

Nicolas Thierry-Mieg

Laboratoire Logiciels-Systèmes-Réseaux, Institut d'Informatique et de Mathématiques Appliquées de Grenoble, BP53,
38041 GRENOBLE Cedex 9, France

Abstract: *In the context of high-throughput screening projects where the goal is the identification of low-frequency positives, pooling has only advantages and no drawbacks: it reduces the number of tests and yet provides critical duplication of the individual experiments, thereby correcting for experimental noise and allowing secure identification of both positive and negative occurrences. The main difficulty consists in designing the pools in a manner that is both efficient and robust: this is known as the group testing problem, or pooling problem. In this paper, we present a new non-adaptive combinatorial pooling design: the "shuffled transversal design" (STD), which relies on prime numbers, and we prove that it allows unambiguous decoding of noisy experimental observations. This design is highly flexible, and can be tailored to function robustly in a wide range of experimental settings (i.e., numbers of objects, fractions of positives, and expected error-rates). Furthermore, we show that STD compares favorably, in terms of efficiency, to the previously described non-adaptive combinatorial pooling designs.*

Keywords: High-throughput screening, experimental design, pooling, group testing

1 Introduction

With the availability of complete genome sequences, biology has entered a new era. Relying on the sequencing data of genomes, transcriptomes or proteomes, scientists have been developing high-throughput screening assays and undertaking a variety of large scale functional genomics projects. While some projects involve quantitative measurements, others consist in applying a basic yes-or-no test to a large collection of samples or "objects", – be they individuals, clones, cells, drugs, nucleic acid fragments, proteins, peptides... A large class of these binary tests aims at identifying relatively rare events. The main goal is of course to obtain information as efficiently and as reliably as possible. Typically, this is achieved by minimizing the cost of the basic assay in terms of time and money, and automating and parallelizing the experiments as much as possible. A major difficulty stems from the fact that high-throughput biological assays are usually somewhat noisy: reproducibility is a known problem of microarray analyses, and both false positive and false negative observations are to be expected in binary type experiments. These experimental artifacts should be identified and properly treated. A clean way to deal with the issue consists in repeating all tests several times, but this is usually prohibitively expensive and time-consuming. A more practical approach, in the case of binary tests, consists in retesting all positive results obtained in a first round. This strategy identifies most of the false positives at a reduced cost, but is powerless with regard to false negatives, leaving us in need of a better solution.

In the case of binary experiments testing for rare events, an intuitively appealing strategy consists in pooling the samples to minimize the number of tests. It requires three conditions. First, the objects under scrutiny must be available individually, in a tagged form. For example, a cDNA library in bulk is not exploitable, but a collection of cDNA clones or of cloned coding regions, such as the one produced by the *C. elegans* ORFeome project [18], is fine. Second, it must be possible to test a pool of objects in a single assay and obtain a positive readout if at least one of the objects is positive. For example, this is the case when searching for a specific DNA sequence by PCR in a mixture of molecules: a product will be amplified if at least one of the pooled molecules contains the target sequence. Third, pooling is especially desirable and efficient when the fraction of expected positives is small (at most a few percent).

Under these conditions, pooling strategies can be applied, and the difficulty then consists in choosing a "good" set of pools. This being an intuitive but rather vague goal, it must be formalized. An initial simple formulation, known as the group testing problem (or pooling problem), is the following. Consider a set of n events which can be true or false, represented by n Boolean variables. Let us call "pool" a subset of variables. We define the value of a pool as the disjunction (i.e., the logical OR operator) of the variables that it contains. Let us assume that at most t variables are true. The goal is to build a set of v pools, where v is small compared to n , such that by testing the values of the v pools, one can unambiguously determine the values of the n variables.

If the pools must be specified in a single step, rather than incrementally by building on the results of previous tests, the problem is called "non-adaptive". Although adaptive designs can require fewer tests, non-

adaptive pooling designs are often better suited to high-throughput screening projects because they allow parallelization and facilitate automation of the experiments, and also because the same pools can be used for all targets, thereby reducing the total project cost.

The ability to deal with noisy observations is an important added benefit to using a pooling system, compared to the classical individual testing strategy. Indeed, noise detection and correction capabilities are inherent in any pooling system, because each variable is present in several pools, hence tested many times. Depending on the expected noise level, the redundancy can be chosen at will, and simply testing a few more pools than would be necessary in the absence of noise results in robust error-correction. It should be noted that minimization of the number of pools and noise correction are two conflicting goals: increasing noise tolerance generally requires testing more pools. Designing a set of pools requires balancing these two objectives, and finding the right compromise to suit the application.

Other application-dependent constraints may be imposed. In particular, the pool sizes are often limited by the experimental setting. For example, in the context of the *C. elegans* protein interaction mapping project led by Marc Vidal [21, 8], it is estimated that, using their high-throughput two-hybrid protocol, reliable readouts can be obtained with pools containing 400 AD-Y clones, or perhaps up to 1000 by tweaking the assay (Marc Vidal, personal communication).

Many groups have used with some success variants of the simple "grid" design, which consists in arraying the objects on a grid and pooling the rows and columns [e.g. 10, 22, 6]. However, although it is better than no pooling, this rudimentary design is vulnerable to noise and behaves poorly when several objects are positive, in addition to being far from optimal in terms of numbers of tests.

In answer to its shortcomings, more sophisticated error-correcting pooling designs have been proposed (see section 4.2). Some of these designs are very efficient in terms of numbers of tests, but lack the robustness and flexibility that most real biological applications require. Others are more adaptable and noise-tolerant at the expense of performance.

In this paper, we present a new pooling algorithm: the "shuffled transversal design" (STD). This design is highly flexible: it can be tailored to allow the identification of any number of positive objects and to deal with important noise levels. Yet it is extremely efficient in terms of number of tests, and we show that it compares favorably to the previously described pooling designs.

The paper is organized as follows. STD is defined in section 2. In section 3, it is shown to be an error-correcting solution to the pooling problem. In section 4, the theoretical performance of STD is evaluated and compared with the main previously described deterministic pooling designs. Finally, section 5 summarizes the results and discusses future directions.

2 The Shuffled Transversal Design

2.1 Preliminaries

The following notations are used throughout this paper, in accordance with the notations from [1].

Let $n \geq 2$, and consider the set $\mathcal{A}_n = \{A_0, \dots, A_{n-1}\}$ of n Boolean variables.

