 1, & \text{for } N = 1. \end{cases}$$

In the equation above $q = 1 - p$. If $p < 1 - 1/3^{1/3} \approx 0.31$ then N_{opt} for scheme D is equal to $1 + \lfloor p^{-1/2} \rfloor$ or $2 + \lfloor p^{-1/2} \rfloor$, otherwise 1.

2.1.2 Extension of Dorfman's procedure (D')

Extension of D - compared to scheme D - not to test last individual in group if all negative before.

$$t_{D'}(N,p) = 1 - q^N + \frac{1}{N} - (1/N)(1-q)q^{N-1}$$

N_{opt} for scheme D' is equal to $\lfloor p^{-1/2} \rfloor$ or $\lceil p^{-1/2} \rceil$ for $0 < p < p_U = \frac{3-\sqrt{5}}{2}$.

2.1.3 Sterrett's procedure (S)

S1) Test initial pool spanning N patients; given negative response, finish; given positive response, proceed to S2;

S2) test one by one until first positive; proceed to S3;

S3) retest the remaining pool; given negative response, finish; given positive response, proceed to S2 again.

$$t_S(N,p) = \frac{1}{N}\left[2N - (N-2)q - \frac{1-q^{N+1}}{1-q}\right]$$

N_{opt} for scheme S is equal to $\lfloor\sqrt{2/p}\rfloor$ or $\lfloor\sqrt{2/p}\rfloor + 1$ or $\lfloor\sqrt{2/p}\rfloor + 2$ for $0 < p < p_U$.

2.1.4 Halving procedure (H)

H1) Test pool spanning all patients; given negative response, finish; given positive response, proceed to H2;

H2) divide the whole set of patients into two approximately equal parts. Apply H1) to each part recursively.

$$t_H(N,p) = \frac{1}{N} + 2\sum_{k=1}^{\log_2 N} \frac{1-q^{2^k}}{2^k}$$

N_{opt} for scheme H is equal to either $\lfloor 1/(2\log_2(1/q)) \rfloor$ or $\lfloor 1/(2\log_2(1/q)) \rfloor + 1$ for $p \in (0,1)$.

2.1.5 Squared array without master pooling (A2)

Let $N = n^2$, then the A2 operation is as follows:

- n^2 specimens are placed on $n \times n$ matrix, $n \in \{2,3,4,\ldots\}$;
- $2n$ pools of size n defined by row–column subsets are tested;
- in case all $2n$ pools test negative, A2 finishes; otherwise, patients corresponding to intersections of positively testing rows and columns are retested individually.

$$t_{A2}(N,p) = \frac{2}{N} + 1 - 2q^N + q^{2N-1}$$

For all $p \in (0, p_*)$, where $p_* \approx 0.252$, n_{opt} ($N_{opt} = n_{opt}^2$) for scheme A2 (based on [7]) is equal to one of the three expressions: $\left\{\left\lfloor \frac{1}{p^{\frac{2}{3}}} + \frac{1}{2p^{\frac{1}{3}}} + 3p^2 + 0.2 \right\rfloor + i : i = 0,1,2\right\}$

2.1.6 Generalisation of 3L extension to multiple dimensions (ML)

Description of the procedure is based on 3L algorithm. Begin with placing $L \times M \times K$ specimens in a cubic array. Then in stage 1, for each of the $M \times K$ dots on the front face of the cube, test the group of size L on the line that is at 90 degrees to this face. Repeat for the $L \times K$ groups of size M, lying on similar lines perpendicular to the top and the bottom faces and for the $L \times M$ groups of size K located on lines perpendicular to the left and right faces. In stage 2, test individually each sample that is located at the intersections of three lines that all tested positive in stage 1. See illustration in fig. 2. Then take extension of described method to multiple dimensions and finally, generalise to d-dimensional cube with sides of length s. The expected number of tests per person provided below.

$$t_{ML}(d, s, p) = ds^{-1} + p + q(1 - q^{s-1})^d$$

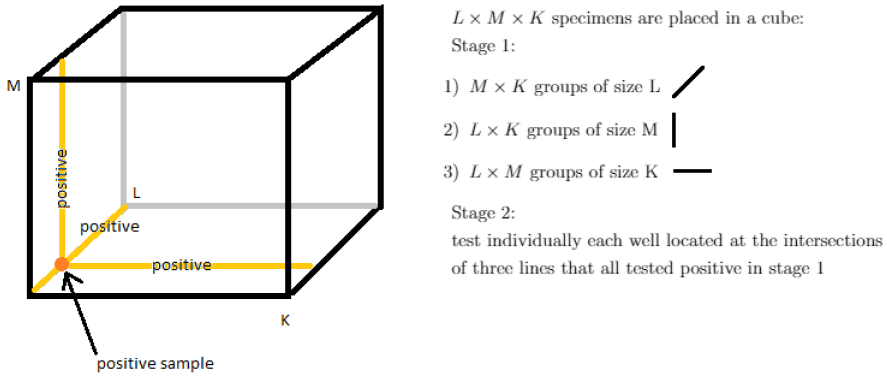

Figure 2: 3L scheme

2.2 Procedures in the presence of test error

Likewise, each section below describes some more pooled testing algorithms, although this time they are described for tests that are known to have sensitivity and/or specificity below unit, meaning that a sample may not test positive even though it is or in other scenario, the test is not able to fully distinguish the right classification, e.g. if test for one virus would result in positive even though the individual is infected, but only with another virus.

2.2.1 Hierarchical algorithm (S=2)

It is an extension of Dorfman procedure: one needs repeatedly divide positive pools into smaller, non-overlapping sub-pools until all positive samples are individually tested. Hierarchical algorithm with S stages (DS) is more general version of this (formulas are available in the cited paper). The expected number of tests for 2-stage Hierarchical algorithm is given below.

$$t_{D2}(N,p) = \frac{1}{N} + (1 - S_p)q^N + S_e(1 - q^N),$$

where S_p, S_e stand for the test's specificity and sensitivity respectively.

2.2.2 Square array without master pooling

The procedure is similar to the one with perfect tests (section 2.1.5), but instead of only testing samples in the intersection of positive pools, but also a full row (column) is tested individually if there are no positive columns (rows) intersecting it. Let $N = n^2$, then:

$$t_{A2}(n,p) = 2/n + g(n) + 2h(n)$$

$$g(n) = S_e f(n) + (1 - S_p - S_e)q^n f(n-1)$$

$$f(n) = (1 - S_p)q^n + S_e(1 - q^n)$$

$$h(n) = \sum_{c=0}^{n} \{\gamma_0(c)\beta_0(c) + \gamma_1(c)\beta_1(c)\}$$

$$\gamma_0(c) = (1 - S_p)(1 - S_e)^c S_p^{n-c}$$

$$\beta_0(c) = \binom{n}{c} q^{n^2 - cn + c}(1 - q^{n-1})^c$$

$$\gamma_1(c) = S_e S_p^{n-c}(1 - S_e)^c$$

$$\beta_1(c) = \binom{n}{c} q^{n^2 - cn}(1 - q^n)^c - \beta_0(c)$$

$$\binom{n}{c} = \frac{n!}{c!(n-c)!}$$

where S_p, S_e stand for the test's specificity and sensitivity respectively.

2.3 Loss functions

Average number of "lost" tests when using not optimal group size is

$$Err(N,p) = t(N,p) - t(N_{opt}(p), p).$$

There are 2 main loss functions:

- minimax: $L_\infty(N) = \sup_{p \in (0, \frac{3-\sqrt{5}}{2})} Err(N,p)$
- average squared error: $L_2(N) = \frac{2}{3-\sqrt{5}} \int_0^{\frac{3-\sqrt{5}}{2}} Err^2(N,p)dp$

2.4 Overview of articles

To begin with, article [5] provides theoretical framework for comparing commonly used designs - Dorfman procedures D (provided as example in introduction, fig. 1) and D', Sterrett procedure S and defines optimal nested procedure R_1. This paper gives a unique perspective on GT, gathering literature on cases under infinite and finite populations, dynamic programming and coding theory, ILB (information lower bound) of efficiency. In particular, analytical expressions for the three before mentioned procedures presented in subsections 2.1.1-2.1.3 above were extracted from [5]. Moreover, the provided appendix and supplementary materials give some more insight on how some of the formulas/results are derived. Finally, even though the article concentrates on cases when p is known, the authors do provide a section, investigating robustness of procedures to the incorrect specification of parameter p and comparing the efficiency of main procedures when only an upper bound U for p is known. They do it by taking a few specific p values from the interval and investigating which procedure has the lowest expected number of tests per person.

The earlier paper [4] by the same authors has a quite similar goal to our work - find optimal group size for unknown p (Minimax solution and Bayesian solutions with Uniform and Jeffrey's prior distributions), but its coverage is narrower: it is restricted to Dorfman procedure. Despite that, the paper is important because it suggests the way of finding N_{opt} when the prevalence is unknown and we adopt the suggested approach in our thesis. In the process of describing the loss functions, there are a few important facts gathered about the behaviour of functions:

- $N_{opt}(p)$ is a non-increasing function of p and $N_{opt}(p) \to \infty$ as $p \downarrow 0$.
- $t(N, p)$ is a non-decreasing function of p for any N.
- $t(N_{opt}(p), p)$ is a non-decreasing function of p and $t(N_{opt}(p), p) \to 0$ as $p \downarrow 0$.

The article uses a loss function that is defined as the difference between expected number of tests and expected number of tests under the optimal scenario (with optimal group size). Minimax group size (optimal group size for minimax solution) is then defined as a group size that minimizes the largest loss. The Bayesian solution is simply achieved by choosing a prior distribution for p and then integrating the loss function, multiplied by density function of chosen distribution. Contrary to this article, we prefer to call the end points of these solutions as loss functions and the loss function from article as the error function - the average number of "lost" tests when using not optimal group size. Such terminology is used going forward, see overview in subsection 2.3. Lastly, in our work we will use the result of optimal group size for Dorfman procedure with minimax approach to confirm our first result.

Even though the article [6] by a few Lithuanian authors does not propose any novel schemes, it compares individual, Dorfman and Halving procedures, including gain behaviour. Gain shows on average how many tests per person are "saved" using specific group testing scheme compared to individual testing $G_p = 1 - t(N_p)$ and it shows in which intervals of prevalence which procedure is more efficient (the gain is bigger). In addition to providing formula of calculating the average

number of tests per person for Halving scheme, they obtained the expression for the optimal group size (see it in section 2.1.4). Again, it is not the first article to mention the Ungar's cut off point (UCP or p_U), which is the value of prevalence with which individual testing becomes more efficient than any GT procedure. Therefore, before applying any GT procedure, it is important to assume $p < p_U = \frac{3-\sqrt{5}}{2} \approx 0.38$. Moreover, authors also talk a bit about the scheme specific upper bound of p with which individual testing is more efficient than that specific GT procedure. This value can be different for each scheme, but of course it is not bigger than the universal value of Ungar's cut off point. This specific value can be obtained by checking when $t(N, p)$ reaches 1 (the average number of tests per person in individual testing).

Another article [1] about gold standard tests is written in more "biological" approach though, authors talk about "assaying wells in a microtiter plate" - analysing samples/containers in a microplate usually containing $8 \times 12 = 96$ of these wells. Consequently, there are some explanations on how to adapt described methods for that. Moreover, this is different from other described articles by the view on considering slightly different definition of efficiency - maximising the expected number of positive wells isolated per assay performed rather than minimising the expected number of tests per person. There is a clear relationship between the two: $\eta = \frac{p}{t(N,p)}$ where η is the described efficiency from this article. The paper argues that if total elapsed time is critical, then many laboratory technicians can work in parallel - which can be achieved by restricting only to two stage procedures and then all assays that belong to the same stage can be performed simultaneously. Basic matrix method (BMM) and it's extensions are described and compared against each other, namely - The Three-Dimensional Planar Method (3P), The Three-Dimensional Linear Method (3L), Generaliation of 3L extension to multiple dimensions (ML). For the expected number of tests per person of the latter, see section 2.1.6.

On the contrary to already described ones, the next article [2] elaborates on procedures when tests are not perfect - have inadequate specificity or sensitivity (i.e. in the presence of error). It also suggests that GT can not only be more efficient than individual testing, but also provide more accuracy. The authors have described Hierarchical (DS) and Squared Array (A2) testing algorithms (see formulas in sections 2.2.1-2.2.2) and in addition to providing the expressions for expected number of tests, they have also defined a few useful metrics that can help calculate the expected tradeoffs of using GT methods for a particular application. Namely, positive predictive value (PPV), negative predictive value (NPV), per-family error rate ($PFER$) and type II ($PFER_2$), per-comparison error rate ($PCER$) and type II ($PCER_2$). The paper also gives an insight on the variability, in particular, if two GT procedures have similar expected number of tests, the one with smaller variance is preferable. Although the mentioned procedures are described rigorously, the authors mostly concentrate on specific examples, i.e. where besides sensitivity and specificity, the prevalence is known precisely too.

