Constrained Group Testing to Predict Binding Response of Candidate Compounds

Paul Quint*  Stephen Scott†  N. V. Vinodchandran‡  Brad Worley§

Abstract
We study the problem of identifying reactive compound(s) in a solution as efficiently as possible, with the goal of minimizing the number of chemical tests required to make exact identification. The area of group testing is appropriate for this problem, except that most group testing approaches assume that arbitrary tests can be performed, which is not the case in our application. To address this, we introduce a new model called mask-based constrained group testing, develop a randomized algorithm for it, and prove that under the right conditions, the algorithm is guaranteed w.h.p. to efficiently identify the active compounds of a solution with a small number of tests. We also show that our algorithm performs very well empirically on synthetic and real data.

1 Introduction
We study the problem of identifying reactive compound(s) in a solution as efficiently as possible. Consider a solution $C$ of compounds $\{c_1, c_2, \ldots, c_n\}$. There is an unknown $c_i$ that is reactive in a particular way to an organism (e.g., causing ACC2 inhibition in an organism, a process related to obesity in rats [12]). We want to isolate the compound $c_i$. We can feed a fractionized version of the solution containing a subset $C$ of the compounds, to the organism. If $c_i \in C$ then this fractionized solution will have the same effect as the original solution. Since in many cases the testing (e.g., animal testing or NMR spectroscopy [11]) can be very expensive in cost and effort, the goal is to perform a minimal number of fractionation processes and subsequent experiments and still succeed in identifying the reactive compound $c_i$.

Formally, let $C = \{c_1, c_2, \ldots, c_n\}$ be the set of compounds, and let $R \subset C$ be an unknown set of $k$ reactive compounds in $C$. Let $T$ be an experiment that reacts in a specific, identifiable way when some $c \in R$ is present. That is, for every $C \subset C$, $T(C) = 1$ if and only if $C \cap R$ is non-empty. We want to design a collection $C_1, C_2, \ldots, C_m$ of $m$ subsets of compounds so that using the $m$ experimental results $T(C_i)$, we can identify $R$. We are interested in designing a collection of tests in advance (non-adaptively) that will work for any $R$ of size $k$ a priori before testing begins. Results from the field of group testing [6, 7] can be applied here to minimize the total number of tests specified in advance needed to isolate reactive compounds.

Our Motivation and a New Model. In a typical group testing scenario, it is assumed that an arbitrary subset of $C$ can be tested via $T$. We call this unconstrained group testing. In this case it is known that $O(k^2 \log n)$ tests are sufficient to identify any set of $k$ reactive compounds [7]. Our present work is motivated by the practical limitations of such an assumption that any arbitrary subset of $C$ can be isolated to test. For example, in the chemical process called fractionation to separate a solution into sub-solutions, the compounds of the original solution are partitioned based on one or more physical properties, such as molecular weight, solubility, or polarity. A natural way to model fractionation is to first consider the molecules in increasing order of a property $p$ and include all those compounds in a test that are below a certain threshold value for $p$. While we can use several properties (and possibly multiple thresholds per property) to create several subsets of compounds to test, it is not practically possible to isolate an arbitrary subset of compounds from the solution. Thus, unlike in many other group testing situations, in the compound identification problem we are constrained by the limitations of the fractionation processes in what sub-solutions we can separate and test.

Mask-based Constrained Group Testing Model. We introduce a new group testing model called the mask-based constrained group testing model. In this model,

---

*Dept. of Computer Science and Engineering, University of Nebraska-Lincoln, email: quint@cse.unl.edu
†Dept. of Computer Science and Engineering, University of Nebraska-Lincoln, email: ssco@unl.edu
‡Dept. of Computer Science and Engineering, University of Nebraska-Lincoln, email: vinod@cse.unl.edu
§Dept. of Chemistry, University of Nebraska-Lincoln, email: bradley.worley@huskers.unl.edu