We will call "pool" a subset of \mathcal{A}_n . We say that a pool is "true", or "positive", if at least one of its elements is true.

Let us call "layer" a partition of \mathcal{A}_n .

Let q be a prime number, with $q < n$.

We define the "compression power" of q relative to n , noted $\Gamma(q,n)$, as the smallest integer γ such that $q^{\gamma+1} \geq n$. We will simply write Γ for $\Gamma(q,n)$ whenever possible.

Let σ_q be the mapping of $\{0,1\}^q$ onto itself defined by:

$$\forall (x_1, \dots, x_q) \in \{0,1\}^q, \quad \sigma_q \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_q \end{bmatrix} = \begin{bmatrix} x_q \\ x_1 \\ \vdots \\ x_{q-1} \end{bmatrix}.$$

Note that σ_q is a cyclic function of order q : σ_q^q is the identity function on $\{0,1\}^q$.

The matrix representation: Any set of pools can be represented by a Boolean matrix, as follows. Each column corresponds to one variable, and each line to one pool. The cell (i,j) is true (value 1) if pool i contains variable j , and false (value 0) otherwise.

Example: Consider the $n=9$ variables $\mathcal{A}_9 = \{A_0, A_1, \dots, A_8\}$. The following matrix defines a set of 3 pools:

$$M_0 = \begin{bmatrix} 1 & 0 & 0 & 1 & 0 & 0 & 1 & 0 & 0 \\ 0 & 1 & 0 & 0 & 1 & 0 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 & 0 & 1 & 0 & 0 & 1 \end{bmatrix}$$

The pools are $\{A_0, A_3, A_6\}$ (defined by the first line), $\{A_1, A_4, A_7\}$ (second line), and $\{A_2, A_5, A_8\}$ (third line). In fact, this set of pools clearly constitutes a layer.

2.2 Definition of STD

A pooling design is a method to construct a set of pools. When the set of pools can be partitioned into subsets, each forming a partition of the set of variables, the pooling design is said to be "transversal". STD is a transversal pooling design: given a prime number q with $q < n$, and k such that $k \leq q+1$, it constructs a set $\text{STD}(n; q; k)$ of pools composed of k layers. The first q possible layers have a unified construction: they each contain q pools of n/q or $(n/q)+1$ variables, and are globally interchangeable. The last buildable layer has a specific construction, and is less regular, yet it complements the others nicely. The intuitive idea behind STD consists in shuffling the variables as much as possible between any two layers, in such a way that the co-occurrences of variables (i.e., the number of pools in which any pair of variables can occur) is limited, and the intersections between pools are of roughly constant size. A formal construction of $\text{STD}(n; q; k)$ follows.

For every $j \in \{0, \dots, q\}$, let M_j be a $q \times n$ Boolean matrix, defined by its columns $C_{j,0}, \dots, C_{j,n-1}$ as follows:

$$C_{0,0} = \begin{bmatrix} 1 \\ 0 \\ \vdots \\ 0 \end{bmatrix}, \text{ and } i \in \{0, \dots, n-1\} \quad C_{j,i} = \sigma_q^{s(i,j)}(C_{0,0}) \text{ where:}$$

$$s(i,j) = \sum_{c=0}^{\Gamma} j^c \cdot \left\lfloor \frac{i}{q^c} \right\rfloor \text{ if } j < q, \text{ and } s(i,q) = \left\lfloor \frac{i}{q^\Gamma} \right\rfloor, \text{ where the semi-bracket denotes the integer part.}$$

Let $L(j)$ be the set of pools of which M_j is the matrix representation. Note that each column $C_{j,i}$ has exactly one occurrence of '1' and $(q-1)$ occurrences of '0'. The index of the '1' identifies the (single) pool of $L(j)$ which contains variable A_i . Therefore, in a given set of pools $L(j)$, each variable is present in exactly one pool, that is to say $L(j)$ constitutes a partition of \mathcal{A}_n : $L(j)$ is a layer.

Finally, for $k \in \{1, 2, \dots, q+1\}$, $\text{STD}(n; q; k)$ is defined as: $\text{STD}(n; q; k) = \bigcup_{j=0}^{k-1} L(j)$.

Example: Consider again the variables \mathcal{A}_9 , and let $q=3$ (hence $\Gamma=1$). M_0 is as defined in 2.1, and M_1, M_2, M_3 are:

$$M_1 = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 1 & 0 & 1 & 0 \\ 0 & 1 & 0 & 1 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 & 1 & 0 & 1 & 0 & 0 \end{bmatrix},$$

$$M_2 = \begin{bmatrix} 1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 \\ 0 & 1 & 0 & 0 & 0 & 1 & 1 & 0 & 0 \\ 0 & 0 & 1 & 1 & 0 & 0 & 0 & 1 & 0 \end{bmatrix},$$

$$M_3 = \begin{bmatrix} 1 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 1 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 1 \end{bmatrix}.$$

The corresponding layers of pools are the following:

Layer 0: $L(0) = \{\{A_0, A_3, A_6\}, \{A_1, A_4, A_7\}, \{A_2, A_5, A_8\}\}$

Layer 1: $L(1) = \{\{A_0, A_5, A_7\}, \{A_1, A_3, A_8\}, \{A_2, A_4, A_6\}\}$

Layer 2: $L(2) = \{\{A_0, A_4, A_8\}, \{A_1, A_5, A_6\}, \{A_2, A_3, A_7\}\}$

Layer 3: $L(3) = \{\{A_0, A_1, A_2\}, \{A_3, A_4, A_5\}, \{A_6, A_7, A_8\}\}$.

STD(9; 3; 2) is the following set of pools: $STD(9; 3; 2) = L(0) \cup L(1)$.

Remark: The method builds at most $q+1$ layers: indeed, if we discard the last particular layer $L(q)$ and attempt to extend the STD construction to any j , it becomes cyclic of order q : for every j , $L(j+q) = L(j)$.

3 Properties of STD

In this section, we establish an important theorem, leading to two corollaries which show that STD constitutes a solution to the pooling problem described in section 1, and that it can be used to detect and correct noisy observations. We then establish another property of STD, which is noteworthy albeit not directly related to the pooling problem.

3.1 Co-occurrence of Variables

So far we have considered the variables that are contained in a given pool. Dually, we may consider the set of pools that contain a given variable. For $k \in \{1, 2, \dots, q+1\}$, we will note $pools_k(i)$ the set of pools of STD(n ; q ; k) that contain variable A_i : $i \in \{0, \dots, n-1\}$, $pools_k(i) = \{p \in STD(n; q; k) \mid A_i \in p\}$.