Finally, article [3] investigates a task that is different than in most articles when talking about group testing. Instead of identifying all positive samples, one needs to estimate the prevalence rate (with known sensitivity and specificity of a test used) and do that with as little testing as possible,

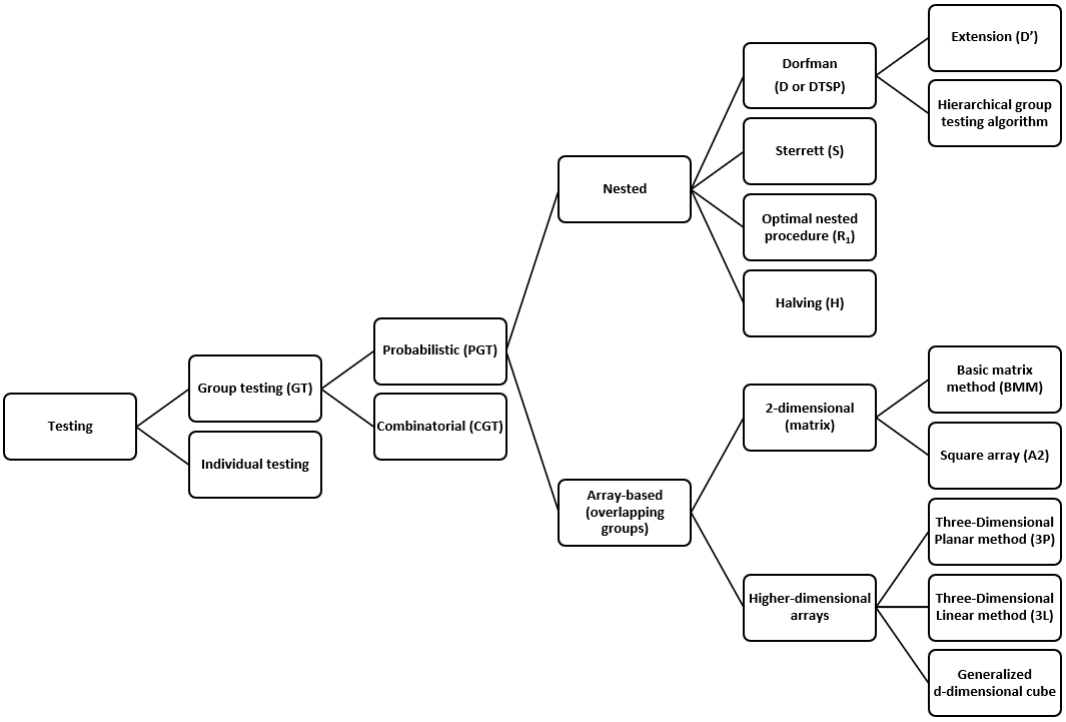

Figure 3: Classification of group testing schemes

that means that retesting individuals is not necessary anymore. The authors have proved that for estimating prevalence in the presence of test error it is more efficient to use a group testing method rather than individual or random sampling design.

To summarise, in a situation where some testing has to be done, depending on the specifics and available information, there is a lot to choose from - see the classification of group testing schemes (fig. 3). Note that the tree is not exhaustive, only schemes from reviewed literature are included. Group testing is usually approached from either probabilistic or combinatorial side (using combinatorial techniques instead). This paper only considers probabilistic schemes. When it is known or assumed that the prevalence is higher than the Ungar's universal cut-off point p_U - the choice of individual testing versus group testing is the right one (none of group testing procedures will be more efficient than 1-by-1 testing). Then, based on if the scheme suggests dividing the samples into not overlapping groups, they can be classified as nested, compared to array based schemes in which groups being tested do overlap. Specific group testing procedures used are described above in the subsections 2.1 and 2.2.

3 Scientific research

This section describes the theoretical approach to achieve the goal. Before that, below are listed assumptions for the work.

3.1 Assumptions

(i) The N tested items are independent.

(ii) Each item is bad with the same constant probability $p \in (0, 1)$.

(iii) The test kit is perfect (or with sensitivity and specificity lower than 1 and in the sequel denoted as S_e and S_p respectively) and there is no dilution effect (test works equally good for both grouped and individual testing).

(iv) p is distributed uniformly (when no prior information is available). Note that due to Ungar's result, p is assumed to be supported only in interval $0 < p < p_U$.

(v) Infinite (large) population is assumed.

The achieved results are meaningful only if all of the assumptions above are satisfied.

3.2 Steps

Having the schemes and their formulas gathered, assumptions listed, it can now be described, how they have been used to get the wanted results - optimal group size for a scheme when prevalence is unknown. The plan of action is quite similar for all the schemes:

1. Find optimal group sizes when p is known (for some procedures this is already available);
2. Calculate error and loss functions;
3. Find optimal group size when p is unknown – group size with which loss function is at its minimum value;
4. Plot, save results.

The process is described by taking already discussed Dorfman scheme example. As already noted in section 2.1.1, $N_{opt}(p)$ is equal to either $1 + \lfloor p^{-1/2} \rfloor$ or $2 + \lfloor p^{-1/2} \rfloor$. Decision on which value to take is based on taking specific p value and calculating $t_D(N, p)$ values (smaller value of the two corresponds to the optimal configuration). To illustrate, if $p = 0.1$, then

$$1 + \lfloor p^{-1/2} \rfloor = 1 + \lfloor 0.1^{-1/2} \rfloor = 1 + 3 = 4$$

$$2 + \lfloor p^{-1/2} \rfloor = 2 + \lfloor 0.1^{-1/2} \rfloor = 2 + 3 = 5$$

Then, using formula for expected number of tests per person, it follows that $t_D(4, 0.1) \approx 0.5939$ and $t_D(5, 0.1) \approx 0.6095$. Therefore, $N_{opt}(0.1) = 4$.

Next is the question what to do when p is unknown, e.g. when prevalence changes frequently and estimation from recently gathered sample data might be already outdated. Firstly, let us define $Err(N, p)$ function as in section 2.3, which shows how many tests on average are lost when prevalence is equal to p and testing is carried out by groups of size N rather than of optimal group size. Clearly, $Err(N, p) \geq 0$ for all $p \in \left(0, \frac{3-\sqrt{5}}{2}\right)$. Assuming that p is unknown, one needs to employ a loss function to minimise function Err in the whole interval of possible p values. There are two of them mentioned in the same section 2.3 - first one (minimax) measures the largest possible error from the optimal configuration, the second one considers the average squared error from optimality when each of the p value is given equal weight, in other words when p has a Uniform distribution in interval $p \in \left(0, \frac{3-\sqrt{5}}{2}\right)$. Let us note that $N_{opt,\infty}$ stands for N_{opt} obtained from loss function L_∞ and likewise $N_{opt,2}$ - N_{opt} from L_2.

As illustration, for Dorfman scheme $Err(N, p) = 1 - q^N + \frac{1}{N} - \left(1 - q^{N_{opt}} + \frac{1}{N_{opt}}\right)$, where $N_{opt}(p) \in \{\lfloor p^{-1/2} \rfloor + i : i = 1, 2\}$ and one of the two values is chosen for each p value individually (as in example above). To summarise, in order to find $N_{opt,\infty}$ one needs to insert $Err(N, p)$ expression above to L_∞ and then find the minimum (and accordingly use L_2 for $N_{opt,2}$). Note that as a constant doesn't affect the minimum, multiplier $\frac{2}{3-\sqrt{5}}$ can be skipped before the integral in all calculations for L_2. Also, for some procedures, the cut point p_U defining interval where application of that GT procedure makes sense is less than $\frac{3-\sqrt{5}}{2}$. In such cases, we replace $\frac{3-\sqrt{5}}{2}$ by p_U.

3.3 ML - extended description and derivation of optimal group size

Before moving on to practical application part, it is important to describe the additional theoretical side of ML procedure.

To begin with, let's investigate the function $f(d, s) = \frac{p}{\frac{d}{s}+p+q(1-q^{s-1})^d}$, where s, d are variables ranging in $[2, \infty)$ whereas $p \in (0, 1)$ (and hence $q = 1 - p$) is a fixed constant. The function $f(d, s)$ is the efficiency $\eta = \frac{p}{t(N,p)}$ for ML procedure, as defined in one of the reviewed articles [1]. Optimal configuration is achieved by maximising the mentioned efficiency or minimising $t(N, p)$, i.e. expected number of tests per person/specimen. Thus, the extremums of $t(N, p)$ are equivalent to the extremums of function $f(d, s)$. Then we need to create an algorithm for calculating extremums of function f. We start by calculating it's partial derivatives:

$$\begin{aligned} f'_d &= \frac{-p}{\left(\frac{d}{s} + p + q(1-q^{s-1})^d\right)^2} \cdot \left(\frac{d}{s} + p + q(1-q^{s-1})^d\right)'_d \\ &= \left[\left(\frac{d}{s}\right)'_d = \frac{1}{s}, \left((1-q^{s-1})^d\right)'_d = (1-q^{s-1})^d \cdot \ln(1-q^{s-1})\right] \\ &= \frac{-p(\frac{1}{s} + q(1-q^{s-1})^d \ln(1-q^{s-1}))}{\left(\frac{d}{s} + p + q(1-q^{s-1})^d\right)^2} \end{aligned}$$

$$
\begin{aligned}
f'_s &= \frac{-p}{\left(\frac{d}{s}+p+q(1-q^{s-1})^d\right)^2}\cdot\left(\frac{d}{s}+p+q(1-q^{s-1})^d\right)'_s \\
&= \left[\left(\frac{d}{s}\right)'_s=\frac{-d}{s^2},\left((1-q^{s-1})^d\right)'_s=d(1-q^{s-1})^{d-1}\cdot(-q^{s-1})'_s=d(1-q^{s-1})^{d-1}(-q^{s-1})\cdot\ln q\right] \\
&= \frac{p\left(\frac{d}{s^2}+d(1-q^{s-1})^{d-1}q^{s-1}\ln q\right)}{\left(\frac{d}{s}+p+q(1-q^{s-1})^d\right)^2}
\end{aligned}
$$

We are searching for critical points in solution interval of system $f'_d=0$, $f'_s=0$. This system is equivalent to the one below (let's equate f'_d and f'_s to 0 and multiply both sides of equations by denominator $\neq 0$ and p^{-1}):

$$
\begin{cases}
\frac{1}{s}=-q(1-q^{s-1})^d\ln(1-q^{s-1}) \\
\frac{1}{s^2}=-q^{s-1}(1-q^{s-1})^{d-1}\ln q
\end{cases}
$$

If we divide first equation by the second (divide left-hand side by left-hand side and right-hand side by right-hand side separately), we get:

$$
s=\frac{(1-q^{s-1})}{q^{s-2}}\frac{\ln(1-q^{s-1})}{\ln q} \tag{1}
$$

This equation (1) can be solved with iteration methods, but to do that one should first provide the first/initial approximation point of this function. However, it makes sense to rearrange this expression, considering the fact the if we take $q\approx 1(p\approx 0)$ then calculating s can give substantial errors and/or even turn out to become an unstable iteration algorithm when raising it to large power.

Let us denote $q^{s-1}=x\in(0,1)$. Then the last equation can be rewritten as:

$$
\begin{aligned}
(s\ln q)q^{s-1} &= q(1-q^{s-1})\ln(1-q^{s-1}) \\
(\ln q+\ln x)x &= q(1-x)\ln(1-x) \\
x\ln q &= q(1-x)\ln(1-x)-x\ln x \\
x &= \frac{1}{\ln q}(q(1-x)\ln(1-x)-x\ln x) \qquad (2)
\end{aligned}
$$

Although we can not find a close form analytical solution of the latter equation (2), it can be efficiently solved numerically. Note that solutions are fixed points of $g(x)=\frac{1}{\ln q}(q(1-x)\ln(1-x)-x\ln x)$ (see fig. 4). Thus, we can employ bisection method. Let $h(x)=x$. Below we outline main steps of our algorithm.

1) At first we choose a point x_0 for which $g(x_0)>h(x_0)$.

2) We then choose a point $x_1<x_0$ for which $g(x_1)<h(x_1)$ and find the first ("left") root with

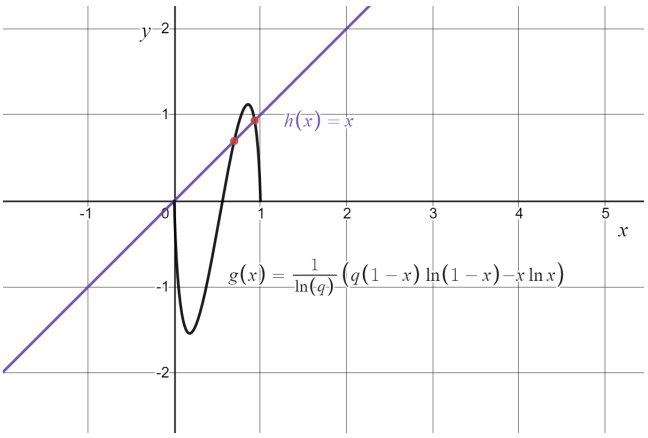

Figure 4: Graph of functions $h(x)$ and $g(x)$

bisection method - by searching for the zero of $h(x) - g(x)$ in $[x_1, x_0]$. Point x_1 can be set to inflection of $g(x)$ function (it can be solved analytically from $g''(x) = 0$ and is equal to $\frac{1}{1+q}$).

3) Next we choose a point $x_2 > x_0$ for which $g(x_2) < h(x_2)$ and find the second "right" root with bisection method - this time searching in the "zero" interval $[x_0, x_2]$ of function $h(x) - g(x)$. Point x_2 can be equated to 1 (it can be showed that $1 - \mathrm{e}^{-3}$ also fits).

Remark 1. *Point x_0 can be chosen with bisection method by searching for the zero of $g'(x)$ in $[x_1, x_2]$ of function $g'(x)$. Such algorithm is used in the R program (see appendix in page 46).*

Remark 2. *Equation 2 (and also the initial system) has a solution only when $q \in (q_0, 1)$, where $q_0 \approx 0.875$, see fig. 5.*

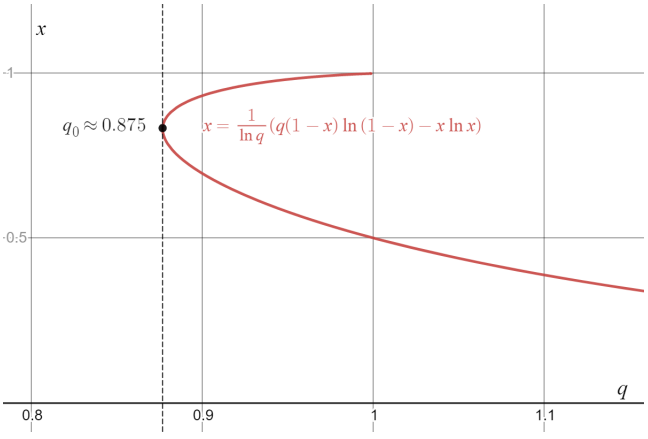

Figure 5: Graph of set of solutions

4 Practical application

In brief, practical realisation strategy of section 3.2 for each scheme includes:

1. taking a partition $P = (p_1, p_2, \ldots, p_m)$ of an interval $\left(0, \frac{3-\sqrt{5}}{2}\right)$ with same small enough sized step throughout the whole interval, e.g. with accuracy of 10^{-6}:
   $P = (10^{-6}, 2 \cdot 10^{-6}, 3 \cdot 10^{-6}, \ldots, \frac{3-\sqrt{5}}{2})$;
2. also taking a big $N \in \mathbb{N}$ and denoting $A_N = \{1, 2, \ldots, N\}$;
3. then calculating $L_\infty = \max_{p \in P} Err(n, p)$ and $L_2 = \frac{1}{m} \sum_{j=1}^{m} Err^2(n, p)$ for each $n \in A_N$;
4. finishing by obtaining $N_{opt,\infty} = \text{argmin}_{n \in A_N} L_\infty(n)$ and $N_{opt,2} = \text{argmin}_{n \in A_N} L_2(n)$.