1 Designing a collection of adaptive tests, so that the collection can change depending on each $R$, is also interesting. However, we will not deal with that problem in this paper.
we assume that one is given a set \( \mathcal{M} \) of masks where each mask represents a fractionation process. For a collection \( \mathcal{C} = \{c_1, c_2, \ldots, c_n\} \) of \( n \) compounds, a mask is an \( n \)-bit vector. A “1” in position \( i \) of this mask indicates that compound \( c_i \) will be retained as part of the solution’s sub-solution after fractionation, while a “0” indicates that the compound will be eliminated after fractionation. From this set of masks we can generate a collection of binary vectors corresponding to subsets of \( \mathcal{C} \) by intersection and union steps. The intersection step is a bitwise AND between the current indicator vector (describing the compounds present in a solution) and a mask. The intersection step models multiple fractionations in series, allowing a fractionized sub-solution to be further fractionized by a different process. This is because if one wanted to apply the fractionation process \( P_i \) followed by process \( P_j \) on the original solution, the final result would be represented by the bitwise AND of the two processes’ mask vectors.

A union step is a bitwise OR of two or more indicator vectors. This corresponds to allowing fractionized sub-solutions to be mixed to form new solutions, which can then be tested. As a practical point, since each fractionation and mixing process degrades the solution, we must limit ourselves to fairly small numbers of intersection steps and union steps. We specify this limitation using the two parameters \( s_\cap \) and \( s_\cup \). Only those vectors realizable by application of a series of such steps are eligible sub-solutions for testing.

**Our Results.** We first present a randomized algorithm to generate a test matrix (where each row corresponds to a sub-solution test to be performed) and a recovery algorithm to identify the target compounds based on the test results. We give a theoretical analysis of the algorithms and prove that under certain randomness assumptions on the masks, the recovery algorithm is guaranteed with high probability to identify the active compounds of a solution with a number of tests that grows with \( O(k^2 \log n) \), where \( n \) is the number of compounds in the solution and \( k \) is the number of active compounds. We then show that our algorithm performs very well empirically on both synthetic and real data sets.

The rest of this paper is organized as follows. In Section 2 we give necessary background and summarize related work. In Section 3 we describe our algorithm for mask-based constrained group testing, and then analyze it in Section 4, proving guaranteed performance bounds. We then empirically evaluate our approach in Section 5 and conclude in Section 6.

### 2 Background and Related Work

The field of group testing originated with Dorfman [6] in order to efficiently screen World War II draftees for syphilis by combining multiple blood samples into a single mixture and testing the entire batch for the presence of syphilitic antigen. The goal is to have a priori (in contrast to adaptively) design a set of mixtures to each be tested that will allow for all reactive elements to be identified. This field has been studied extensively with many applications, including sequence analysis [15], network security [17], and manufacturing defect testing [16]. More detail on approaches and applications can be found in Du and Hwang [7]. In addition to the many applications, variations of group testing have been studied, including algorithms to tolerate noisy test results [3].

There has been also work on group testing in which there are constraints on the combinations that can be tested. Specifically, the constraints are imposed by a specified graph structure [4,9], and algorithms for these types of problems are rooted in random walks on the graphs. In contrast, our model describes constraints in terms of limited intersections and unions of bit vectors that represent subsets allowable for testing.

Group testing in general and our results in particular are closely related to the problem of compressed sensing (CS) (see Donoho [5] and references therein). In compressed sensing one takes measurements of a signal and in doing so infers the signal, exactly or approximately. Formally, one specifies a set of measurements to take in the form of a real matrix \( A \in \mathbb{R}^{m \times n} \), observes the results of these measurements in a vector \( b \in \mathbb{R}^m \), and from these measurements infers the signal \( x^* \in \mathbb{R}^n \) such that \( Ax^* = b \). The main design goal is to make as few measurements as possible, i.e., limit \( m \), the number of rows of \( A \). In this regard it is known that if the signal \( x^* \) is \( k \)-sparse (all but \( k \) entries are 0), then we can design a measurement matrix with a very small number