Theorem 1: Recall that q is prime.

$$i_1, i_2 \in \{0, \dots, n-1\}, [i_1 i_2] \in [Card(pools_{q+1}(i_1) \cap pools_{q+1}(i_2)) \quad \Gamma(q, n)].$$

Proof: Let $i_1, i_2 \in \{0, \dots, n-1\}$ with $i_1 \neq i_2$. Since each layer of pools is a partition of \mathcal{A}_n , there cannot be more than one pool per layer containing both A_{i_1} and A_{i_2} . Furthermore, there exists a pool in layer $L(j)$ that contains both A_{i_1} and A_{i_2} if and only if the columns for A_{i_1} and A_{i_2} are equal in M_j , that is to say $C_{j, i_1} = C_{j, i_2}$. Therefore the number of pools of STD(n ; q ; $q+1$) that contain both i_1 and i_2 , $Card(pools_{q+1}(i_1) \cap pools_{q+1}(i_2))$, is the number of values of j in $\{0, \dots, q\}$ such that $C_{j, i_1} = C_{j, i_2}$. However, the following equivalencies hold $j \in \{0, \dots, q-1\}$:

$$\begin{aligned} C_{j, i_1} = C_{j, i_2} &\Leftrightarrow s(i_1, j) \equiv s(i_2, j) \pmod{q} \\ &\Leftrightarrow \sum_{c=0}^{\Gamma} j^c \cdot \left\lfloor \frac{i_1}{q^c} \right\rfloor \equiv \sum_{c=0}^{\Gamma} j^c \cdot \left\lfloor \frac{i_2}{q^c} \right\rfloor \pmod{q} \\ &\Leftrightarrow \sum_{c=0}^{\Gamma} j^c \cdot \left(\left\lfloor \frac{i_1}{q^c} \right\rfloor - \left\lfloor \frac{i_2}{q^c} \right\rfloor \right) \equiv 0 \pmod{q} \quad (1) \end{aligned}$$

Since q is prime, $\mathbb{Z}/q\mathbb{Z}$ is a field, namely the Galois field $GF(q)$.

Furthermore, since $i_1 i_2$, there exists at least one value $c \in \{0, \dots, \Gamma\}$ such that $\left(\left\lfloor \frac{i_1}{q^c} \right\rfloor - \left\lfloor \frac{i_2}{q^c} \right\rfloor\right) \not\equiv 0 \pmod{q}$.

Indeed, $i_1, i_2 \in \{0, \dots, n-1\}$ and $n \leq q^{\Gamma+1}$ entails that $i_1 = \sum_{c=0}^{\Gamma} \left(\left\lfloor \frac{i_1}{q^c} \right\rfloor \% q\right) \cdot q^c$ and $i_2 = \sum_{c=0}^{\Gamma} \left(\left\lfloor \frac{i_2}{q^c} \right\rfloor \% q\right) \cdot q^c$, where $\%$ denotes the modulus (these are the unique decompositions of i_1 and i_2 in base q). Hence, $i_1 - i_2 = \sum_{c=0}^{\Gamma} \left(\left(\left\lfloor \frac{i_1}{q^c} \right\rfloor - \left\lfloor \frac{i_2}{q^c} \right\rfloor\right) \% q\right) \cdot q^c$. Supposing that $\left(\left\lfloor \frac{i_1}{q^c} \right\rfloor - \left\lfloor \frac{i_2}{q^c} \right\rfloor\right) \equiv 0 \pmod{q}$ for every $c \in \{0, \dots, \Gamma\}$ would lead to $i_1 - i_2 = 0$, which is contradictory with the hypothesis that $i_1 i_2$.

It follows that the above (1) can be seen as a non-zero polynomial (in j) of degree at most Γ on $GF(q)$. As is well-known, such a polynomial has at most Γ roots in $GF(q)$. That is to say, there are at most Γ values of j in $\{0, \dots, q-1\}$ such that a pool of $L(j)$ contains both A_{i_1} and A_{i_2} . This proves the theorem if $C_{q, i_1} \neq C_{q, i_2}$. Furthermore, if $C_{q, i_1} = C_{q, i_2}$, the coefficient of j^Γ in (1) is zero by definition of $s(i, q)$, and (1) is of degree at most $\Gamma-1$. Therefore if A_{i_1} and A_{i_2} are elements of the same pool in $L(q)$, then there are at most $\Gamma-1$ pools in $L(0), \dots, L(q-1)$ that contain both A_{i_1} and A_{i_2} . This concludes the proof of the theorem.

Example: Consider again the example $n=9, q=3, k=4$, for which the layers $L(0), L(1), L(2)$, and $L(3)$ are known (see 2.2). The set of pools containing A_0 is:

$$\text{pools}_4(0) = \{\{A_0, A_3, A_6\}, \{A_0, A_5, A_7\}, \{A_0, A_4, A_8\}, \{A_0, A_1, A_2\}\}.$$

One can easily see that A_0 is present exactly once with every other variable. In fact, each pair of variables is present in exactly 1 ($=\Gamma(3,9)$) pool, in conformity with theorem 1.

Remark: The property holds a fortiori when $k < q+1$, i.e. when considering $\text{STD}(n; q; k)$ instead of $\text{STD}(n; q; q+1)$.

3.2 A Solution to the Pooling Problem

Corollary 1: Let t be an integer such that $t\Gamma(q, n) \leq q$. Let $k=t\Gamma+1$, and consider the set of pools $\text{STD}(n; q; k)$. Suppose that the value of each pool has been observed, and that there are at most t positive variables in \mathcal{A}_n . Then, in the absence of noise (i.e., if all pool values are correctly observed), the value of every variable can be identified.

Proof: Consider the following algorithm, which tags variables as negative or positive.

Algorithm 1: all the variables present in at least one negative pool are tagged negative; any variable present in at least one positive pool where all other variables have been tagged negative, is tagged positive.

We show that this algorithm correctly identifies the value of each and every variable.

Let A_i be a negative variable. A_i is present in exactly k pools: one pool in each layer. Theorem 1 asserts that no variable other than A_i is present in more than Γ of these $t\Gamma+1$ pools. Therefore, since at most t variables are positive, A_i is present in at least one pool where no positive variable is present. Consequently, examination of this pool yields a negative answer (since all observations are correct), which leads algorithm 1 to tag A_i negative. This shows that every negative variable is correctly identified.