4.1 Supplementary notes

Furthermore, there are quite a few additional important things to mention. Firstly, the choice of the step size of p (i.e. accuracy $= 10^{-6}$) can be explained along the lines of the fact that it has to be arbitrarily small for the final results to be accurate - the function of expected number of tests per person is continuous, thus the goal is to take small enough steps (sub-intervals) of p so that the values don't change over each sub-interval, i.e. $|t(N, p_i) - t(N, p_{i+1})|$ should be near 0 for each $i = 1, 2, \ldots, m-1$. It is also important though not to create too big partition P for the computations (it is nearly 400 thousand p values in P with accuracy of 10^{-6}).

Secondly, as mentioned already, not all GT schemes have analytical expression for the optimal group size when p is known. Hence, we need a way to find $N_{opt}(p)$ for each p in the partition P before the next steps of the process. For each $p \in P$ we need to find the group size N with which expected number of tests per person $t(N, p)$ is at it's smallest value, i.e.

$$N_{opt}(p) = \underset{N \in [2, \infty)}{\text{argmin}}\, t(N, p)$$

for fixed $p \in P$.

However, during the process of computations in R it appeared that the functions available are not efficient to use: both `functClust::argmin` and `which.min` require a vector of values as input (to use it one would need to find $t(N, p)$ for all $N \in [2, \infty)$ and save all values in format of a vector) and the function `42n4/dmr.util::arg.min` as input takes the arguments, function and number of how many top minimum values to output. The latter sounds promising, but upon investigation it appears that instead of only saving the best values in progress, it first calculates the function for all values, then sorts them smallest to biggest (meaning all the values are kept in working memory) and only then the first value of ordered list is outputed. The exact specifics of how the function is built makes it run slowly. In our case, it has to be utilised quite many times - for each $p \in P$.

For this reason, we have built a function ourselves. An improvement would already be if the function would go through values of N and only save the N and $t(N, p)$ if the latter one is smaller than

the last saved one, eventually, after checking all values of N, outputting the last saved N. Rather with settling at this point, we have made use of the known fact that $N_{opt}(p)$ is a non-increasing function of p (behaviour of functions covered in section 2.4 from article [4]). Namely, having arranged partition $p_1 < p_2 \ldots < p_m$ and after finding $N_{opt}(p_1)$, we have looked for $N_{opt}(p_i), i = 2, \ldots, m$ in the set $\{1, 2, \ldots, N_{opt}(p_{i-1}) + 1\}$. To justify our choice, several experiments were carried out. In these experiments, comparing the runtime of built-in `arg.min` and our own functions, we have discovered that, for small p's, the runtime reduces approximately 3 times.

Continuing with comments on the realisation strategy, although generally all the p values from the interval $\left(0, \frac{3-\sqrt{5}}{2}\right)$ should be used in the calculations listed above, particular procedures have an upper bound of p (let's denote it p_X where X is specific GT scheme) above which the expected number of tests per person is not defined (e.g. for the extension of generalised ML scheme $p_{ML} \approx 0.125$) or/and exceed 1 (e.g. for Dorfman scheme $p_D < 1 - 1/3^{1/3} \approx 0.31$). Consequently, interval of p for calculation of loss functions to find N_{opt} should be reduced to $p \in (0, p_X)$. The value of p_X is retrieved either from analytical expressions if available or selected after $t_X(N_{opt}(p), p)$ is calculated for each p.

The said above applies to all schemes, but for some of them has to be adjusted / additional steps added, for instance if the presence of test error is assumed (the case when sensitivity S_e and specificity S_p parameters of particular test are known, these values are fixed), then the implementation of the same steps is needed using those fixed values, meaning that to have all situations covered, the steps will be executed for each combination of S_e, S_p pair. To be precise, the calculations are performed with all combinations of $S_e, S_p \in \{0.9, 0.91, \ldots, 0.99, 1\}$. The interval $S_e, S_p \in [0.9, 1]$ is chosen based on what is acceptable to use as tests, and so it is rare to see tests with lower than 90% sensitivity or specificity. The accuracy of 0.01 is taken based on the balance between:

- the fact that the sensitivity and specificity are usually declared on the percentage level or sometimes even within a range of a few,
- and substantially increasing computational time when smaller step is considered.

4.2 Results

This section describes the achieved results - Dorfman scheme presented with more detailed and intermediate results, then all of the procedures for gold standard tests with summarised results and their comparison, and lastly, final results for the schemes in the presence of test error.

4.2.1 Dorfman (D)

To begin with, as mentioned already, implementation is analogous for all procedures. Similarly as in multiple sections above, we give a more detailed description of Dorfman procedure results first to better understand how the goal of the paper is achieved. Following the steps of practical application (section 4), we start by calculating the optimal group size $N_{opt}(p)$ when p is known to

be able to then calculate the expected number of tests per person in optimal configuration for each value of p, see the latter depicted in fig. 6.

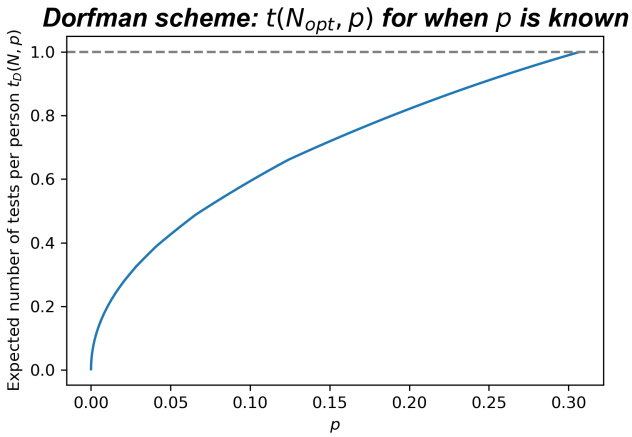

Figure 6: Dorfman procedure - Expected number of tests per person when p is known

Using the obtained values, we are able to compute the error Err function amounts for each p and $N \in A_N$. Afterwards, values of the two loss functions (L_∞ and L_2) for each N can be retrieved, visually presented in fig. 7.

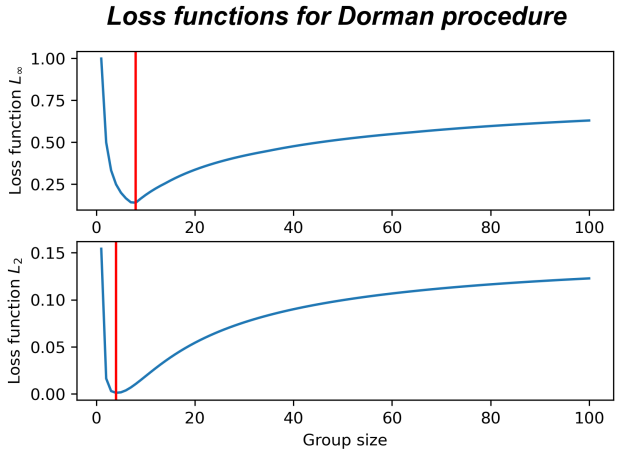

Figure 7: Dorfman procedure - Loss functions (red marked are N_{opt} values)

As we can see, both loss functions are unimodal (defined as if a function has a global maximum or global minimum) and the group sizes in the minimum points of each can be taken then - $N_\infty = 7$ and $N_2 = 3$ (marked in the plot with vertical red line). Both of those group sizes minimise the expected number of tests per person, but by calculating the area under the plot of $t_D(N^*_{opt}, p)$ with fixed $N = N^*_{opt}$ we can know which of the two loss functions results is better (expected to overall need less tests used to find all positive samples/individuals). Here N^*_{opt} stands for N_{opt} when p is unknown and $N^*_{opt} \in \{N_\infty, N_2\}$ (results from each of the two loss functions). We can see from fig. 8 that the area under the plot of $t_D(N^*_{opt}, p)$ using loss function L_2 seems to be smaller, that is

confirmed with the numerical calculation: $area_\infty = 0.2379724 > 0.215348 = area_2$, where $area_\infty$ and $area_2$ stands for the the area under the plot of $t_D(N_\infty, p)$ and $t_D(N_2, p)$ accordingly.

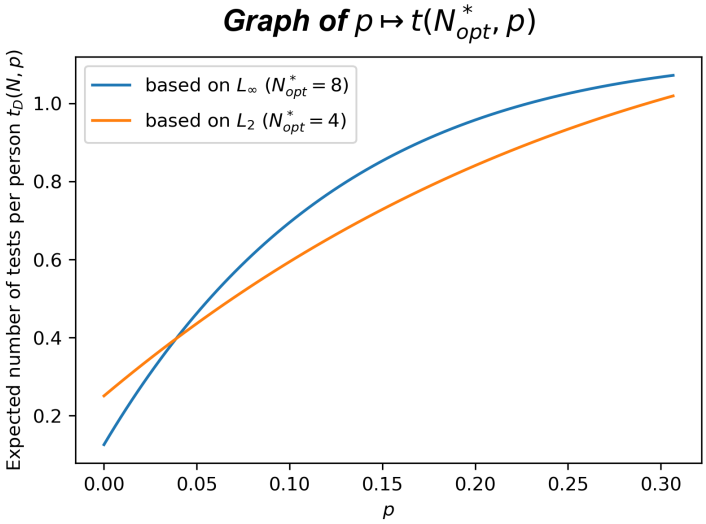

Figure 8: N^*_{opt} indicates the optimal group size, not dependant from p, that is found by minimising one of the loss functions.

In conclusion, we were able to find the optimal group size for Dorfman scheme when p is unknown, i.e. $N^{(D)}_{opt} = 4$. Similar approach is used for the remaining procedures with gold standard tests, see main results in the next section below.

4.2.2 Results for gold standard tests procedures

Firstly, see fig. 9, in which the expected number of tests per person with optimal configuration is presented for all of the procedures calculated when p is known. In the image, the grey dashed line at $t(N, p) = 1$ stands for the expected number of tests for individual testing. The coloured dashed lines correspond to the previously discussed procedure specific p break point. As we can see from this plot, with most of the p values (approximately $p \in (0.05, 0.38)$) Sterrett procedure provides best results (it's number of tests is smaller in this interval of p). For particularly small p values it seems that the generalised ML procedure is best, i.e. brown curve is the lowest in the interval of roughly $p < 0.05$. Also, it is worth mentioning the fact that if looked closely, curve of $t_H(N_{opt}, p)$ is not exactly monotonically increasing, the small error likely appears due to the assumption of $N = 2^n$ in article [6] - domain of a function $N \mapsto t(N, p)$ was decreased to the set of set $\{2, 4, 8, 16, ...\}$ before retrieving the closed analytical expression of optimal group size.

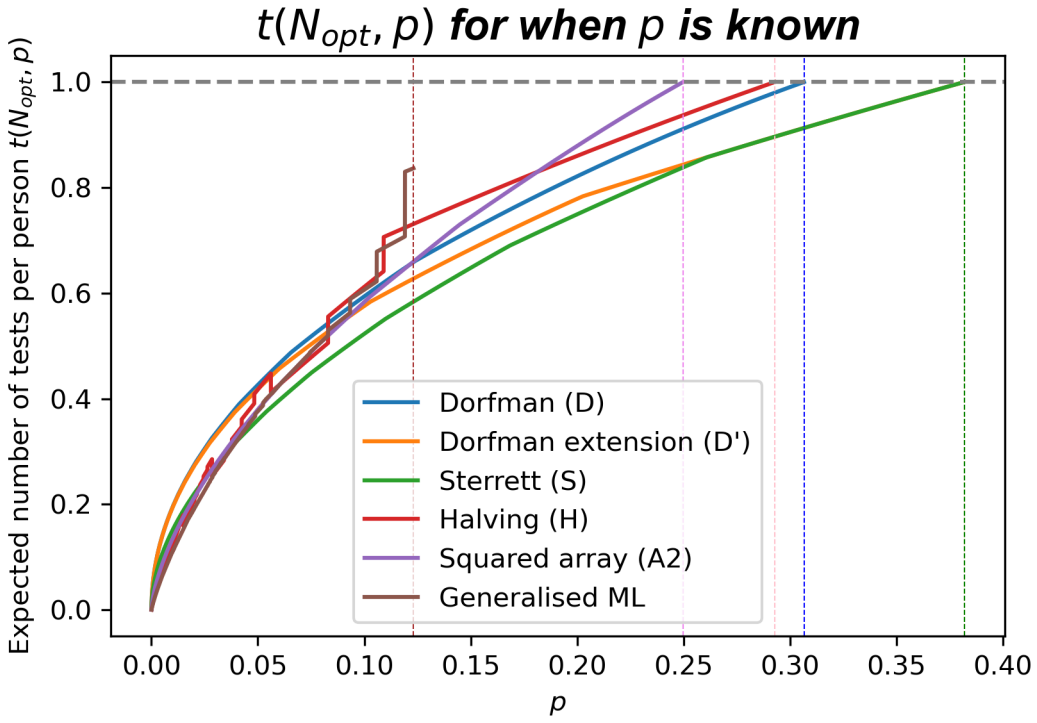

Figure 9: Expected number of tests per person when p is known for all procedures with gold standard tests

Secondly, after group sizes with which loss functions are at their minimum values are calculated (N_∞ and N_2) for each procedure, results are gathered in table 1. The highlighted numbers are the ones with which the area under the plot of expected number of tests is the smaller one. From the table it can be observed that the loss function L_2 minimised the number of tests a bit better for all the procedures. The optimal group size when p is unknown appears to be quite different between these few procedures, that can be explained with the fact that different procedures are defined in different size interval of p, i.e. the p break point is different. It is important to mention that the group size for squared array procedure is equal to $N = n^2$ and $N = s^d$ for generalized ML procedure.