**Figure 1:** Illustration of intersection and union steps to generate a subset of compounds from masks \( M1, M2 \) and \( M3 \).
of rows \((m = O(k \log n))\) if \(x^*\) is \(k\)-sparse). Variations of this problem include inferring an approximation of the signal \(x^*\) as well as only inferring the support (locations of non-zero entries) of \(x^*\) rather than the actual values based on limited information on the measured vector \(b\). This latter variation, known as support recovery with 1-bit compressed sensing is of particular interest to our work. Here the goal is to design a matrix \(A\) so that we can recover support of \(x^*\) from the sign vector of \(Ax^*\) [2]. If we limit \(A\) and \(x^*\) to be non-negative, this problem is exactly the group testing problem. Thus, a natural extension of the constrained group testing problem is the constrained compressed sensing problem where we impose restrictions on the test matrix \(A\).

3 Generation and Recovery Algorithms

Algorithm 1 describes the building of test matrix \(A\) for the constrained group testing problem. It takes as input the set of masks \(\mathcal{M}\) that are allowed to be combined via bitwise AND and OR to create rows of the measurement matrix \(A\). Its other two parameters define how many AND and OR operations are allowed per row of \(A\). Specifically, \(s_{\cap}\) is the largest number of fractionations (mask intersections) allowed to get a sub-solution (mask intersection result) to be combined, and \(s_{\cup}\) is the largest number of sub-solutions (mask intersection results) that may be combined (ORed) to get a row of \(A\).

Algorithm 1 repeats a simple sampling process. For each row \(r\) from 1 to \(m\), it decides how many sub-solutions it will combine for that row (call it \(u\)). Then for each sub-solution \(z\) it plans to create, it decides how many intersections of masks from \(\mathcal{M}\) to perform (i.e., how many sequential fractionations to perform). Call that number \(c\). It then chooses \(c\) masks to intersect and then combines via union that intersection result with the previous one(s), and repeats the process. Once the row \(r\) is defined, it is added to \(A\) and the next row is computed independently of the previous ones.

Once \(A\) is determined, in practice those \(m\) tests would each be performed and the \(m\) results presented in result vector \(b\). We then recover the support of \(x^*\) and thereby identify the subset of compounds that are active via the pseudocode in Algorithm 2 (adapted from Gopi et al. [8]). The algorithm simply determines a bit vector \(x\) such that \(Ax = b\). For column \(i\) of \(A\) (referred to as vector \(a\)), if its \(j\)th bit is 1 and the \(j\)th bit of \(b\) is positive, then, based on the properties of \(A\) (see Section 4), we know with high probability that \(x_i^*\) is non-zero and part of the support of \(x^*\).

4 Theoretical Analysis

We next set up a theoretical framework and show that if the masks \(\mathcal{M}\) are generated i.i.d. u.a.r., then the measurement matrix \(A\) is a \(k\)-union-free matrix and the support recovery algorithm provably works. We start with the definition of a \(k\)-union-free matrix (in short, \(k\)-UFM). In the literature, this notion is defined in terms of a family of sets, but we use the Boolean matrix representation.

**Definition 1.** Let \(A\) be an \(m \times n\) Boolean matrix and \(k\) be an integer \(1 \leq k \leq n\). Let \(c_1, \ldots, c_n\) be the columns of \(A\). Then \(A\) is said to be a \(k\)-union-free matrix iff: for any \((k + 1)\) columns \(c_i, c_i, \ldots, c_i\), there is a \(j:\) \(1 \leq j \leq m\), so that the \(j\)th bit of \(c_i\) is 1 but \(j\)th bit of the bitwise-OR of \(c, \ldots, c\) is 0.

It is known that if the test matrix \(A\) is a \(k\)-UFM, then the recovery algorithm recovers support of any \(k\)-sparse vector (a vector with at most \(k\) non-zero entries)
from the support of the measurement vector. We give a proof in the end of this section.