Now let A_j be a positive variable. Since there are no observational errors, all pools containing A_j are positive. Again according to theorem 1, no other variable is present in more than Γ of these $t\Gamma+1$ pools. Therefore, since there are at most $t-1$ other positive variables, A_j is present in at least $\Gamma+1$ pools where all other variables are negative (and have been tagged negative, according to the above paragraph). This shows that every positive variable is also identified, and therefore proves the corollary.

Example: Consider again our example $\text{STD}(9; 3; 2) = \{\{A_0, A_3, A_6\}, \{A_1, A_4, A_7\}, \{A_2, A_5, A_8\}, \{A_0, A_5, A_7\}, \{A_1, A_3, A_8\}, \{A_2, A_4, A_6\}\}$. Let $t=1$, and suppose that a single variable in \mathcal{A}_9 is positive. For reasons of symmetry, the name of that variable is inconsequential: all are equivalent. Let us suppose that the only positive variable is A_8 . Then pools $\{A_0, A_3, A_6\}$, $\{A_1, A_4, A_7\}$, $\{A_0, A_5, A_7\}$, and $\{A_2, A_4, A_6\}$ are negative, which shows that variables A_0, A_1, \dots, A_7 are negative; and pools $\{A_2, A_5, A_8\}$ and $\{A_1, A_3, A_8\}$ are positive,

which each prove that A_8 is positive (given that A_2, A_5, A_1 and A_3 have been shown to be negative).

Remark: If more than t variables are positive, this fact is revealed: clearly, at most $n-(t+1)$ variables are tagged negative, contrary to when there are at most t positives. In fact, all tags produced by the above algorithm are still correct, but some variables may not be tagged at all: these variables are called "unresolved", or "ambiguous".

3.3 Dealing with Noise

As stated in the introduction, pooling designs have an intrinsic potential for noise-correction, due to the redundancy of variables. In the case of STD, this potential can be taken advantage of by simply testing a few extra layers of pools and using a modified algorithm, as shown here.

Corollary 2: Let t and E be integers such that $t\Gamma(q,n)+2E \leq q$, and let $k=t\Gamma+2E+1$. Consider the set of pools $\text{STD}(n; q; k)$, and suppose that the value of each pool has been observed. Furthermore, suppose that there are at most t positive variables in \mathcal{A}_n , and that there are at most E observation errors. Then, all errors can be detected and corrected, and the value of every variable can be identified.

Proof: Consider the following tagging algorithm.

Algorithm 2: all the variables present in at least $E+1$ negative pools are tagged negative; any variable present in at least $E+1$ positive pools where all other variables have been tagged negative, is tagged positive.

The proof is similar to that of corollary 1 (see section 3.2): we show that algorithm 2 correctly tags every variable. In this case, theorem 1 shows that each negative variable is necessarily present in at least $2E+1$ negative pools. Since there are at most E observation errors, it follows that at least $E+1$ of these negative pools are correctly observed. Therefore, algorithm 2 tags all negative variables correctly. Conversely, a positive variable A_i appears in at least $t\Gamma+E+1$ positive pools (since there are at most E errors), of which at most $(t-1)\Gamma$ contain at least one other positive variable (according to theorem 1). Therefore A_i is present in at least $(t\Gamma+E+1) - (t-1)\Gamma = \Gamma+E+1$ positive pools where all other variables are negative. Since these negative variables have been correctly tagged as such (as shown above), A_i is tagged positive. This shows that algorithm 2 also correctly tags all positive variables.

Finally, any observation which is contradictory with the obtained tagging is necessarily erroneous. In other words, false negative and false positive observations are identified.

Remarks:

1. Note that few restrictions are imposed when choosing the value of the parameter q : it must simply be a prime number smaller than n . Consequently, STD can be used successfully even when very high noise levels are expected, by picking a large value for q . Of course, as is to be expected in low signal-to-noise situations, this high corrective power comes at the price of lower compression performance, since larger q values mean more pools per layer.

2. As in the noiseless case discussed in section 3.2, algorithm 2 still tags variables correctly if more than t variables are positive, although some variables may be unresolved. However, if there are more than E observation errors, problems may occur (e.g., a positive variable might be "mis-tagged" negative, or a variable might be tagged both positive and negative). Therefore, in real applications where the number of errors will probably exceed E in at least a few instances, more sophisticated tagging algorithms should be used (paper in preparation).

3.4 Even Redistribution of Variables

We have just shown that STD constitutes a solution to the pooling problem in the presence of experimental noise. Although it digresses from the main focus of this paper, the following theorem provides an interesting characterization of STD, basically showing that the STD layers work well together, information-wise.

Theorem 2: Let $m \leq k \leq q$ and consider a set of m pools $\{P_1, \dots, P_m\} \subset \text{STD}(n; q; k)$, each belonging to a different layer. Then:

$$\lambda_m \leq \left| \bigcap_{h=1}^m P_h \right| \leq \lambda_m + 1, \text{ where } \lambda_m = \sum_{c=m}^{\Gamma} \left[\left\lfloor \frac{n-1}{q^c} \right\rfloor \% q \right] \cdot q^{c-m}.$$

Proof: Let $j_1, \dots, j_m \in \{0, \dots, q-1\}$ be the layer numbers and $p_1, \dots, p_m \in \{0, \dots, q-1\}$ be the pool indexes that define $\{P_1, \dots, P_m\}$: for every $h \in \{1, \dots, m\}$, P_h contains all variables of index $i \in \{0, \dots, n-1\}$ such that $s(i, j_h) \equiv p_h \pmod{q}$.