Loss function	D	D'	S	H	A2	ML
L_∞	8	7	6	**5**	$12^2 = 144$	$14^2 = 196$
L_2	**4**	**3**	**4**	**5**	7^2 =**49**	11^2 =**121**

Table 1: N^*_{opt} for different procedures based on the two loss functions

Furthermore, as generalized ML procedure is a bit more tricky as it has 2 parameters instead of 1 (s and d compared to just N), plot of the loss functions can not be shown as with other procedures (fig. 7). Therefore we have visualised it in the format of a heatmap, see fig. 10. The darker the red colour - the smaller the value of loss function fig. 10. As it is not clear from these images, where the minimum points are we have emphasised it in fig. 11 by temporarily overwriting the minimum value with even smaller value so that it appears with bigger contrast from other points. To add,

the brighter upper part of all for heatmaps correspond to bigger value of d which is the power in the expression $N = s^d$.

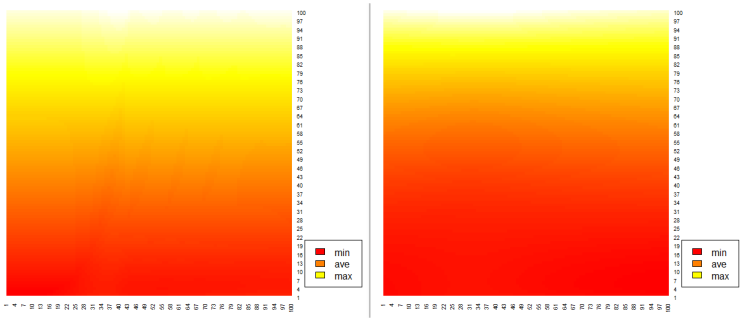

Figure 10: ML procedure: heatmaps of loss function values for each d, s combination (L_∞ on the left side and L_2 on the right, d on vertical axis and s horizontally)

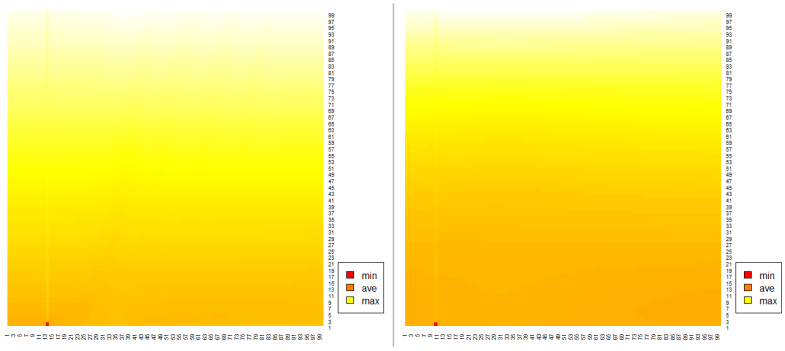

Figure 11: ML procedure: heatmaps of loss function values for each d, s combination with min dots emphasised

To sum up, we were able to find the optimal group size (minimising the expected number of tests per person) for investigated group testing procedures for gold standard tests, see the table 2. In it the upper bound of p (the mentioned break point) for each scheme is given in approximated values. From this table we can see that the optimal group size is significantly bigger for procedures with smaller interval of possible p values. Thus, before considering the options reader is advised to consider, if it is likely that the p value in tested population is below e.g. 0.12, and if not - not to use generalized ML procedure.

Procedure	D	D'	S	H	A2	ML
N^*_{opt}	4	3	4	5	49	121
$p_X \approx$	0.30664	0.38197	0.38197	0.29289	0.24979	0.12312

Table 2: Final N^*_{opt} when p is unknown; p_X is the cut point for values of p above which individual testing is preferable to the corresponding GT procedure

4.2.3 In the presence of test error - Hierarchical (D2)

Next, we take into consideration the test error. The sensitivity and/or specificity of a particular test is often below 1. We started with the calculation of $t_{D2}(N_{opt}(p), p)$ when p is known as with procedures above. Due to the fact that the visualisation of all combinations of S_e and S_p with 0.01 step size would be hardly interpretable, We have only taken the combinations of $S_e, S_p \in \{0.9, 0.95, 1\}$, see fig. 12. Main combinations are emphasised - $S_e = S_p = 1$ in black (corresponding to Dorfman procedure with gold standard test), $S_e = 1, S_p = 0.9$ in red and $S_e = 0.9, S_p = 1$ in blue. As we can see, the blue curve is the lowest and red is highest in the whole interval of p values. This means that with lower sensitivity the expected number of tests is a bit smaller and with lower specificity - a bit bigger.

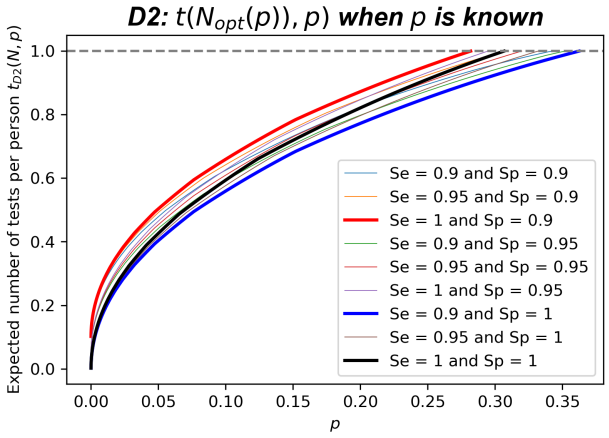

Figure 12: Hierarchical - $t(t(Nopt, p))$ for each combination of S_e, S_p when p is known

We have also calculated the p_{D2} (break point at which $t(N_{opt}, p)$ reaches 1), see table/heatmap in fig. 13. The background of the table is coloured based on the value of p_{D2} - the higher the value, the darker the green shade. This colouring helps see that p_{D2} is smallest at combination $S_e = 1$ and $S_p = 0.9$ and highest at $S_e = 0.9$ and $S_p = 1$. If we check the diagonal from left upper corner to bottom right corner, we can see that values decrease slowly with sensitivity and specificity both increasing at the same rate. The particular p_{D2} values are used as upper bound of p in the calculation of loss functions for each combination of S_e and S_p.

p_{D2}	S_p (specificity)										
S_e (sensitivity)	0.9	0.91	0.92	0.93	0.94	0.95	0.96	0.97	0.98	0.99	1
0.9	0.342	0.344	0.346	0.348	0.350	0.352	0.354	0.356	0.358	0.360	0.362
0.91	0.333	0.335	0.337	0.339	0.341	0.344	0.346	0.349	0.351	0.353	0.356
0.92	0.325	0.327	0.329	0.332	0.335	0.337	0.340	0.342	0.345	0.347	0.349
0.93	0.318	0.321	0.323	0.326	0.329	0.331	0.334	0.336	0.339	0.341	0.343
0.94	0.312	0.315	0.318	0.320	0.323	0.325	0.328	0.330	0.333	0.335	0.337
0.95	0.307	0.309	0.312	0.315	0.317	0.320	0.322	0.325	0.327	0.330	0.332
0.96	0.301	0.304	0.307	0.309	0.312	0.314	0.317	0.319	0.322	0.324	0.326
0.97	0.296	0.299	0.301	0.304	0.307	0.309	0.312	0.314	0.317	0.319	0.321
0.98	0.291	0.294	0.297	0.299	0.302	0.304	0.307	0.309	0.311	0.314	0.316
0.99	0.286	0.289	0.292	0.294	0.297	0.299	0.302	0.304	0.307	0.309	0.311
1	0.282	0.284	0.287	0.290	0.292	0.295	0.297	0.300	0.302	0.304	0.307

Figure 13: Hierarchical - p cut off point for each combination of S_e, S_p

The optimal group size for each combination of S_e and S_p can be seen in table 3. The main thing that one would notice is that the optimal group size is equal to either 4 or 5 for all combinations and the bigger value of N^*_{opt} when p is unknown is related with smaller S_e and/or S_p.

S_e \ S_p	0.9	0.91	0.92	0.93	0.94	0.95	0.96	0.97	0.98	0.99	1
0.9	5	5	5	5	5	5	5	5	5	5	5
0.91	5	5	5	5	5	5	5	5	5	5	5
0.92	5	5	5	5	5	5	5	5	5	5	4
0.93	5	5	5	5	5	5	5	5	5	4	4
0.94	5	5	5	5	5	5	5	5	5	4	4
0.95	5	5	5	5	5	5	5	5	4	4	4
0.96	5	5	5	5	5	5	5	4	4	4	4
0.97	5	5	5	5	5	5	4	4	4	4	4
0.98	5	5	5	5	5	4	4	4	4	4	4
0.99	5	5	5	5	4	4	4	4	4	4	4
1	5	5	5	5	4	4	4	4	4	4	4

Table 3: Hierarchical (D2) N^*_{opt} for each combination of S_e, S_p when p is not known

4.2.4 In the presence of test error - Squared array (A2)

Finally, for the last procedure investigated, p_{A2} (break point at which $t(N_{opt}, p)$ reaches 1 or if for this procedure mostly from which the minimum for N_{opt} is not defined anymore) is provided in the format of table/heatmap in fig. 14. The background of the table is coloured based on the value of p_{A2} - in the same way as in the section 4.2.3. Here we can see that bigger p_{A2} values seem to be related with sensitivity that are is near 1 and high specificity.

p_{A2}	S_p (specificity)										
S_e (sensitivity)	0.9	0.91	0.92	0.93	0.94	0.95	0.96	0.97	0.98	0.99	1
0.9	0.281	0.284	0.286	0.289	0.292	0.295	0.297	0.300	0.302	0.304	0.307
0.91	0.287	0.290	0.293	0.296	0.298	0.301	0.303	0.305	0.308	0.271	0.282
0.92	0.294	0.296	0.299	0.301	0.304	0.270	0.279	0.286	0.291	0.296	0.300
0.93	0.299	0.266	0.275	0.282	0.287	0.292	0.297	0.301	0.305	0.309	0.312
0.94	0.282	0.287	0.292	0.297	0.301	0.305	0.308	0.312	0.315	0.318	0.322
0.95	0.296	0.300	0.304	0.307	0.311	0.314	0.318	0.321	0.324	0.327	0.330
0.96	0.306	0.310	0.313	0.316	0.320	0.323	0.326	0.329	0.331	0.334	0.337
0.97	0.315	0.318	0.321	0.324	0.327	0.330	0.333	0.336	0.338	0.341	0.343
0.98	0.323	0.326	0.329	0.331	0.334	0.337	0.339	0.342	0.344	0.347	0.349
0.99	0.330	0.332	0.335	0.338	0.340	0.343	0.345	0.348	0.350	0.352	0.355
1	0.336	0.339	0.341	0.344	0.346	0.349	0.351	0.353	0.356	0.358	0.360

Figure 14: Squared array - p cut off point for each combination of S_e, S_p

The particular p_{A2} values from table above for each combination of S_e and S_p are used as upper bound of p in the calculation of loss functions, eventually providing the optimal group size when p is unknown for each combination of S_e and S_p, see table 4. It is clear that the tendency is different compared to the Hierarchical procedure (D2) in the previous section. The optimal group size varies from $N^*_{opt} = 12^2 = 144$ (with $S_e = 1$, $S_p = 0.9$) to $N^*_{opt} = 21^2 = 441$ (with $S_e = 0.9$, $S_p = 1$) and the biggest value of N^*_{opt} is for when p is unknown is related with smaller S_e and/or S_p.

It is important to remember that group size is $N = n^2$ in the definition of Squared array algorithm without master pool, also, the the procedure in the presence of test error is a bit different for the one with perfect tests - not only by the formula including the parameters of sensitivity and specificity, but also when a group tests positive then in the presence of test error the samples are tested individually if it is in intersection of positive column and row (as with perfect tests) or if sample is only in positive row or only in positive column. Therefore, if "gold standard" test is used, it is advised to use the information and results from section 4.2.2 rather than the ones from the right bottom corner of described tables.

S_e \ S_p	0.9	0.91	0.92	0.93	0.94	0.95	0.96	0.97	0.98	0.99	1
0.9	196	225	225	256	256	289	289	324	361	400	441
0.91	196	196	225	225	256	256	289	324	361	400	441
0.92	196	196	225	225	225	256	289	324	324	361	441
0.93	196	196	196	225	225	256	256	289	324	361	400
0.94	169	196	196	225	225	256	256	289	324	361	400
0.95	169	196	196	196	225	225	256	256	289	324	361
0.96	169	169	196	196	196	225	256	256	289	324	361
0.97	169	169	169	196	196	225	225	256	289	289	324
0.98	169	169	169	196	196	196	225	256	256	289	324
0.99	144	169	169	169	196	196	225	225	256	289	289
1	144	144	169	169	196	196	196	225	256	256	289

Table 4: Squared array (A2) N^*_{opt} for each combination of S_e, S_p when p is not known

4.2.5 Additional comments

There are a few things to mention before conclusions, for instance, some additional procedures with more than one parameter for group size were tried out and it appeared that the available resources for that were too small. We have gathered two ideas though on how to improve that a little bit. In particular, finding the optimal size for the Three-Dimensional Planar Method (3P) and the Three-Dimensional Linear Method (3L) from [1] - instead of looping through all combinations of 3 parameters for group size (e.g. from 1 to 10^4), decrease the number of them with only checking the combinations up to the diagonal as the diagonal is the "squared" case and everything below it is repeating already checked combinations as the combination of the same values but different order is the same case. Also, adding the limit on multiplication of all 3 parameters, e.g. $a \cdot b \cdot c < 10^6$ if a, b, c were theose 3 parameters.

From the described and investigated GT procedures, computations of algorithms with the presence of test error are more complex, i.e. a lot more simulations have to be checked through. Therefore we have been using the supercomputer of the university to make the computations for the latter.

5 Conclusions

To recap, the main goal was to provide a guideline to optimal group size when the prevalence is unknown. We were able to obtain these results - a few tables with optimal group sizes with unknown prevalence for main schemes from literature (tables 2 and 3).