First we define notation and establish bounds on the probability of getting a 1 in any entry of the columns of $A$ generated by Algorithm 1, starting with an arbitrary set of $\ell$ masks. Then we show that if the masks are drawn according to certain natural random processes, with very high probability $A$ will be a $k$-UFM.

Let $M$ be a $\ell \times n$ mask matrix built from stacking the masks of $M$ on top of each other. For column $i$, let $f_i$ denote the fraction of ones in column $i$ of $M$. Let $f_{\text{min}} = \min_{1 \leq i \leq n} f_i$ and similarly defined $f_{\text{max}}$ be lower and upper bounds on $f_i$. We can now analyze the makeup of each column of the $m \times n$ test matrix $A$.

**Lemma 4.1.** Let $A$ be an $m \times n$ test matrix generated by Algorithm 1. Then for a column $i$ of $A$, each of the $m$ entries of this column is the result of an i.i.d. Bernoulli trial with probability $p_{\gamma,i}$ given by the following expression:

$$
(4.1) \quad p_{\gamma,i} = p(f_i, s_\gamma, s_u) = 1 - \frac{1}{s_{\gamma_i}} \sum_{u=1}^{s_u} (1 - p_{\gamma,i})^u,
$$

where $p_{\gamma,i} = p(f_i, \ell, s_\gamma)$ is given by the expression

$$
(4.2) \quad p_{\gamma,i} = p(f_i, s_\gamma) = \frac{1}{s_\gamma} \sum_{c=1}^{s_\gamma} f_i^c.
$$

**Proof.** Since each bit of the $i$th column of $A$ is generated with independent intersection-union operations of the $i$th column of mask matrix, each of the $m$ bits are independent. To analyze the probability that the $j$th bit of this column is 1, first consider the quantity $p_{\gamma,j} = p(f_j, s_\gamma) =$ the probability that Algorithm 1’s intersection operation yields a one in this position. Recall that the intersection process first chooses a value $c$ u.a.r. from $\{1, \ldots, s_\gamma\}$, draws $c$ masks at random (here we are analyzing with replacement), and intersects them. Thus, $p_{\gamma,j} = (1/s_\gamma) \sum_{c=1}^{s_\gamma} f_i^c$.

In Algorithm 1, the intersection process is repeated $u$ times (with $u$ chosen u.a.r. from $\{1, \ldots, s_u\}$), yielding $u$ bit vectors, which are then combined with a bitwise OR (union) operation. The probability that each of the $u$ bit vectors has a zero in position $i$ is $(1 - p_{\gamma,i})^u$, so the probability that a union of $u$ bit vectors (with $u$ chosen u.a.r. from $\{1, \ldots, s_\gamma\}$) has a one in position $i$ is

$$
p_{\gamma,i} = p(f_i, s_\gamma, s_u) = 1 - \frac{1}{s_{\gamma,i}} \sum_{u=1}^{s_u} (1 - p_{\gamma,i})^u.
$$

Square here.

For any fixed $(s_\gamma, s_u)$ pair, $p_{\gamma,i}$ is a monotonic function of $f_i$. Hence, we get a maximum and minimum value for $p_{\gamma,i}$ from $f_{\text{max}}$ and $f_{\text{min}}$. Let $p_{\text{max}} =\ldots$

\[\begin{array}{c}
\text{Figure 2: } p_{\gamma,i} (y-axis) \text{ as a function of } f_{i} (x-axis) \text{ for } (s_\gamma, s_u) \text{ values (5,1), (5,3), (2,2), (3,5) and (1,5)} \end{array}\]

$p_{\gamma,i}(f_{\text{max}}, s_\gamma, s_u)$ and $p_{\text{min}} = p_{\gamma,i}(f_{\text{min}}, s_\gamma, s_u)$. Then for all $i \leq n$, $p_{\text{min}} \leq p_{\gamma,i} \leq p_{\text{max}}$. Figure 2 illustrates how $p_{\gamma,i}$ varies as a function of $f_i$ for various values of $(s_\gamma, s_u)$. Later, in Theorem 4.1, we show that it is desirable to have $p_{\gamma,i} \approx 1/k$. This can be achieved using Figure 2, since given a mask matrix $M$, one can choose $(s_\gamma, s_u)$ that yield $p_{\gamma,i} \approx 1/k$ for $M$’s $f_{\text{min}}$ and $f_{\text{max}}$.