$\left| \bigcap_{h=1}^m P_h \right|$ is the number of values $i \in \{0, \dots, n-1\}$ such that: $\forall h \in \{1, \dots, m\}$, $s(i, j_h) \equiv p_h \pmod{q}$. Writing $i = \sum_{c=0}^{\Gamma} \alpha_c \cdot q^c$ with $\alpha_0, \dots, \alpha_{\Gamma} \in \{0, \dots, q-1\}$ (this is the unique decomposition of i in base q), the above is equivalent to:

$$\forall h \in \{1, \dots, m\} \sum_{c=0}^{\Gamma} \alpha_c \cdot j_h^c \equiv p_h \pmod{q} \quad (2)$$

This system can be written:

$$\begin{bmatrix} 1 & j_1 & j_1^2 & \cdots & j_1^{\Gamma} \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ 1 & j_m & j_m^2 & \cdots & j_m^{\Gamma} \end{bmatrix} \cdot \begin{bmatrix} \alpha_0 \\ \vdots \\ \alpha_{\Gamma} \end{bmatrix} \equiv \begin{bmatrix} p_1 \\ \vdots \\ p_m \end{bmatrix} \pmod{q}.$$

If $m \geq \Gamma+1$: consider the square sub-matrix composed of the first $\Gamma+1$ rows of the left member. Since P_1, \dots, P_m belong to different layers, the j_h values are all distinct. Therefore, recalling that q is prime, this sub-matrix can be seen as a Vandermonde matrix with elements in the Galois field $GF(q)$: it is nonsingular. This shows the existence of a unique tuple of values for $\alpha_0, \dots, \alpha_{\Gamma} \in \{0, \dots, q-1\}$ satisfying the first $\Gamma+1$ congruences of (2). The remaining $m-(\Gamma+1)$ congruences may or may not be satisfied with these $\alpha_0, \dots, \alpha_{\Gamma}$ values, and the corresponding $i = \sum_{c=0}^{\Gamma} \alpha_c \cdot q^c$ might be too large (i.e. $\geq n$); but in any case, there is at most one value of i satisfying the system: the theorem is proved when $m \geq \Gamma+1$ (given that in this case $\lambda_m = 0$).

Otherwise, $m \leq \Gamma$: consider the square sub-matrix composed of the first m columns of the left member. Again, this sub-matrix is a Vandermonde matrix in $GF(q)$, hence it is nonsingular. Consequently, given any values for $\alpha_m, \dots, \alpha_{\Gamma}$, there exists a unique tuple of values for $\alpha_0, \dots, \alpha_{m-1}$ in $\{0, \dots, q-1\}$ satisfying (2) (simply shift the terms in $\alpha_m, \dots, \alpha_{\Gamma}$ to the right member). The question therefore becomes: how many tuples of values

for $\alpha_m, \dots, \alpha_{\Gamma}$ exist, such that $i = \sum_{c=0}^{\Gamma} \alpha_c \cdot q^c < n$, where $\alpha_0, \dots, \alpha_{m-1}$ are determined by $\alpha_m, \dots, \alpha_{\Gamma}$ as explained

above. To answer this, consider the unique decomposition of $n-1$ in base q : $n-1 = \sum_{c=0}^{\Gamma} \beta_c \cdot q^c$, where

$$\beta_c = \left\lfloor \frac{n-1}{q^c} \right\rfloor \% q \text{ for } c \in \{0, \dots, \Gamma\}. \text{ Under this representation, it is clear that } i < n, \text{ i.e. } i \leq n-1, \text{ if and only if:}$$

$$\alpha_{\Gamma} < \beta_{\Gamma} \text{ or}$$

$$(\alpha_{\Gamma} = \beta_{\Gamma} \text{ and } (\alpha_{\Gamma-1} < \beta_{\Gamma-1} \text{ or}$$

$$(\dots \text{ and } (\alpha_m < \beta_m \text{ or}$$

$$(\alpha_m = \beta_m \text{ and } \sum_{c=m}^{\Gamma} \alpha_c \cdot q^c < n) \dots))).$$

For each $c \in \{m, \dots, \Gamma\}$, the branch ending at $(\alpha_c < \beta_c)$ yields $\beta_c \cdot q^{(c-m)}$ different tuples. Indeed, for $d > c$ $\alpha_d = \beta_d$ in this branch, and $\alpha_0, \dots, \alpha_{m-1}$ are bound to $\alpha_m, \dots, \alpha_{\Gamma}$: there are β_c possible choices for α_c , and q choices each for $\alpha_m, \dots, \alpha_{c-1}$. As to the final branch, it can yield at most one solution, given that all the α values are set or bound in this branch.

Consequently, there are a total of $\lambda_m = \sum_{c=m}^{\Gamma} \beta_c \cdot q^{c-m}$ or λ_m+1 solutions: the theorem is also proved when $m \leq \Gamma$.

Remarks:

1. λ_m depends only on m and not on the choice of P_1, \dots, P_m ; hence this theorem can be expressed simply as follows: each pool is redistributed evenly in every other layer, and furthermore the intersection between any two or more pools from different layers is also redistributed evenly in the remaining layers. This property is very interesting because it means that knowing that any given pool is positive doesn't bring any information regarding which pools of another layer will be positive; hence, the information content of the other layers remains high.

2. Note that the theorem specifies $k \leq q$ rather than $q+1$: the last layer that can be built with STD, $L(q)$, is particular and does not satisfy theorem 2.

4. Theoretical Evaluation of STD

To evaluate and compare pooling designs, a fair performance measure is needed. A widely-used and reasonable choice consists in considering the number of pools required to guarantee the correction of all errors and the identification of all variables' values: we call this the "guarantee requirement". This criterion is used here to study the behavior and performance of STD, and to compare it to the main published deterministic error-correcting pooling designs.

4.1 Guaranteed Performance of STD

We define the "gain" of a design as the ratio between the number of variables and the number of pools: n/v . The gain is called "guaranteed gain" if the guarantee requirement is satisfied. This measure is particularly useful for comparing settings where n varies.

Given the specifications of an application, i.e. values for n (total number of objects to be tested), t (number of expected positives), and E (expected number of errors to be corrected), STD can propose many sets of pools, by selecting various values for the parameter q and setting the number of layers k accordingly (as specified by section 3.3). These pool sets are of different sizes, but all satisfy the guarantee requirement. The optimal choice, q_{opt} , is the one with maximum guaranteed gain. Let q_{min} be the smallest possible q such that $t(q,n)+2E \leq q$, and let $q_{max} = q_{min}, n$. At a fixed value for n , the number of layers k necessary to satisfy the guarantee requirement is constant; therefore the best gain at fixed n is always obtained with the smallest q whose compression is ≥ 1 . It follows that q_{opt} can be identified easily by finding the smallest q for each value of n in $\{1, \dots, q_{max}\}$, and calculating the corresponding gain. In practice we often have $q_{opt} = q_{min}$, but this is not compulsory, as illustrated by Table 1 in the case $n=10\ 000$, $t=5$, $E=0$.

q	k	v	gain
≤ 13	≥ 3	≥ 16	$k > q+1$, can't use these values
17	3	16	272
19	3	16	304
23	2	11	253
29	2	11	319
...	2	11	...
97	2	11	1067
101	1	6	606

Table 1: Choosing the optimal value for the number of pools per layer, q . This table shows the gains obtained with various q values, when the total number of variables to be tested is $n=10000$ and the number of expected positives is $t=5$, in a noiseless experiment ($E=0$). q is the compression power (i.e. logarithm of n in base q , see section 2.1), k is the number of layers, v is the number of pools (i.e. kq), and the gain is defined as n/v . By construction, STD requires $k \leq q+1$; and to guarantee the identification of t positives while correcting E errors, section 3.3 showed that we must choose $k=t+2E+1$; in this example, $k=5+1$. Often, the smallest useable q (i.e., satisfying $k \leq q+1$), q_{min} , yields the highest gain, but this is not always the case. In this example, $q_{min}=17$, but $q=23$ (smallest q such that $k=2$) yields the highest gain: 39.5.