The reader should be able to find the needed information, i.e. optimal group size for his situation, by considering if the test being used is "golden-standard" or has a known sensitivity and specificity, if exact prevalence is not known. In some cases there could be a few schemes to chose from, further choice can be made with the available resources in mind. On the other hand it might become clear that it doesn't make sense to use group testing for the situation at all. Consequently, using the provided guideline can help save some time and money.

From the gathered results of main probabilistic schemes, it appears that the optimal group size (when p is unknown) varies mostly between the schemes that have different upper bound of p ($p \in (0, p_X)$, where X is particular procedure). Namely, if it is smaller, then the optimal group size N^*_{opt} is significantly bigger and vice versa (see table 2). For instance, extension of generalised ML procedure has $p_{ML} \approx 0.12$ and $N^*_{opt} = 121$ while extension of Dorfman's scheme has $p_{D'} \approx 0.38$ and $N^*_{opt} = 3$. That can be explained by the two facts: the assumption of uniformly distributed p in the interval $p \in (0, p_X)$ and that $N_{opt}(p) \to \infty$ as $p \downarrow 0$ as described in the literature (when p is known). We were also able to notice that for Hierarchical (D2) procedure the values of sensitivity and specificity, varying between 0.9 and 1 do not affect much the value of optimal group size (see table 3, where p_{D2} varies from 0.282 to 0.362 and N^*_{opt} is equal to either 4 or 5). Lastly, behaviour of N^*_{opt} of Squared array algorithm (also in the presence of test error) is different than Hierarchical, likely partially due to differences of upper bound of p, i.e. p_{D2} and p_{A2}.

In addition to the main results, there are quite a few findings gathered through this work that are worth mentioning as well. In particular, the two used loss functions often yield different optimal group sizes, of course, the better one can be chosen by calculating the expected number of tests per person using integral (calculating the area under the curve). Moreover, after trying the available options of `argmin` function in R, to find with which group size (when p is known and fixed) the expected number of tests is at its minimum value, proved to be not the most efficient to use as it calculates the function for all values, sort them and then return the first value from ordered list. Therefore, instead creating a bit more simple function (that doesn't need the ordering of all values) and applying the knowledge of monotonically decreasing function property shortened the computational time significantly. However, even with this technique for schemes that use not one, but multiple (e.g. 3) different group sizes, calculation time grows exponentially. As a result, a few schemes were left out for future work, for instance - three-dimensional array based methods. A couple ideas to shorten the looping through all possible group sizes are mentioned above (see section 4.2.5). With this in mind, recommendations would include setting aside bigger resources for calculations and using mentioned ideas to also provide results for three-dimensional (or even higher dimension) methods. Further research could also include extended goal - based on current work, build a generalized script or even user-friendly platform, to find out optimal group size for

any scheme, maybe for some that don't even exist in the literature yet. One could also extend the guidance to combinatorial schemes as well. Finally, besides the fact that variance and not only average expected number of tests could be considered, for testing in the presence of misclassification, the error rate metrics could be added for guideline user to be aware of with specific optimal group size.

References

[1] Toby Berger, James W. Mandell, and P. Subrahmanya. Maximally Efficient Two-Stage Screening. *Biometrics*, 56(3):833–840, September 2000.

[2] Hae-Young Kim, Michael G. Hudgens, Jonathan M. Dreyfuss, Daniel J. Westreich, and Christopher D. Pilcher. Comparison of Group Testing Algorithms for Case Identification in the Presence of Test Error. *Biometrics*, 63(4):1152–1163, December 2007.

[3] A. Liu, C. Liu, Z. Zhang, and P. S. Albert. Optimality of group testing in the presence of misclassification. *Biometrika*, 99(1):245–251, March 2012.

[4] Yaakov Malinovsky and Paul S. Albert. A Note on the Minimax Solution for the Two-Stage Group Testing Problem. *The American Statistician*, 69(1):45–52, January 2015.

[5] Yaakov Malinovsky and Paul S. Albert. Revisiting Nested Group Testing Procedures: New Results, Comparisons, and Robustness. *The American Statistician*, 73(2):117–125, April 2019.

[6] Viktor Skorniakov, Remigijus Leipus, Gediminas Juzeliūnas, and Kęstutis Staliūnas. Group testing: Revisiting the ideas. *Nonlinear Analysis: Modelling and Control*, 26(3):534–549, May 2021.

[7] Ugnė Čižikovienė and Viktor Skorniakov. On the Optimal Configuration of a Square Array Group Testing Algorithm. *arXiv:2106.15603 [math, stat]*, June 2021. arXiv: 2106.15603.

Appendices

All appendices below list R program codes for particular procedures - Dorfman's procedure (D), Extension of Dorfman's procedure (D'), Sterrett's procedure (S), Halving procedure (H), Squared array (A2), Generalisation of 3L extension to multiple dimensions (ML), Hierarchical algorithm (D2) and Squared array (A2) in the presence of test error. The codes for each of the procedures are quite similar, main differences are of course the formula of expected number of tests per person and extra calculations if test is not perfect. Plotting of all the figures in the work is achieved using Python and its program code is added as the last appendix.

Appendix D

To begin with, the below R program code is for calculating the optimal group size when p is unknown for Dorfman's procedure.

```
################################
# Dorfman procedure
################################

######
# step 1: let's take P as a sequence of same size step in interval (0 , (3-sqrt(5))/2)
# e.g. with step of 10^(-6)

step <- 10^(-6)
start <- step
p_D <- 1-(1/3)^(1/3)

P <- seq(from = start, to = p_D, by=step)
m <- length(P)

######
# step 2: Let's now take a big N, it will probably be enough to take 100.
# let's mark A = {1,2,...,N}
N = 100
A <- 1:N

######
# step 3: for each N from A let's calculate L_inf and L_2

### expected number of tests per person
# t_D(N,p)
t_D <- function(p,n) {
  if (n>1) {1-(1-p)^n+1/n}
  else if (n==1) {1}
}
```

```
# matr_t_n[P,A]
matr_t_n <- matrix(NA, nrow = length(P), ncol = length(A))
for(i in 1:N) {
  matr_t_n[,i] <- sapply(P, t_D, n = i)
}

# t_D(N_opt,p)
func_n_opt1 <- function(p) {
  if (p<p_D) {(1+floor(p^(-1/2)))}
  else {1}
}
func_n_opt2 <- function(p) {
  if (p<p_D) {(2+floor(p^(-1/2)))}
  else {1}
}
N_opt1 <- sapply(P, func_n_opt1)
N_opt2 <- sapply(P, func_n_opt2)
func_t_n_opt <- function(p)(min(t_D(p, func_n_opt1(p)), t_D(p, func_n_opt2(p))))
t_n_opt <- sapply(P, func_t_n_opt)

# Err(N,p) = t_N(p) - t_n_opt(p)
matr_Err <- matrix(NA, nrow = length(P), ncol = length(A))
L_inf <- vector()
L_2 <- vector()
for(i in 1:N) {
  matr_Err[,i] <- matr_t_n[,i] - t_n_opt
  L_inf[i] <- max(matr_Err[,i])
  L_2[i] <- 1/m * sum(matr_Err[,i]^2)
}

# plot(A, L_inf, main="Procedure D")
# plot(A, L_2, main="Procedure D")

######
# step 4: we then take N_inf=argmin(L_inf) and N_2=argmin(L_2)
N_inf <- which.min(L_inf)
N_2 <- which.min(L_2)
N_inf
N_2

# Procedure D:
# > N_inf
# [1] 8
```

```
# > N_2
# [1] 4

#####
# step 5: area under the curve

area_inf <- 0
area_2 <- 0
for (i in 1:m) {
  area_inf = area_inf + matr_t_n[i,N_inf] * step
  area_2 = area_2 + matr_t_n[i,N_2] * step
}
area_inf
area_2

# > area_inf
# [1] 0.2379724
# > area_2
# [1] 0.215348

if (area_inf<area_2) {
  N_opt_final <- N_inf
} else {
  N_opt_final <- N_2
}
N_opt_final
# [1] 4

#####
# step 6: Saving the results for plotting in Python:

# t_n_opt:
df_t_d_opt <- data.frame(P,t_n_opt)
write.csv(df_t_d_opt, file = "t_D_opt.csv", row.names = FALSE)

# loss functions:

# to data frame
df <- data.frame(A,L_inf,L_2)
write.csv(df, file = "D_loss.csv", row.names = FALSE)

# t_N_inf, t_N_2:

# to data frame
```

```
t_inf <- matr_t_n[,N_inf]
t_2 <- matr_t_n[,N_2]
df <- data.frame(P,t_inf,t_2)

write.csv(df, file = "D_t_loss.csv", row.names = FALSE)
```

Appendix D'

Secondly, the R program code beneath calculates the optimal group size when p is unknown for the extension of Dorfman's procedure.

```
################################
# Dorfman extension procedure
################################

######
# step 1: let's take P as a sequence of same size step in interval (0 , (3-sqrt(5))/2)
# e.g. with step of 10^(-6)

step <- 10^(-6)
start <- step
end <- (3-sqrt(5))/2
P <- seq(from = start, to = end, by=step)
m <- length(P)

######
# step 2: Let's now take a big N, it will probably be enough to take 100.
# let's mark A = {1,2,...,N}
N = 100
A <- 1:N

######
# step 3: for each N from A let's calculate L_inf and L_2

### expected number of tests per person
# t_D'(N,p)
t_D_ext <- function(p, n)(1-(1-p)^n+1/n-(1/n)*p*(1-p)^(n-1))

# matr_t_n[P,A]
matr_t_n <- matrix(NA, nrow = length(P), ncol = length(A))
for(i in 1:N) {
  matr_t_n[,i] <- sapply(P, t_D_ext, n = i)
}

# t_D'(N_opt,p)
```

```
func_n_opt1 <- function(p)(floor(p^(-1/2)))
func_n_opt2 <- function(p)(ceiling(p^(-1/2)))
func_t_n_opt <- function(p)(min(t_D_ext(p, func_n_opt1(p)), t_D_ext(p, func_n_opt2(p))))
t_n_opt <- sapply(P, func_t_n_opt)


# Err(N,p) = t_N(p) - t_n_opt(p)
matr_Err <- matrix(NA, nrow = length(P), ncol = length(A))
L_inf <- vector()
L_2 <- vector()
for(i in 1:N) {
  matr_Err[,i] <- matr_t_n[,i] - t_n_opt
  L_inf[i] <- max(matr_Err[,i])
  L_2[i] <- 1/m * sum(matr_Err[,i]^2)
}

# plot(A, L_inf, main="Procedure D'")
# plot(A, L_2, main="Procedure D'")

######
# step 4: we then take N_inf=argmin(L_inf) and N_2=argmin(L_2)
N_inf <- which.min(L_inf)
N_2 <- which.min(L_2)
N_inf
N_2

# Procedure D':
# > N_inf
# [1] 7
# > N_2
# [1] 3

#####
# step 5: area under the curve

area_inf <- 0
area_2 <- 0
for (i in 1:m) {
  area_inf = area_inf + matr_t_n[i,N_inf] * step
  area_2 = area_2 + matr_t_n[i,N_2] * step
}
area_inf
area_2
```

```
# > area_inf
# [1] 0.3119656
# > area_2
# [1] 0.2820567

if (area_inf<area_2) {
  N_opt_final <- N_inf
} else {
  N_opt_final <- N_2
}
N_opt_final
# [1] 3

#####
# step 6: Saving the results for plotting in Python:

# t_n_opt:
df_t_D_ext_opt <- data.frame(P,t_n_opt)
write.csv(df_t_D_ext_opt, file = "t_D_ext_opt.csv", row.names = FALSE)

# loss functions:
# to data frame
df <- data.frame(A,L_inf,L_2)
write.csv(df, file = "D_ext_loss.csv", row.names = FALSE)

# t_N_inf, t_N_2:
# to data frame
t_inf <- matr_t_n[,N_inf]
t_2 <- matr_t_n[,N_2]
df <- data.frame(P,t_inf,t_2)
write.csv(df, file = "D_ext_t_loss.csv", row.names = FALSE)
```

Appendix S

Next R program code is dedicated for Sterrett's procedure.

```
################################
# Sterrett procedure
################################

######
# step 1: let's take P as a sequence of same size step in interval (0 , (3-sqrt(5))/2)
# e.g. with step of 10^(-6)
```

```
step <- 10^(-6)
start <- step
end <- (3-sqrt(5))/2
P <- seq(from = start, to = end, by=step)
m <- length(P)

######
# step 2: Let's now take a big N, it will probably be enough to take 100.
# let's mark A = {1,2,...,N}
N = 100
A <- 1:N

######
# step 3: for each N from A let's calculate L_inf and L_2

### expected number of tests per person
# t_S(N,p)
t_S <- function(p, n)( (1/n)*( 2*n - (n-2)*(1-p) - (1-(1-p)^(n+1))/p ) )

# matr_t_n[P,A]
matr_t_n <- matrix(NA, nrow = length(P), ncol = length(A))
for(i in 1:N) {
  matr_t_n[,i] <- sapply(P, t_S, n = i)
}

# t_S(N_opt,p)
func_n_opt1 <- function(p)(floor(sqrt(2*p^(-1))))
func_n_opt2 <- function(p)(func_n_opt1(p)+1)
func_n_opt3 <- function(p)(func_n_opt1(p)+2)
func_t_n_opt <- function(p)(min(
  t_S(p, func_n_opt1(p)),
  t_S(p, func_n_opt2(p)),
  t_S(p, func_n_opt3(p))))
t_n_opt <- sapply(P, func_t_n_opt)

# Err(N,p) = t_N(p) - t_n_opt(p)
matr_Err <- matrix(NA, nrow = length(P), ncol = length(A))
L_inf <- vector()
L_2 <- vector()
for(i in 1:N) {
  matr_Err[,i] <- matr_t_n[,i] - t_n_opt
  L_inf[i] <- max(matr_Err[,i])
  L_2[i] <- 1/m * sum(matr_Err[,i]^2)
}
```

```
# plot(A, L_inf, main="Procedure S")
# plot(A, L_2, main="Procedure S")

######
# step 4: we then take N_inf=argmin(L_inf) and N_2=argmin(L_2)
N_inf <- which.min(L_inf)
N_2 <- which.min(L_2)
N_inf
N_2

# Procedure S:
# > N_inf
# [1] 6
# > N_2
# [1] 4

#####
# step 5: area under the curve

area_inf <- 0
area_2 <- 0
for (i in 1:m) {
  area_inf = area_inf + matr_t_n[i,N_inf] * step
  area_2 = area_2 + matr_t_n[i,N_2] * step
}
area_inf
area_2

# > area_inf
# [1] 0.2851822
# > area_2
# [1] 0.2741443

if (area_inf<area_2) {
  N_opt_final <- N_inf
} else {
  N_opt_final <- N_2
}
N_opt_final
# [1] 4

#####
# step 6: Saving the results for plotting in Python:
```

```
# t_n_opt:
df_t_s_opt <- data.frame(P,t_n_opt)
write.csv(df_t_s_opt, file = "t_S_opt.csv", row.names = FALSE)

# loss functions:
# to data frame
df <- data.frame(A,L_inf,L_2)
write.csv(df, file = "S_loss.csv", row.names = FALSE)

# t_N_inf, t_N_2:
# to data frame
t_inf <- matr_t_n[,N_inf]
t_2 <- matr_t_n[,N_2]
df <- data.frame(P,t_inf,t_2)
write.csv(df, file = "S_t_loss.csv", row.names = FALSE)
```