From an arbitrary set of masks, it is not possible to prove any guarantee on the quality of $A$ or on the correctness of the recovery process. For example, if two columns of the mask matrix are identical, then any test matrix generated from this mask matrix will have their corresponding columns identical. Hence, information theoretically, it is impossible to distinguish the variables corresponding to these two columns.

However, if the mask matrix is generated via some random process, then there is a possibility of proving an absolute guarantee. In particular, we show that if each bit of the mask matrix is the result of independent and identically distributed Bernoulli trials, then with very high probability the matrix $A$ generated is a $k$-UFM with a small number of rows.

**Theorem 4.1.** For any $n$, $k \leq n$, $0 < \delta, \epsilon_1, p_M < 1$, $s_u$, $s_\gamma$, and $0 < \epsilon_2 < 1/k$, a random $m \times n$ Boolean matrix $A$ generated by first picking a random $\ell \times n$ matrix $M$ with probability of 1 in any entry being $p_M$, and then using the process described in Algorithm 1, is a $k$-UFM with probability $\geq (1-\delta)$, provided that

- $m \geq \max\{\ln n + \ln \frac{3}{\delta}\}/\epsilon_2^2 p_{\text{max}}, (2k \ln n + \ldots$

\[\begin{array}{c}
\text{Figure 2: } p_{\gamma,i} (y-axis) \text{ as a function of } f_{i} (x-axis) \text{ for } (s_\gamma, s_u) \text{ values (5,1), (5,3), (2,2), (3,5) and (1,5)} \end{array}\]
\[
\ln \left(\frac{3}{\delta}\right)/(p_{\min}(1 - k(p_{\max} + \epsilon_2)))
\]
\[
\ell \geq (\ln n + \ln \frac{3}{\delta})/p_M \epsilon_1^2
\]
\[
p_{\max} < \frac{1}{k} - \epsilon_2
\]
\[
\text{where } p_{\max} = p_{\cup,i}(p_M + \epsilon_1, s_{\cap}, s_{\cup}) \text{ and } p_{\min} = p_{\cap,i}(p_M - \epsilon_1, s_{\cup}, s_{\cap}).
\]

Thus, if \(\delta, \epsilon_1, \epsilon_2\) are chosen to be small constants < 1 and \(p_M\) is chosen such that \(p_{\min}, p_{\max} = \Theta(1/k)\), then with very high probability Algorithm 1 generates an \(m \times n\) test matrix \(A\) which is a \(k\)-UFM from a random \(\ell \times n\) mask matrix where \(\ell = \Omega(\log n/p_M)\) and \(m = \Omega(k^2 \log n)\). It is known that any \((m \times n)\) \(k\)-UFM should have \(m = \Omega(k^2 \log n/\log k)\).

**Proof.** (of Theorem 4.1.) Consider the process of generating a random \(A\). First, a random \(\ell \times n\) matrix \(M\) is generated with each bit of the matrix being 1 with probability \(p_M\). From this the matrix, \(A\) is generated as described in Algorithm 1. The expected number of ones in each column of \(A\) is \(\ell p_M\). First we bound the probability that the fraction of ones in each column of \(A\) is outside the range \([p_M - \epsilon_1, p_M + \epsilon_1]\] for a small constant \(\epsilon_1\). Fix a column of \(A\) and let \(E_1\) be the event that the fraction of ones in this column is outside this range. Since the entries of this column are independent, using Chernoff’s bound,

\[
\Pr(E_1) \leq 2 \exp\left(-\epsilon_1^2 \ell p_M/3\right).
\]

Applying a union bound over all \(n\) columns, we get that for \(\ell \geq 3(\ln n + \ln \frac{3}{\delta})/p_M \epsilon_1^2\), with probability \(\geq 1 - \delta/3\), each column of \(M\) has a fraction of ones between \(f_{\min}\) and \(f_{\max}\).