The above method allows to easily calculate the best guaranteed gain that STD can offer, in any specified

(n, t, E) setting. Therefore, the behavior of STD can be studied under various angles. In particular, one interesting approach consists in using fixed values for t and E , and studying the evolution of the best guaranteed gain (obtained using q_{opt}) when n increases. For example, Table 2 displays the number of pools necessary to identify three positives and correct two errors, when the number of variables ranges from 100 to 10^6 . When n increases, the gain increases substantially and fairly regularly: it is multiplied by a factor ranging from 6 to 9 every time n gains an order of magnitude. Note that in a real application, the fact that the pool sizes are generally constrained by practical considerations can result in forcing to use values of $q > q_{\text{opt}}$ and hence limit the gain.

N	q_{opt}	pool size	k	v	gain
100	11	9	8	88	1.1
1000	11	91	11	121	8.3
10^4	13	769	14	182	55
10^5	19	5263	14	266	376
10^6	19	52631	17	323	3096

Table 2: Gains obtained when the identification of 3 positives and the correction of 2 errors is guaranteed ($t=3, E=2$). For each value of n (total number of variables), the optimal q value q_{opt} has been calculated, as well as the associated pool size, the number of layers k , the total number of pools v , and the gain.

4.2 Comparison with previous work

In this section, after a brief overview of the known construction methods, we compare STD, in terms of flexibility and performance under the guarantee requirement, to the main published error-correcting deterministic pooling designs. In general, the guaranteed gains can be difficult to compare analytically, because the numbers of pools and variables can be defined by formulas that are often rather involved. However, each paper describing a new design typically holds a numerical example, which would hardly be disadvantageous to the described design. Therefore, when the methods cannot be easily compared, it seems fair to use each paper's numerical example for comparison with STD. Note that the guarantee requirement cannot be satisfied by random designs [e.g. 5], which are consequently not studied here.

Detailed reviews of deterministic pooling designs can be found in [1, 16, 9], and we will only very briefly recapitulate them here. Broadly speaking, there are three main construction methods: set packings, transversal designs, and direct constructions. In fact, the non-adaptive pooling problem is strongly connected to the problem of constructing superimposed codes [12], which was analyzed forty years ago to deal with the questions of representing rare document attributes in an information retrieval system and of assigning channels to relieve congestion in shared communications bands. The focus is different: each variable is seen as a code word and the goal is to maximize the number of code words n at fixed length v rather than the other way around; and these problems were noiseless, contrary to our own situation where error-correction is critical. Yet [12] had already suggested constructions of superimposed codes based on set packings, as well as constructions based on q -nary codes (which are in fact transversal designs) and on compositions of q -nary codes (which are not transversal anymore, and are more compact). Set packings, such as the designs presented in [2], can yield very efficient designs, but are mainly limited to $t \leq 2$ [1]. Transversal designs include the well-known grid (or row-and-column) design. This design is initially limited to identifying a single positive in the absence of noise, and is not very efficient, but it has been improved in two directions: hypercube designs [4] generalize it by considering higher dimension grids, and various methods [e.g. 7] have been proposed to build several "synergical" grids that work well together. Finally, some authors have proposed direct constructions of error-correcting pooling designs [13, 17].

Note that STD, although directly constructed, is in fact a transversal design. Furthermore, STD can be seen as a constructive definition of a q -nary code as proposed by [12], i.e. a concatenated code where the inner code is simply the unary code, and the outer code has some similarities with a Reed-Solomon code [19]. Yet although related, the methods are clearly different: for example, STD doesn't produce useful pools if q is a prime power; on the other hand, STD allows to build up to $q+1$ layers, whereas the Reed-Solomon based construction can only build up to $q-1$. Furthermore, STD produces efficient pools independently of the number of variables n , contrary to the Reed-Solomon approach where one is faced with the difficult problem

of choosing a good subset of code words except for some n values. The relationship between the two approaches requires further investigations.

Set packing designs. Regarding set packing designs, the main results taking into account error-correction are presented in [2]. The authors define designs that can correct any number of errors and identify up to $t=2$ positives, and prove that these designs are optimal when the construction is possible (it is not always possible if $t=2$). A real-world application is described in [3], where suboptimal variants of the previous designs are proposed for screening a clone map with $n=1530$ and $t=2$. The authors start off with an optimal design for $t=2$ and $E=0$, which can deal with 4368 variables, and select 1530 of these to serve as clones in their design. This was presumably done because the optimal designs cannot be built for arbitrary values of n : n must be the size of a certain Steiner system. Although the resulting designs are no longer optimal, they can retain high performance and obviously still satisfy the guarantee requirement. Additionally, this strategy reduces the sizes of pools, providing increased robustness (e.g., some information can still be obtained if, exceptionally, three objects are positive), and complying with the application-imposed pool size constraints. In the example, $n=1530$ and $t=2$, and the authors propose two designs: one with 65 pools of 118 clones each, and one with 54 pools of 142 clones. These numbers are very close to what would be recommended with STD: we could propose $\text{STD}(1530; 13; 5)$ which has 65 pools of 118 clones, or $\text{STD}(1530; 7; 7)$ with 49 pools of 218 clones. All of these designs guarantee the identification of 2 positives in the absence of noise. Furthermore, although noise-tolerance is not guaranteed in any of them, simulations we have performed suggest that substantial error-rates can be corrected in the STD designs, as is the case in the others. Therefore these designs and STD appear to achieve very similar performances, but STD is more flexible since it can be used for any number of positives, instead of being limited to $t \leq 2$.