Appendix H

Further, R program code here is for calculating the optimal group size N^*_{opt} when p is unknown for Halving scheme.

```
################################
# Halving procedure
################################

######
# step 1: let's take P as a sequence of same size step in interval (0 , (3-sqrt(5))/2)
# e.g. with step of 10^(-6)

step <- 10^(-6)
start <- step
end <- (3-sqrt(5))/2
# P <- seq(from = start, to = end, by=step)
# p_H is calculated below (first taking bigger interval to find the value)
p_H <- 0.292893
P <- seq(from = start, to = p_H, by=step)
m <- length(P)

######
# step 2: Let's now take a big N, it will probably be enough to take 100.
# let's mark A = {1,2,...,N}
N = 100
A <- 1:N
```

```
######
# step 3: for each N from A let's calculate L_inf and L_2

### expected number of tests per person
# t_H(N,p)

t_H <- function(p, n){
  sum <- 0
  # print(floor(log2(n)))
  for (i in 1:round(log2(n))) {
    sum <- sum + (1-(1-p)^(2^i))/(2^i)
  }
  result <- 1/n + 2 * sum
  return (result)
}

# matr_t_n[P,A]
matr_t_n <- matrix(NA, nrow = length(P), ncol = length(A))
for(i in 1:N) {
  matr_t_n[,i] <- sapply(P, t_H, n = i)
}

# t_D(N_opt,p)
func_n_opt1 <- function(p) { max(floor(1/(2*log2(1/(1-p)))),1) }
func_n_opt2 <- function(p) { floor(1/(2*log2(1/(1-p)))) + 1 }

func_t_n_opt <- function(p)(min(t_H(p, func_n_opt1(p)), t_H(p, func_n_opt2(p))))
t_n_opt <- sapply(P, func_t_n_opt)

# finding prevalence upper bound p_H
# for(i in 2:m) {
#   if (t_n_opt[i-1] < 1 & t_n_opt[i] >= 1) {
#     p_i <- i-1
#   }
# }
# P[p_i]
# [1] 0.292893

# Err(N,p) = t_N(p) - t_N_opt(p)
matr_Err <- matrix(NA, nrow = length(P), ncol = length(A))
L_inf <- vector()
L_2 <- vector()
for(i in 1:N) {
  matr_Err[,i] <- matr_t_n[,i] - t_n_opt
```

```
  L_inf[i] <- max(matr_Err[,i])
  L_2[i] <- 1/m * sum(matr_Err[,i]^2)
}


######
# step 4: we then take N_inf=argmin(L_inf) and N_2=argmin(L_2)
N_inf <- which.min(L_inf)
N_2 <- which.min(L_2)
N_inf
N_2

# Procedure H:
# > N_inf
# [1] 5
# > N_2
# [1] 5

#####
# step 5: area under the curve

# No need to calculate area under the curve to see
# which loss function gave better result as both got the same - 5

area_inf <- 0
area_2 <- 0
for (i in 1:m) {
  area_inf = area_inf + matr_t_n[i,N_inf] * step
  area_2 = area_2 + matr_t_n[i,N_2] * step
}
area_inf
area_2
# 0.2001141

if (area_inf<area_2) {
  N_opt_final <- N_inf
} else {
  N_opt_final <- N_2
}
N_opt_final
# [1] 5

# plot(matr_t_n[,N_opt_final])
# plot(t_n_opt)
```

```
#####
# step 6: Saving the results for plotting in Python:

# t_n_opt:
df_t_d_opt <- data.frame(P,t_n_opt)
write.csv(df_t_d_opt, file = "t_H_opt.csv", row.names = FALSE)

# loss functions:

# to data frame
df <- data.frame(A,L_inf,L_2)
write.csv(df, file = "H_loss.csv", row.names = FALSE)

# t_N_inf: (t_N_2 will be the same)

# to data frame
t_inf <- matr_t_n[,N_inf]
df <- data.frame(P,t_inf)

write.csv(df, file = "H_t_loss.csv", row.names = FALSE)
```

Appendix A2

Similarly, the R program code below is for Squared array procedure (with perfect tests).

```
################################
# Squared array procedure
# (without master pool)
# for golden standard tests
# Se=Sp=1
################################

######
# step 1: let's take P as a sequence of same size step in interval (0 , (3-sqrt(5))/2)
# e.g. with step of 10^(-6)

step <- 10^(-6)
start <- step
end <- (3-sqrt(5))/2
p_A2 <- 1-0.750209961
P <- seq(from = start, to = p_A2, by=step)
m <- length(P)

######
# step 2: Let's now take a big N, it will probably be enough to take 100.
```

```
# let's mark A = {1,2,...,N}
N = 100
A <- 1:N

######
# step 3: for each N from A let's calculate L_inf and L_2

### expected number of tests per person
# t_A2(N,p)
t_A2 <- function(p,n)(2/n+1-2*(1-p)^n+(1-p)^(2*n-1))

# matr_t_n[P,A]
matr_t_n <- matrix(NA, nrow = length(P), ncol = length(A))
for(i in 1:N) {
  matr_t_n[,i] <- sapply(P, t_A2, n = i)
}

# t_A2(N_opt,p)
func_n_opt1 <- function(p)( floor(1/(p^(2/3)) + 1/(2*p^(1/3)) + 3*p^2 + 0.2) )
func_n_opt2 <- function(p)(func_n_opt1(p) + 1)
func_n_opt3 <- function(p)(func_n_opt1(p) + 2)
func_t_n_opt <- function(p)(min(
  t_A2(p, func_n_opt1(p)),
  t_A2(p, func_n_opt2(p)),
  t_A2(p, func_n_opt3(p))))
t_n_opt <- sapply(P, func_t_n_opt)

# Err(N,p) = t_N(p) - t_n_opt(p)
matr_Err <- matrix(NA, nrow = length(P), ncol = length(A))
L_inf <- vector()
L_2 <- vector()
for(i in 1:N) {
  matr_Err[,i] <- matr_t_n[,i] - t_n_opt
  L_inf[i] <- max(matr_Err[,i])
  L_2[i] <- 1/m * sum(matr_Err[,i]^2)
}

# plot(A, L_inf, main="Procedure A2")
# plot(A, L_2, main="Procedure A2")

######
# step 4: we then take N_inf=argmin(L_inf) and N_2=argmin(L_2)
N_inf <- which.min(L_inf)
N_2 <- which.min(L_2)
```

```
N_inf
N_2

# Procedure A2:
# > N_inf
# [1] 12
# > N_2
# [1] 7

#####
# step 5: area under the curve

area_inf <- 0
area_2 <- 0
for (i in 1:m) {
  area_inf = area_inf + matr_t_n[i,N_inf] * step
  area_2 = area_2 + matr_t_n[i,N_2] * step
}
area_inf
area_2

# > area_inf
# [1] 0.1828688
# > area_2
# [1] 0.1663942

if (area_inf<area_2) {
  N_opt_final <- N_inf
} else {
  N_opt_final <- N_2
}
# N = n^2
N_opt_final^2

# > N_opt_final^2
# [1] 49

#####
# step 6: Saving the results for plotting in Python:

# t_n_opt:
df_t_A2_opt <- data.frame(P,t_n_opt)
write.csv(df_t_A2_opt, file = "t_A2_opt.csv", row.names = FALSE)
```

```
# loss functions:

# to data frame
df <- data.frame(A,L_inf,L_2)
write.csv(df, file = "A2_loss.csv", row.names = FALSE)

# t_N_inf, t_N_2:

# to data frame
t_inf <- matr_t_n[,N_inf]
t_2 <- matr_t_n[,N_2]
df <- data.frame(P,t_inf,t_2)
write.csv(df, file = "A2_t_loss.csv", row.names = FALSE)
```

Appendix ML

In this appendix there are 2 R codes, first is for optimal configuration (find best pair of s and d) calculation when p is known:

```
# target function
tN <- function(s,d,q){
  d/s + (1-q) + q*(1-q**(s-1))**d
}
# scaled g function
g <- function(q,x){
  # 1/log(q)*(q*(1-x)*log(1-x)-x*log(x)) - unscaled
  if(x>0 & x<1)  1/log(q)*(q*(1-x)*log(1-x)-x*log(x))
  else 0
}
# scaled derivative of g
g1 <- function(q,x){
  # -1/log(q)*(q*log(1-x) + log(x) + 1 + q) - unscaled
  return(q*log(1-x) + log(x) + 1 + q)
}
# inflection point
iPoint <- function(q){
  1/(1 + q)
}
# starting point for search of critical points
startPoint <- function(q){
  a <- iPoint(q)
  b <- 1-exp(-3)
  sPoint <- (a+b)/2
  while(g(q,sPoint) <= sPoint){
    if(g1(q,a)*g1(q,sPoint)<0) b <- sPoint
```

```
    else a <- sPoint
    sPoint <- (a+b)/2
  }
  return(sPoint)
}
# bisection method for arbitrary function
getZero <- function(a0,b0,eps,maxIter,f){
  if(!(f(a0)*f(b0)<0)){
    print("[a,b] is not suitable for bisection method")
    return()
  }
  a <- a0
  b <- b0
  x0 <- (a+b)/2
  for(i in c(1:maxIter)){
    if(abs(f(x0))<eps) {break}
    if(f(x0)*f(a)<0) b <- x0
    else a <- x0
    x0 <- (a+b)/2
  }
  if(i==maxIter) print("Maximum number of iterations was reached!")
  return(x0)
}
# testing
f1<-function(x){(x-1)*(x+2)}
f2<-function(x){sin(x)}
getZero(a0=-0.01,b0=1.5,eps=0.00001, maxIter=100, f=f2)

# function for retrieval s and d given solution of auxiliary equation
get_s_d_pair <- function(x,q){
  s <- log(x)/log(q)+1
  d <- log(1/(s*q*(-log(1-q**(s-1)))))/log(1-q**(s-1))
  return(c(s=s,d=d))
}
getMax_Pair <-  function(q,eps,maxIter){
  f <- function(x){
    retVal <- x*log(q)
    if(x>0 & x<1) retVal <- retVal - (q*(1-x)*log(1-x)-x*log(x))
    return(retVal)
    }
  sPoint <- startPoint(q)
  leftZero <- getZero(a0=0.25,b0=sPoint,eps=eps, maxIter=1000, f=f)
  rightZero <- getZero(a0=sPoint,b0=1,eps=eps, maxIter=1000, f=f)
  leftPoints <- get_s_d_pair(x = leftZero,q = q)
```

```
  rightPoints <- get_s_d_pair(x = rightZero,q = q)
  if(tN(s = leftPoints["s"],d = leftPoints["d"],q=q)>
     tN(s = rightPoints["s"],d = rightPoints["d"],q=q)){
    return(c(rightPoints,loc="right"))
  }
  else {
    return(c(leftPoints,loc="left"))
  }
}
# testing: observe that difference in 0.0009 gives quite large deviation for s
# and that value of q=0.9909 matches the result of the paper
getMax_Pair(q=0.9909,eps = 0.00001,maxIter = 10000)
getMax_Pair(q=0.99,eps = 0.00001,maxIter = 10000)

result <- getMax_Pair(q=0.9909,eps = 0.00001,maxIter = 10000)
s <- as.numeric(result["s"])
d <- as.numeric(result["d"])
s1 <- floor(s)
s2 <- ceiling(s)
d1 <- floor(d)
d2 <- ceiling(d)
s1
s2
d1
d2
# > s1
# [1] 74
# > s2
# [1] 75
# > d1
# [1] 5
# > d2
# [1] 6
```