For a set of parameters \(f_{\min}, f_{\max}, s_{\cup}, s_{\cap}, \ell\), we get \(p_{\min}\) and \(p_{\max}\), the minimum and maximum probability that any entry of any column of \(A\) is a 1 (as formulated earlier). In the following we will work with \(p_{\min}\) and \(p_{\max}\) and bound the probability that a randomly generated \(A\) is a \(k\)-UFM.

Now we show that with high probability, every column of \(A\) has not more than \(m p_{\max}\) ones. Fix any column \(c_i\) of \(A\) and let \(X_i\) be the number of ones in \(c_i\). Then, for \(0 < \epsilon_2 < 1\), using Chernoff’s bound,

\[
\Pr(X_i > (1 + \epsilon_2)m p_{\max}) \leq \exp\left(-\epsilon_2^2 m p_{\max}/3\right).
\]

Using a union bound we get that if \(m \geq (\ln n + \ln \frac{3}{\delta})/\epsilon_2^2 p_{\max}\), the probability that there is some column that has more than \(m(p_{\max} + \epsilon_2)\) ones is at most \(\delta/3\).

Finally, we bound the probability that \(A\) is not a \(k\)-UFM given that the probabilities are within the range \([p_{\min}, p_{\max}]\) and each column of \(A\) has at most \(m(p_{\max} + \epsilon_2)\) ones. Consider any fixed \((k + 1)\) columns of \(A\): \(c_0, c_1, \ldots, c_k\). Let \(E_2\) be the bad event that \(c_0\) is contained in the bitwise-OR of \(c_1, c_2, \ldots, c_k\) (that is, \(c_0\) does not have a 1 in any position where the bitwise-OR has a 0, violating the \(k\)-UFM condition). Since the mask bits are generated independently,

\[
\Pr(E_2) \leq (1 - p_{\min})^{m(1 - k(p_{\max} + \epsilon_2))}.
\]

There are \(n^{(n-1)}\) such bad events. Thus, using a union bound, the probability that a matrix \(A\) is not a \(k\)-UF

\[
\Pr(E_2) \leq n \left(\frac{n - 1}{k}\right)^{m(1 - k(p_{\max} + \epsilon_2))} \leq \exp(2k \ln n - p_{\min} m(1 - k(p_{\max} + \epsilon_2))) \leq \exp(2k \ln n - p_{\min} m(1 - k(p_{\max} + \epsilon_2))).
\]

For \(m \geq (2k \ln n + \ln(\frac{3}{\delta}))/p_{\min}(1 - k(p_{\max} + \epsilon_2))\), the above probability is less than \(\delta/3\).

Therefore, if

\[
m \geq \max\{\left(\ln n + \ln(3/\delta)/\epsilon_2^2 p_{\max}\right), (2k \ln n + \ln(3/\delta))/p_{\min}(1 - k(p_{\max} + \epsilon_2))\}\]

and

\[
\ell \geq (\ln n + \ln(3/\delta)) 3p_M/\epsilon_1^2,
\]

then \(A\) is a \(k\)-UFM with probability \(\geq 1 - \delta\).

We now give a proof that if \(A\) is a \(k\)-UFM, then the recovery algorithm (Algorithm 2) recovers the support of \(k\)-sparse target vectors.

**Theorem 4.2.** If \(A\) is a \(k\)-UFM, then Algorithm 2 recovers the support of any \(k\)-sparse vector from the support vector \(b\) of \(Ax\).