Transversal designs. An interesting generalization of the grid design is described in [4]. The authors propose to array the variables in a D -dimensional cube, instead of the 2 dimensions used in the standard grid design. Furthermore, they advise that the length of the cube's side be chosen prime: let us denote it q . A pool is then obtained from each hyperplane, so that the D -dimensional cube yields D layers of q pools, each comprising up to n/q variables. To obtain more layers, the authors propose a criterion to construct "efficient transforming matrices" that produce additional cubes, where variables are as shuffled as possible; in fact, the purpose of their "efficiency" criterion is identical to the "co-occurrence of variables" property satisfied by STD (theorem 1, see 3.1). Seen like this, their system is clearly related to STD: D is $\Gamma+1$, and although the authors do not investigate their design's behavior under the guarantee requirement, corollaries 1 and 2 from sections 3.2 and 3.3 can in essence be applied. Furthermore, when the cube is "full", i.e. when $n=q^D$, their pools satisfy an analog of theorem 2 (i.e. they are "information-efficient" in some sense, see 3.4). Note that this cannot be the case when q is arbitrary; this may explain why the authors limit their options for q to the smallest primes larger than $n^{1/D}$, for each D value. However, each D -dimensional cube provides only D layers, and the proposed criterion for building additional cubes is not systematic, so that the total number of layers that can be built is unclear but seems much smaller than with STD. In addition, the authors don't take observational noise into account (they do talk of "false positives", but are really referring to what we call ambiguous variables). For these reasons, we cannot rigorously compare the designs under the guarantee requirement, but in general the fact that STD can build more layers is clearly favorable, since it allows dealing with a greater number of positives and/or errors at any chosen q value. In a numerical example concerning the screening of the CEPH YAC library, $n=72000$ and the authors argue that the optimal dimension and side length to use are $D=3$ and $q=43$, respectively. They then exhibit a set of transforming matrices that allows the construction of at most 3 additional cubes, yielding a total of 12 layers. By contrast, using the same values for D and q , STD can build up to 44 layers, which all satisfy the efficiency criterion. We believe that some of these extra layers could prove valuable, especially when allowing for experimental noise. In addition, smaller values for q can be used with STD (while still being information-efficient in the sense of theorem 2), although simulations would be necessary to choose the best value.

Two other transversal pooling designs, which generalize the grid design by providing additional 2-dimensional grids, are described in [7]. In essence, they are very similar to STD when $\Gamma=1$: writing $q=\sqrt{n}$, they allow the construction of up to $q+1$ layers of pools (where each layer contains q pools of size q) which satisfy the property that any pair of variables appears in at most one pool. Theorem 1 of section 3.1 shows that this property, known as the "unique colinearity" condition, is in fact verified by STD when $\Gamma=1$ (in accord with $q=\sqrt{n}$). We can observe that these designs, as well as STD when $\Gamma=1$, are maximal under this condition, since each pair of variables is in fact present in exactly one pool. Corollaries 1 and 2 of sections 3.2 and 3.3 can be applied, and show that they allow the identification of up to t positives while correcting E observation errors, provided that $t+2 \cdot E+1 \leq q+1$. The performance of the designs from [7] is therefore identical to that of STD when $\Gamma=1$. However, STD is superior to these designs in two respects. First, their

constructions are only possible if q is prime and $q \equiv 5 \pmod{6}$ (using the so-called RCF construction), or if q is prime and $q \equiv 3 \pmod{4}$ (using the "Union Jack Design"). By contrast, STD only requires that q is prime. Second, STD can be used with any compression power, rather than being limited to $\Gamma=1$. This flexibility is an advantage, because STD can be customized to suit more applications. Notably, when the fraction of positives is small, the Union Jack and RCF designs perform less well: the pools are too small, and observing that a pool is negative brings little information. By contrast, pools in STD can be very large (when choosing a small q), so that every observation is informative. To illustrate this point, let us consider the numerical example of [17] discussed below, where the fraction of positives is particularly low ($n=18,918,900$ and t is 2 or 9). The best usable design from [7] would be a Union Jack with $q=4363$, and would require a total of 13,089 pools for 2 positives - 77 times more than STD - and 43,630 pools to guarantee the identification of 9 positives - 32 times more than STD.

Direct constructions. In [13], the author proposes a direct construction allowing the detection of an arbitrary number of positives. Although this design is not very efficient under the guarantee requirement, the author shows in [14] that the pools designed for detecting 2 positives allow with high probability the detection of more positives. A numerical example, presented in [16], is the following. If $n=10^6$ and $t=5$, using 946 pools guarantees the identification of 2 positives and successfully identifies up to 5 positives with probability 97.1%. In comparison, under the guarantee requirement (i.e. with probability 100%), $\text{STD}(n; 11; 11)$ contains 121 pools and identifies 2 positives, and $\text{STD}(n; 23; 21)$, which comprises 483 pools, guarantees the identification of up to 5 positives.

Finally, another group [17] described two new classes of non-adaptive pooling designs, which allow the detection of any number of positives and the correction of half as many errors. Following the idea from [14], they also show that their designs for $t=2$ have high probabilities of being successful for more positives. In a numerical example, they consider the case $n=18,918,900$, and propose a design with 5460 pools which guarantees the identification of 2 positives, and can in addition identify up to 9 positives with 98.5% chance of success. By contrast, $\text{STD}(n; 13; 13)$ contains 169 pools and guarantees the identification of 2 positives, and the identification of 9 positives is guaranteed with the 1369 pools of $\text{STD}(n; 37; 37)$.

5. Conclusion

In this paper, we have presented a new pooling design: the "shuffled transversal design" (STD). We have proven that it constitutes an error-correcting solution to the pooling problem. This design is highly flexible: it can be tailored to deal efficiently with many experimental settings (i.e., numbers of variables, positives and errors). Finally, under a standard performance criterion, i.e. requiring that the correction of all errors and the identification of all variables' values be guaranteed mathematically, we have shown that STD compares favorably, in terms of numbers of pools, to the main previously described deterministic pooling designs.

This approach is being experimentally validated in collaboration with Marc Vidal's laboratory at the Dana Farber Cancer Institute, Boston. In a pilot project, pools have been generated with 940 AD-Y preys, using the $\text{STD}(940;13;13)$ design, and we are screening the 169 resulting pools against 50 different baits. This experiment will provide estimations for the technical noise levels of their high-throughput 2-hybrid protocol, in addition to producing valuable interaction data and yielding a real-world evaluation of the STD approach.