After the above code in R is compiled, then the R code beneath can be used for calculation of optimal configuration when p is unknown:

```
################################
# ML procedure
# using the function getMax_Pair
################################

######
# step 1: let's take P as a sequence of same size step in interval (0 , (3-sqrt(5))/2)
# e.g. with step of 10^(-6)
```

```
step <- 10^(-5)
# step <- 10^(-6) # Error: cannot allocate vector of size 9.2 Gb
start <- step
end <- (3-sqrt(5))/2
end_ML <- 1-0.876874
P <- seq(from = start, to = end_ML, by=step)
m <- length(P)

######
# step 2: Let's now take a big N, it will probably be enough to take 100.
# let's mark A = {1,2,...,N}
N = 100
A <- 1:N

######
# step 3: for each N from A let's calculate L_inf and L_2

t_ML <- function(p, d, s){d*s^(-1) + p + (1-p)*(1-(1-p)^(s-1))^d}
matr_t_n <- array(NA,dim=c(length(P),length(A),length(A)))
for (i in 1:N) {
  for (j in 1:N) {
    matr_t_n[,i,j] <- sapply(P, t_ML, d = i, s = j)
  }
}
# matr_t_n[,i,j] is a vector with value for each P
# matr_t_n[P,D,S]

### t_n_opt

func_d_opt1 <- function(p) {max(floor(
  as.numeric(getMax_Pair(q=1-p,eps = 0.00001,maxIter = 10000)["d"])
  ),2)} # d >=2
func_d_opt2 <- function(p) {max(ceiling(
  as.numeric(getMax_Pair(q=1-p,eps = 0.00001,maxIter = 10000)["d"])
  ),2)} # d >=2
func_s_opt1 <- function(p) {max(floor(
  as.numeric(getMax_Pair(q=1-p,eps = 0.00001,maxIter = 10000)["s"])
  ),2)} # s >=2
func_s_opt2 <- function(p) {max(ceiling(
  as.numeric(getMax_Pair(q=1-p,eps = 0.00001,maxIter = 10000)["s"])
  ),2)} # s >=2

func_t_n_opt <- function(p) {
```

```
  min(
    t_ML(p, func_d_opt1(p), func_s_opt1(p)),
    t_ML(p, func_d_opt1(p), func_s_opt2(p)),
    t_ML(p, func_d_opt2(p), func_s_opt1(p)),
    t_ML(p, func_d_opt2(p), func_s_opt2(p))
  )
}
t_n_opt <- sapply(P, func_t_n_opt)

################
# Loss function

# Err(N,p) = t_N(p) - t_n_opt(p)
matr_Err <- array(NA,dim=c(length(P),length(A),length(A)))
L_inf <- matrix(NA, nrow = length(A), ncol = length(A))
L_2 <- matrix(NA, nrow = length(A), ncol = length(A))
for (i in 2:N) { # d >=2
  for (j in 2:N) { # s >=2
    matr_Err[,i,j] <- matr_t_n[,i,j] - t_n_opt
    L_inf[i,j] <- max(matr_Err[,i,j])
    L_2[i,j] <- 1/m * sum(matr_Err[,i,j]^2)
  }
}

######
# step 4: we then take N_inf=argmin(L_inf) and N_2=argmin(L_2)
min(L_inf, na.rm = T)
min(L_2, na.rm = T)
# N_inf <- which.min(L_inf)
# N_2 <- which.min(L_2)
# NA's in firsts column and row - where s=1 or d=1 combinations would be,
# but they are not allowed in ML scheme by definition
N_inf <- which(L_inf == min(L_inf, na.rm = T), arr.ind = TRUE)
N_2 <- which(L_2 == min(L_2, na.rm = T), arr.ind = TRUE)
N_inf
N_2

View(L_inf)

# if only s,d from 2:
# > N_inf
# row col
# [1,]   2  14
# > N_2
```

```
# row col
# [1,]   2  11

#####
# step 5: area under the curve

area_inf <- 0
area_2 <- 0
for (i in 1:m) {
  area_inf = area_inf + matr_t_n[i,N_inf[1],N_inf[2]] * step
  area_2 = area_2 + matr_t_n[i,N_2[1],N_2[2]] * step
}
area_inf
area_2

# > area_inf
# [1] 0.06077097
# > area_2
# [1] 0.05621666

N_opt_final <- N_inf
if (area_2 <= area_inf) {
  N_opt_final <- N_2
}
d_opt <- N_opt_final[1]
s_opt <- N_opt_final[2]
d_opt
s_opt

# > d_opt
# [1] 2
# > s_opt
# [1] 11

######
# step 6: heatmaps of loss functions

min(L_inf, na.rm = T)
min(L_2, na.rm = T)

heatmap(L_inf, Colv = NA, Rowv = NA, scale="column",col=heat.colors(100))
legend(x="bottomright", legend=c("min", "ave", "max"), fill=heat.colors(3))

heatmap(L_2, Colv = NA, Rowv = NA, scale="column",col=heat.colors(100))
```

```
legend(x="bottomright", legend=c("min", "ave", "max"), fill=heat.colors(3))

# make more contrast to see the min value
L_inf_copy = L_inf
L_inf_copy[N_inf[1],N_inf[2]] = -10

heatmap(L_inf_copy, Colv = NA, Rowv = NA, scale="column",col=heat.colors(100))
legend(x="bottomright", legend=c("min", "ave", "max"), fill=heat.colors(3))

# make more contrast to see the min value
L_2_copy = L_2
L_2_copy[N_2[1],N_2[2]] = -100

heatmap(L_2_copy, Colv = NA, Rowv = NA, scale="column",col=heat.colors(100))
legend(x="bottomright", legend=c("min", "ave", "max"), fill=heat.colors(3))


#####
# step 7: save results for plots to csv (plot in Python)

# t_n_opt:
df_t_n_opt <- data.frame(P,t_n_opt)
write.csv(df_t_n_opt, file = "t_ML_opt.csv", row.names = FALSE)

# t_N_inf, t_N_2:
area_inf = area_inf + matr_t_n[i,N_inf[1],N_inf[2]] * step
area_2 = area_2 + matr_t_n[i,N_2[1],N_2[2]] * step
# to data frame
t_inf <- matr_t_n[,N_inf[1],N_inf[2]]
t_2 <- matr_t_n[,N_2[1],N_2[2]]
df <- data.frame(P,t_inf,t_2)
# save
write.csv(df, file = "ML_t_loss.csv", row.names = FALSE)
```

Appendix D2

Next, the R program code below calculates the optimal group size when p is unknown in the presence of test error for Hierarchical procedure (H2).

```
######################
# Hierarchical full
######################

# print the index of knot (0-10)
# x <- 0
```

```
x <- as.numeric(Sys.getenv("SLURM_PROCID"))
paste("x value is:", x)

# Final version of our argmin function:
my_argmin <- function(N_max, f){
  Nmin<-1
  if (N_max>=2) {
    for (N in 2:N_max) {
      if (f(N)<f(Nmin)) {
        Nmin<-N
      }
    }
  }
  return(Nmin)
}

######
# step 1: let's take P as a sequence of same size step in interval (0 , (3-sqrt(5))/2)
# e.g. with step of 10^(-6)

step <- 10^(-6) #-2
start <- step
end <- (3-sqrt(5))/2
P <- seq(from = start, to = end, by=step)
m <- length(P)

# Se and Sp
Se <- seq(from = 0.9, to = 1, by=0.01) # 0.01
Sp <- Se

# let's say there are 11 knots used, then we can fix Se for each x
# fixed Se for each knot
Se <- Se[x+1]

# paste("Se[x+1] value is:", Se)

######
# step 2: Let's now take a big N, it will probably be enough to take 100.
# let's mark A = {1,2,...,N}
N = 100
A <- 1:N

# print(paste("\"Se","Sp","Nopt","p_break\"", sep="\",\""), quote = FALSE)
```

```
filename = paste("res_", x,".txt",sep="")
file.create(filename)
fileConn<-file(filename)
writeLines(paste("\"Se","Sp","Nopt","p_break\"", sep="\",\""), fileConn)
close(fileConn)

# Hierarchical S=2
# expected number of tests per person
t_D2 <- function(p,n,S_e,S_p) { 1/n + (1-S_p)*(1-p)^n + S_e*(1-(1-p)^n) }

######
# step 2: we then take N_opt=arg.min(t_N)

N_opt <- array(NA,dim=c(length(P)))
t_n_opt <- array(NA,dim=c(length(P),length(Se),length(Sp)))
# index of p_break
p_i <- matrix(NA, nrow = length(Se), ncol = length(Sp))

for(sen in 1:length(Se)) {
  for(spe in 1:length(Sp)) {
    i <- 1
    N_opt[i] <- my_argmin(10^6, function(n)(t_D2(P[i],n,Se[sen],Sp[spe])))
    t_n_opt[i,sen,spe] <- t_D2(P[i],N_opt[i],Se[sen],Sp[spe])
    if (t_n_opt[i,sen,spe] < 1) {
      p_i[sen,spe] <- i
    }
    for(i in 2:m) {
      N_opt[i] <- my_argmin(N_opt[i-1]+1, function(n)(t_D2(P[i],n,Se[sen],Sp[spe])))
      t_n_opt[i,sen,spe] <- t_D2(P[i],N_opt[i],Se[sen],Sp[spe])
      if (t_n_opt[i-1,sen,spe] < 1 & t_n_opt[i,sen,spe] >= 1) {
        p_i[sen,spe] <- i-1
      }
    }
  }
}

paste("first part finished:", x)

######
# step 3: for each N from A let's calculate L_inf and L_2

# initialise different sizes matrices
matr_t_n <- array(NA,dim=c(length(P),length(A)))
matr_Err <- array(NA,dim=c(length(P)))
```

```
# L_inf <- array(NA,dim=c(length(A),length(Se),length(Sp)))
# L_2 <- array(NA,dim=c(length(A),length(Se),length(Sp)))
L_inf <- array(NA,dim=c(length(A)))
L_2 <- array(NA,dim=c(length(A)))
N_opt_final <- array(NA,dim=c(length(Se),length(Sp)))
# calculate N_opt for each Se,Sp combination
for(sen in 1:length(Se)) {
  for(spe in 1:length(Sp)) {
    # calculate loss functions
    for (i in 1:N) {
      matr_t_n[,i] <- sapply(P, t_D2, n = i, S_e = Se[sen], S_p = Sp[spe])
      matr_Err <- matr_t_n[,i] - t_n_opt[,sen,spe]
      L_inf[i] <- max(matr_Err[1:p_i[sen,spe]])
      L_2[i] <- 1/m * sum(matr_Err[1:p_i[sen,spe]]^2)
    }
    N_inf <- which.min(L_inf)
    N_2 <- which.min(L_2)
    # calculate area under curve for both loss functions
    area_inf <- 0
    area_2 <- 0
    for (p_ind in 1:p_i[sen,spe]) {
      area_inf <- area_inf + matr_t_n[p_ind,N_inf] * step
      area_2 <- area_2 + matr_t_n[p_ind,N_2] * step
    }
    # choose which loss N_opt is better
    if (area_inf<area_2) {
      N_opt_final[sen,spe] <- N_inf
    } else {
      N_opt_final[sen,spe] <- N_2
    }
    write(
      paste(Se[sen], Sp[spe], N_opt_final[sen,spe], P[p_i[sen,spe]], sep=","),
      file=filename,append=TRUE)
    # print(paste(Se[sen], Sp[spe], N_opt_final[sen,spe], P[p_i[sen,spe]], sep=","))
  }
}
# N_opt_final
paste("done:", x)


################################
# Hierarchical only nine curves
################################

# print the index of knot (0-10)
```

```
# x <- 0
x <- as.numeric(Sys.getenv("SLURM_PROCID"))
paste("x value is:", x)

# Final version of our argmin function:
my_argmin <- function(N_max, f){
  Nmin<-1
  if (N_max>=2) {
    for (N in 2:N_max) {
      if (f(N)<f(Nmin)) {
        Nmin<-N
      }
    }
  }
  return(Nmin)
}

######
# step 1: let's take P as a sequence of same size step in interval (0 , (3-sqrt(5))/2)
# e.g. with step of 10^(-6)

step <- 10^(-6) #-2
start <- step
end <- (3-sqrt(5))/2
P <- seq(from = start, to = end, by=step)
m <- length(P)

Se <- seq(from = 0.9, to = 1, by=0.05)
Sp <- Se

# let's say there are 9 knots used
# Se, Sp value for each knot
Se <- Se[x%%3+1]
Sp <- Sp[x%/%3+1]

paste(Se, Sp, sep=",")

filename = paste("res_t_", x,".txt",sep="")
file.create(filename)
fileConn<-file(filename)
writeLines(paste("\"p","Se","Sp","tNopt\"", sep="\",\""), fileConn)
close(fileConn)

# Hierarchical S=2
```

```
# expected number of tests per person
t_D2 <- function(p,n) { 1/n + (1-Sp)*(1-p)^n + Se*(1-(1-p)^n) }

######
# step 2: we then take N_opt=arg.min(t_N)

N_opt <- array(NA,dim=c(length(P)))
t_n_opt <- array(NA,dim=c(length(P),length(Se),length(Sp)))

# write(paste(P[i], Se, Sp, t_n_opt[i], sep=","),file=filename,append=TRUE)

i <- 1
N_opt[i] <- my_argmin(10^6, function(n)(t_D2(P[i],n)))
t_n_opt[i] <- t_D2(P[i],N_opt[i])
if (t_n_opt[i] < 1) {
  p_i <- i
}
write(paste(P[i], Se, Sp, t_n_opt[i], sep=","),file=filename,append=TRUE)
for(i in 2:m) {
  N_opt[i] <- my_argmin(N_opt[i-1]+1, function(n)(t_D2(P[i],n)))
  t_n_opt[i] <- t_D2(P[i],N_opt[i])
  write(paste(P[i], Se, Sp, t_n_opt[i], sep=","),file=filename,append=TRUE)
    if (t_n_opt[i-1] < 1 & t_n_opt[i] >= 1) {
    p_i <- i-1
    break
  }
}

paste(Se, Sp, P[p_i], sep=",")
paste("first part finished:", x)
```

Appendix A2 (SeSp)