**Proof.** Let \(x\) be an arbitrary \(k\)-sparse vector and \(b\) be the support vector of \(Ax\). For an \(1 \leq i \leq n\), let \(x_i\) be nonzero. Then since \(b\) is the bitwise-OR of all those (at most \(k\)) columns of \(A\) where there is a nonzero entry in \(x\), \(c_i\) is contained in \(b\) (i.e., if the \(j\)th bit of \(c_i = 1\) then the \(j\)th bit of \(b = 1\)), then \(x_i\) will be set 1 by the algorithm. Now consider an \(i\) so that \(x_i = 0\). Then since \(A\) is a \(k\)-UFM, there will be an entry in \(x_i\) which is 1, but the corresponding entry in \(b = 0\). Hence for this \(i\), the flag will be set to 0 and correspondingly \(x_i\) will be set to 0.

**5 Experimental Results**

To empirically evaluate our method, we first generated synthetic data using random masks \(M\) (via two different processes) and target vectors \(x^*\). Further, we evaluated our algorithm on real chemistry data. We computed how many rows the measurement matrix \(A\) would need to perfectly recover the target vector and also plotted error rate versus number of rows.
5.1 Synthetic Dataset

5.1.1 Bernoulli-Based Masks In our synthetic experiments, we fixed \( n = 10^4 \) and varied \( k, \ell, s_\cap, s_\cup, \) and \( p_M \). For each value of \( \ell \in \{100, 200\} \) and each value of \( p_M \in \{0.2, 0.5\} \), we generated ten sets of \( \ell \) masks each, each with probability \( p_M \) of a one in each entry (i.e., each bit in each mask was the result of a Bernoulli trial with probability \( p_M \)). For each value of \( k \in \{4, 10\} \), we created ten target vectors \( \mathbf{x}^* \), with \( k \) ones at positions chosen u.a.r. and the rest zeros. Finally, we varied the pairs \( (s_\cap, s_\cup) \in \{(2, 2), (5, 1), (5, 2), (5, 3), (5, 4), (3, 5)\} \).

We specified inputs \( s_\cap, s_\cup, \) the masks, and some number \( m \) of rows for \( A \). After Algorithm 1 built \( A \), we computed \( \mathbf{b} = A\mathbf{x}^* \) and ran Algorithm 2 on \( A \) and \( \mathbf{b} \) to attempt to recover \( \mathbf{x}^* \). If the recovered vector \( \mathbf{x} = \mathbf{x}^* \), we call that trial a success, otherwise a failure. For each combination of \( k, \ell, s_\cap, s_\cup, \) and \( p_M \), we computed the minimum value of \( m \) such that all ten sets of masks would perfectly recover all ten target vectors, i.e., we computed the minimum \( m \) (called \( m^* \)) that yielded 100 successes out of 100 mask set-target pairs.

Table 1 shows \( m^* \) for the unconstrained case when the rows of test matrix \( A \) is a randomly generated \( n \)-bit vector as given in [8]. Results for our constrained algorithm are presented in Table 2 for masks generated with \( p_M = 0.5 \) and in Table 3 for \( p_M = 0.2 \).

Table 1: Value of \( m^* \) to recover support of random \( \mathbf{x}^* \) using unconstrained random rows.

<table>
<thead>
<tr>
<th>( n )</th>
<th>( k )</th>
<th>( m^* )</th>
<th>( k )</th>
<th>( m^* )</th>
</tr>
</thead>
<tbody>
<tr>
<td>( 10^4 )</td>
<td>4</td>
<td>293</td>
<td>10</td>
<td>385</td>
</tr>
</tbody>
</table>

We first note that, even when constrained to 100 masks with \( p_M = 0.5 \), there exist values of \( s_\cup \) and \( s_\cap \) such that the number of rows sufficient to fully recover all ten \( \mathbf{x}^* \) vectors was just greater than that of the unconstrained algorithm for \( k = 4 \), and about five times as many for \( k = 10 \). When \( \ell = 200 \), these ratios got smaller to the point where some values were very competitive. Similar patterns were observed for \( p_M = 0.2 \), though the ratios are larger than for \( p_M = 0.5 \), and not as pronounced when \( \ell = 200 \). We conclude that for random target vectors and random masks, our constrained algorithm was competitive with one that had the ability to use arbitrary rows in \( A \).