Although this work is motivated by protein interaction mapping, as we have been collaborating with Marc Vidal's group for several years, its scope is certainly not limited to high-throughput two-hybrid projects. Potential applications include a wide range of high-throughput PCR-based assays such as gene knockout projects [e.g. 20], drug screening projects [e.g. 11], and various proteomics studies [e.g. 15]. Furthermore, this general problem certainly has applications outside biology.

In practice, an important point is made in [14], where the author shows that his pooling design can be used to detect with high probability more positives than guaranteed. Simulations we have performed show that this observation is also true with STD: the gains can be increased substantially if one tolerates a small fraction of ambiguous variables that will need to be retested. However, these considerations are outside the scope of this paper, because they cannot be studied rigorously, by analytical means, but require simulations. Yet using such a strategy in practice with STD significantly improves the performance. For example, consider the case $n=10000$ and $t=5$, and suppose that the assay has an error-rate of 1%. To guarantee the identification of all variables' values, one must use 483 pools (with $q=23$ and $k=21$). However, if one tolerates up to 10 ambiguous variables, even when overestimating the error-rate to 2% for safety's sake, 143 pools prove amply sufficient. It is clear that this "ambiguity-tolerant" approach should be preferred in

practical applications. This approach and the corresponding computer program, which performs simulations to select the STD parameter values best suited for a given application and includes original efficient algorithms for preparing the pools and decoding the outcomes, will be discussed in another paper.

Another interesting track will be to study the efficiency of pooling designs from the point of view of Shannon's Information Theory. We are planning to investigate STD's behavior in this context. In particular, theorem 2 from section 3.4 could prove useful for this.

Finally, the connection between STD and constructions based on superimposed codes, e.g. q-nary Reed-Solomon codes [12], warrants further studies.

Acknowledgements

I would like to thank Laurent Trilling, Danielle Thierry-Mieg, Jean Thierry-Mieg and Jean-Louis Roch for stimulating discussions and for carefully reading the manuscript. This work was funded by a BQR grant from the Institut National Polytechnique de Grenoble (INPG).

References

- [1] Balding, D., Bruno, W., Knill, E., & Torney, D. (1996) A comparative survey of non-adaptive pooling designs. In *Genetic mapping and DNA sequencing*. Springer, New York, pp. 133-154.
- [2] Balding, D., & Torney, D. (1996) Optimal pooling designs with error correction. *J. Comb. Theory Ser A*, **74**, 131-140.
- [3] Balding, D., & Torney, D. (1997) The design of pooling experiments for screening a clone map. *Fungal genet. biol.* **21**, 302-307.
- [4] Barillot, E., Lacroix, B., & Cohen, D. (1991) Theoretical analysis of library screening using a N-dimensional pooling strategy. *Nucl. Acids Res.* **19**, 6241-6247.
- [5] Bruno, W., Knill, E., Balding, D., Bruce, D., *et al* (1995) Efficient pooling designs for library screening. *Genomics* **26**, 21-30.
- [6] Cai, W., Chen, R., Gibbs, R., & Bradley, A. (2001) A clone-array pooled shotgun strategy for sequencing large genomes. *Genome Res.* **11(10)**, 1619-1623.
- [7] Chateaneuf, M., Colbourn, C., Kreher, D., Lamken, E., *et al* (1999) Pooling, lattice square, and union jack designs. *Ann. Comb.* **3**, 27-35.
- [8] Davy, A., Bello, P., Thierry-Mieg, N., Vaglio, *et al* (2001) A protein-protein interaction map of the *Caenorhabditis elegans* 26S proteasome. *EMBO Rep.* **2(9)**, 821-828.
- [9] Du, D.-Z., & Hwang, F. (2000) *Combinatorial Group Testing and Its Applications*, 2nd edn., World Scientific, Singapore.
- [10] Evans, G., & Lewis, K. (1989) Physical mapping of complex genomes by cosmid multiplex analysis. *Proc. Natl. Acad. Sci. USA* **86(13)**, 5030-5034.
- [11] Hann, M., Hudson, B., Lewell, X., Lifely, R., *et al* (1999) Strategic pooling of compounds for high-throughput screening. *J. Chem. Inf. Comput. Sci.* **39**, 897-902.
- [12] Kautz, W., & Singleton, H. (1964) Nonrandom binary superimposed codes. *IEEE Trans. Inform. Theory* **10**, 363-377.
- [13] Macula, A. (1996) A simple construction of d-disjunct matrices with certain constant weights. *Discrete Math.* **162**, no. 1-3, 311-312.
- [14] Macula, A. (1999) Probabilistic nonadaptive group testing in the presence of errors and dna library screening. *Ann. Comb.* **3**, 61-69.
- [15] Martzen, M., McCraith, S., Spinelli, S., Torres, F., *et al* (1999) A biochemical genomics approach for identifying genes by the activity of their products. *Science* **286**, 1153-1155.
- [16] Ngo, H., & Du, D.-Z. (2000) A survey on combinatorial group testing algorithms with applications to DNA library screening. DIMACS Ser. Discrete Math. Theoret. Comput. Sci., vol. 55, 171-182.
- [17] Ngo, H., & Du, D.-Z. (2002) New constructions of non-adaptive and error-tolerance pooling designs. *Discrete Math.* **243**, no. 1-3, 161-170.
- [18] Reboul, J., Vaglio, P., Rual, J.-F., Lamesch, P., *et al* (2003) C. elegans ORFeome version 1.1: experimental verification of the genome annotation and resource for proteome-scale protein expression. *Nat. Genet.* **34(1)**, 35-41.
- [19] Reed, I., & Solomon, G. (1960) Polynomial codes over certain finite fields. *J. Soc. Ind. Appl. Math.*, vol. 8, 300-304.
- [20] The C.elegans Sequencing Consortium (1998) Genome sequence of the nematode C. elegans: a platform for investigating biology. *Science* **282**, 2012-2018.
- [21] Walhout, A., Sordella, R., Lu, X., Hartley, J., *et al* (2000) Protein interaction mapping in C. elegans using proteins involved in vulval development. *Science* **287**, 116-122.
- [22] Zwaal, R., Broeks, A., van Meurs, J., Groenen, J., *et al* (1993) Target-selected gene inactivation in *Caenorhabditis elegans* by using a frozen transposon insertion mutant bank. *Proc. Natl. Acad. Sci. USA* **90(16)**, 7431-7435