Lastly, the R program code here calculates the optimal group size when p is unknown in the presence of test error for Squared array scheme (A2).

```
######################
# Squared array
# in the presence of error
######################

# print the index of knot (0-10)
# x <- 0
x <- as.numeric(Sys.getenv("SLURM_PROCID"))
paste("x value is:", x)
```

```
# adjusted version of our argmin function:
my_argmin2 <- function(N_max, f){
  Nmin<-1
  if (N_max>=2) {
    for (N in 2:N_max) {
      if (f(N) < f(Nmin)) {
        Nmin<-N
      }
  # if minimum is not reached up to N_max,
  # stop at the current value of p
      if (Nmin == N_max) {
# set flag to "true"
        end_p <- 1
      } else {
        end_p <- 0
      }
    }
  }
  return(c(Nmin=Nmin,end_p=end_p))
}

######
# step 1: let's take P as a sequence of same size step in interval (0 ; (3-sqrt(5))/2)
# e.g. with step of 10^(-6)

step <- 10^(-4) # second part runs too slow for 10^(-6)
start <- step
end <- (3-sqrt(5))/2
P <- seq(from = start, to = end, by=step)
m <- length(P)

# Se and Sp
Se <- seq(from = 0.9, to = 1, by=0.01) # 0.01
Sp <- Se

# let's say there are 11 knots used, then we can fix Se for each x
# fixed Se for each knot
Se <- Se[x+1]

# paste("Se[x+1] value is:", Se)

######
# step 2: Let's now take a big N, it will probably be enough to take 100.
```

```
# let's mark A = {1,2,...,N}
N = 100
A <- 1:N

# print(paste("\"Se","Sp","Nopt","p_break\"", sep="\",\""), quote = FALSE)

filename = paste("res_A2_", x,".txt",sep="")
file.create(filename)
fileConn<-file(filename)
writeLines(paste("\"Se","Sp","Nopt","p_break\"", sep="\",\""), fileConn)
close(fileConn)

# Squared array (A2)
# expected number of tests per person
t_A2 <- function(p,n,S_e,S_p) { 2/n + g(p,n,S_e,S_p) + 2*h(p,n,S_e,S_p) }
g <- function(p,n,S_e,S_p) { S_e * f(p,n,S_e,S_p) + (1-S_p-S_e) * (1-p)^n * f(p,n-1,S_e,S_p) }
f <- function(p,n,S_e,S_p) { (1-S_p) * (1-p)^n + S_e * (1-(1-p)^n) }
h <- function(p,n,S_e,S_p) {
  sum <- 0
  for(c in 0:n) {
    sum <- sum +
      gamma_0(p,n,c,S_e,S_p) * beta_0(p,n,c) +
      gamma_1(p,n,c,S_e,S_p) * beta_1(p,n,c)
  }
  return(sum)
}
gamma_0 <- function(p,n,c,S_e,S_p) { (1-S_p) * (1-S_e)^c * S_p^(n-c) }
beta_0 <- function(p,n,c) { choose(n,c) * (1-p)^(n^2-c*n+c) * (1-(1-p)^(n-1))^c }
gamma_1 <- function(p,n,c,S_e,S_p) { S_e * S_p^(n-c) * (1-S_e)^c }
beta_1 <- function(p,n,c) { choose(n,c) * (1-p)^(n^2-c*n) * (1-(1-p)^(n))^c }

######
# step 2: we then take N_opt=arg.min(t_N)

N_opt <- array(NA,dim=c(length(P)))
t_n_opt <- array(NA,dim=c(length(P),length(Se),length(Sp)))
# index of p_break
p_i <- matrix(NA, nrow = length(Se), ncol = length(Sp))

for(sen in 1:length(Se)) {
  for(spe in 1:length(Sp)) {
    i <- 1
    # N=n^2
    # pair of Nopt for each p and flag if p_break is reached
```

```
    temp_n_p <- my_argmin2(10^3, function(n)(t_A2(P[i],n,Se[sen],Sp[spe])))
# stop at current value of p if minimum is not found
    if (temp_n_p["end_p"] == 1) {
      p_i[sen,spe] <- 1
      t_n_opt[1,sen,spe] <- 1
    } else {
      # N_opt[i] <- my_argmin(10^6, function(n)(t_A2(P[i],n,Se[sen],Sp[spe])))
      N_opt[i] <- temp_n_p["Nmin"]
      t_n_opt[i,sen,spe] <- t_A2(P[i],N_opt[i],Se[sen],Sp[spe])
  # calculate p_break index where t(N,p) reaches 1
      if (t_n_opt[i,sen,spe] < 1) {
        p_i[sen,spe] <- 1
      }
      # next
      i <- 2
      last_p <- m
      while (i < last_p) {
        # N_opt[i] <- my_argmin(N_opt[i-1]+1, function(n)(t_A2(P[i],n,Se[sen],Sp[spe])))
        temp_n_p <- my_argmin2(N_opt[i-1]+1, function(n)(t_A2(P[i],n,Se[sen],Sp[spe])))
# stop at current value of p if minimum is not found
        if (temp_n_p["end_p"] == 1) {
          p_i[sen,spe] <- i
          t_n_opt[i,sen,spe] <- 1
          i <- last_p
        } else {
          N_opt[i] <- temp_n_p["Nmin"]
          t_n_opt[i,sen,spe] <- t_A2(P[i],N_opt[i],Se[sen],Sp[spe])
          # calculate p_break index where t(N,p) reaches 1
  if (t_n_opt[i-1,sen,spe] < 1 & t_n_opt[i,sen,spe] >= 1) {
            p_i[sen,spe] <- i-1
          }
          i <- i+1
        }
      }
    }
  }
}

paste("first part finished:", x)

######
# step 3: for each N from A let's calculate L_inf and L_2

t_A2_modified <- function(p,n,S_e,S_p,p_end) {
```

```
  if (p<p_end) {
    result <- t_A2(p,n,S_e,S_p)
  } else {
    result <- NA
  }
  return(result)
}

# initialise different sizes matrices
matr_t_n <- array(NA,dim=c(length(P),length(A)))
matr_Err <- array(NA,dim=c(length(P)))
L_inf <- array(NA,dim=c(length(A)))
L_2 <- array(NA,dim=c(length(A)))
N_opt_final <- array(NA,dim=c(length(Se),length(Sp)))
# calculate N_opt for each Se,Sp combination
for(sen in 1:length(Se)) {
  for(spe in 1:length(Sp)) {
    # calculate loss functions
    for (i in 1:N) {
      matr_t_n[,i] <- sapply(P, t_A2_modified,
                             n = i, S_e = Se[sen], S_p = Sp[spe],
                             p_end=p_i[sen,spe])
      matr_Err <- matr_t_n[,i] - t_n_opt[,sen,spe]
      L_inf[i] <- max(matr_Err[1:p_i[sen,spe]])
      L_2[i] <- 1/m * sum(matr_Err[1:p_i[sen,spe]]^2)
    }
    N_inf <- which.min(L_inf)
    N_2 <- which.min(L_2)
    # calculate area under curve for both loss functions
    area_inf <- 0
    area_2 <- 0
    for (p_ind in 1:p_i[sen,spe]) {
      area_inf <- area_inf + matr_t_n[p_ind,N_inf] * step
      area_2 <- area_2 + matr_t_n[p_ind,N_2] * step
    }
    # choose which loss N_opt is better
    if (area_inf<area_2) {
      N_opt_final[sen,spe] <- N_inf
    } else {
      N_opt_final[sen,spe] <- N_2
    }
    write(
      paste(Se[sen], Sp[spe], N_opt_final[sen,spe], P[p_i[sen,spe]], sep=",")
      ,file=filename,append=TRUE)
```

```
  }
}
# N_opt_final
paste("done:", x)
```

Appendix Python

After all results are gathered from R codes above, the following Python code is used for various plotting of all investigated procedures.

```
"""
Plots from saved files in R
"""

#######################################
# import packages
import pandas as pd
import matplotlib.pyplot as plt
# improve definition of images
plt.rcParams['figure.dpi'] = 300

#######################################
# Dorfman t(N_opt,p) plot

# read
t_D_opt = pd.read_csv("t_D_opt.csv")

# plot t(N_opt,p)
t_D_opt.plot(x = 'P', y = ['t_n_opt'], legend=None)
# horizontal line = 1 where individual testing becomes is more efficient
plt.axhline(y=1, color="grey", linestyle="dashed")
plt.xlabel("$p$")
plt.ylabel("Expected number of tests per person $t_D(N,p)$")
plt.title("Dorfman scheme: $t(N_{opt},p)$ for when $p$ is known",
              fontstyle='italic',fontweight="bold",fontname="Arial",
              fontsize=16)
plt.show()

#######################################
# Dorfman loss functions

# read the data
loss = pd.read_csv("D_loss.csv")

# plotting initialisation
```

```
fig = plt.figure()

# first loss function
ax = fig.add_subplot(2,1,1)
ax.plot(loss['A'],loss['L_inf'])
# vertical line marking the smallest value of the loss funciton
ax.axvline(x=8, color="red")
ax.set_xlabel("Group size")
ax.set_ylabel("Loss function $L_\infty$")

# second loss function
ax2 = fig.add_subplot(2,1,2)
ax2.plot(loss['A'],loss['L_2'])
# vertical line marking the smallest value of the loss funciton
ax2.axvline(x=4, color="red")
ax2.set_xlabel("Group size")
ax2.set_ylabel("Loss function $L_2$")

# format settings for the full plot
fig.suptitle("Loss functions for Dorman procedure",
             fontstyle='italic',fontweight="bold",fontname="Arial",
             fontsize=16)
plt.show()


########################################
# Dorfman t(N_inf,p), t(N_2,p)

# read the data
t_loss = pd.read_csv("D_t_loss.csv")

# plot
t_loss.plot(x = 'P', y = ['t_inf', 't_2'])
plt.xlabel("$p$")
plt.ylabel("Expected number of tests per person $t_D(N,p)$")
plt.legend(('based on $L_\infty$ ($N_{opt}^*=8$)','based on $L_2$ ($N_{opt}^*=4$)') , loc='best')
plt.suptitle("Graph of $p \mapsto t(N_{opt}^*,p)$",
             fontstyle='italic',fontweight="bold",fontname="Arial",
             fontsize=16)
plt.show()

########################################
# t(N_opt,p) plot
# all schemes for perfect tests
```

```
# read the data
t_D_opt = pd.read_csv("t_D_opt.csv")
t_D_ext_opt = pd.read_csv("t_D_ext_opt.csv")
t_S_opt = pd.read_csv("t_S_opt.csv")
t_H_opt = pd.read_csv("t_H_opt.csv")
t_A2_opt = pd.read_csv("t_A2_opt.csv")
t_ML_opt = pd.read_csv("t_ML_opt.csv")

# plot (x axis, y axis, text for legend)
plt.plot(t_D_opt['P'],t_D_opt['t_n_opt'], label = "Dorfman (D)")
plt.plot(t_D_ext_opt['P'],t_D_ext_opt['t_n_opt'], label = "Dorfman extension (D')")
plt.plot(t_S_opt['P'],t_S_opt['t_n_opt'], label = "Sterrett (S)")
plt.plot(t_H_opt['P'],t_H_opt['t_n_opt'], label = "Halving (H)")
plt.plot(t_A2_opt['P'],t_A2_opt['t_n_opt'], label = "Squared array (A2)")
plt.plot(t_ML_opt['P'],t_ML_opt['t_n_opt'], label = "Generalised ML")
# vertical lines of p break points for each
plt.axvline(x=max(t_D_opt['P']), color="blue", linewidth=0.5, linestyle="dashed")
plt.axvline(x=max(t_D_ext_opt['P']), color="orange", linewidth=0.5, linestyle="dashed")
plt.axvline(x=max(t_S_opt['P']), color="green", linewidth=0.5, linestyle="dashed")
plt.axvline(x=max(t_H_opt['P']), color="pink", linewidth=0.5, linestyle="dashed")
plt.axvline(x=max(t_A2_opt['P']), color="violet", linewidth=0.5, linestyle="dashed")
plt.axvline(x=max(t_ML_opt['P']), color="brown", linewidth=0.5, linestyle="dashed")
# horizontal line = 1 where individual testing becomes is more efficient
plt.axhline(y=1, color="grey", linestyle="dashed")
# format settings for the full plot
plt.xlabel("$p$")
plt.ylabel("Expected number of tests per person $t(N_{opt},p)$")
plt.legend()
plt.title("$t(N_{opt},p)$ for when $p$ is known",
              fontstyle='italic',fontweight="bold",fontname="Arial",
              fontsize=16)
plt.show()


########################################
# t(N_opt,p) plot

# read
hier0 = pd.read_csv("res_t_0.txt")
hier1 = pd.read_csv("res_t_1.txt")
hier2 = pd.read_csv("res_t_2.txt")
hier3 = pd.read_csv("res_t_3.txt")
hier4 = pd.read_csv("res_t_4.txt")
```

```
hier5 = pd.read_csv("res_t_5.txt")
hier6 = pd.read_csv("res_t_6.txt")
hier7 = pd.read_csv("res_t_7.txt")
hier8 = pd.read_csv("res_t_8.txt")

# plot
plt.plot(hier0['p'],hier0['tNopt'],
         label = "Se = {} and Sp = {}".format(hier0['Se'][0],hier0['Sp'][0]),
         linewidth=0.5)
plt.plot(hier1['p'],hier1['tNopt'],
         label = "Se = {} and Sp = {}".format(hier1['Se'][0],hier1['Sp'][0]),
         linewidth=0.5)
plt.plot(hier2['p'],hier2['tNopt'],
         label = "Se = {} and Sp = {}".format(hier2['Se'][0],hier2['Sp'][0]),
         c="red", linewidth=2)
plt.plot(hier3['p'],hier3['tNopt'],
         label = "Se = {} and Sp = {}".format(hier3['Se'][0],hier3['Sp'][0]),
         linewidth=0.5)
plt.plot(hier4['p'],hier4['tNopt'],
         label = "Se = {} and Sp = {}".format(hier4['Se'][0],hier4['Sp'][0]),
         linewidth=0.5)
plt.plot(hier5['p'],hier5['tNopt'],
         label = "Se = {} and Sp = {}".format(hier5['Se'][0],hier5['Sp'][0]),
         linewidth=0.5)
plt.plot(hier6['p'],hier6['tNopt'],
         label = "Se = {} and Sp = {}".format(hier6['Se'][0],hier6['Sp'][0]),
         c="blue", linewidth=2)
plt.plot(hier7['p'],hier7['tNopt'],
         label = "Se = {} and Sp = {}".format(hier7['Se'][0],hier7['Sp'][0]),
         linewidth=0.5)
plt.plot(hier8['p'],hier8['tNopt'],
         label = "Se = {} and Sp = {}".format(hier8['Se'][0],hier8['Sp'][0]),
         c="black", linewidth=2)
plt.axhline(y=1, color="grey", linestyle="dashed")
plt.xlabel("$p$")
plt.ylabel("Expected number of tests per person $t_{D2}(N,p)$")
plt.legend()
plt.title("D2: $t(N_{opt}(p)),p)$ when $p$ is known",
              fontstyle='italic',fontweight="bold",fontname="Arial",
              fontsize=16)
plt.show()
```