Contrasting \( k = 4 \) to \( k = 10 \), we see that the number of rows needed to succeed grows, which is unsurprising given the dependence on \( k \) of our theoretical analysis. In contrast, increasing \( \ell \) improves performance a great deal. This is because increasing \( \ell \) decreases the variance of the \( f_\ell \) values of the mask matrices, which gives us less variance in the values of \( p_{\ell,i} \) and thus more stability in the entries of the \( A \).

Table 2: Value of \( m^* \) to recover support of random \( \mathbf{x}^* \) for \( n = 10^4 \) and \( p_M = 0.5 \).

<table>
<thead>
<tr>
<th>( k )</th>
<th>( \ell )</th>
<th>( (s_\cap, s_\cup) )</th>
<th>( m^* )</th>
<th>( k )</th>
<th>( \ell )</th>
<th>( (s_\cap, s_\cup) )</th>
<th>( m^* )</th>
</tr>
</thead>
<tbody>
<tr>
<td>100</td>
<td>4</td>
<td>(2, 2)</td>
<td>460</td>
<td>100</td>
<td>4</td>
<td>(2, 2)</td>
<td>4915</td>
</tr>
<tr>
<td></td>
<td></td>
<td>(5, 1)</td>
<td>311</td>
<td></td>
<td></td>
<td>(5, 1)</td>
<td>1229</td>
</tr>
<tr>
<td></td>
<td></td>
<td>(5, 2)</td>
<td>406</td>
<td></td>
<td></td>
<td>(5, 2)</td>
<td>1638</td>
</tr>
<tr>
<td></td>
<td></td>
<td>(5, 3)</td>
<td>321</td>
<td></td>
<td></td>
<td>(5, 3)</td>
<td>1681</td>
</tr>
<tr>
<td></td>
<td></td>
<td>(5, 4)</td>
<td>303</td>
<td></td>
<td></td>
<td>(5, 4)</td>
<td>2013</td>
</tr>
<tr>
<td></td>
<td></td>
<td>(3, 5)</td>
<td>751</td>
<td></td>
<td></td>
<td>(3, 5)</td>
<td>2015</td>
</tr>
<tr>
<td>200</td>
<td>10</td>
<td>(2, 2)</td>
<td>368</td>
<td>200</td>
<td>10</td>
<td>(2, 2)</td>
<td>5315</td>
</tr>
<tr>
<td></td>
<td></td>
<td>(5, 1)</td>
<td>247</td>
<td></td>
<td></td>
<td>(5, 1)</td>
<td>945</td>
</tr>
<tr>
<td></td>
<td></td>
<td>(5, 2)</td>
<td>301</td>
<td></td>
<td></td>
<td>(5, 2)</td>
<td>1254</td>
</tr>
<tr>
<td></td>
<td></td>
<td>(5, 3)</td>
<td>305</td>
<td></td>
<td></td>
<td>(5, 3)</td>
<td>1613</td>
</tr>
<tr>
<td></td>
<td></td>
<td>(5, 4)</td>
<td>358</td>
<td></td>
<td></td>
<td>(5, 4)</td>
<td>1843</td>
</tr>
<tr>
<td></td>
<td></td>
<td>(3, 5)</td>
<td>451</td>
<td></td>
<td></td>
<td>(3, 5)</td>
<td>3842</td>
</tr>
</tbody>
</table>

Table 3: Value of \( m^* \) to recover support of random \( \mathbf{x}^* \) for \( n = 10^4 \) and \( p_M = 0.2 \).

<table>
<thead>
<tr>
<th>( k )</th>
<th>( \ell )</th>
<th>( (s_\cap, s_\cup) )</th>
<th>( m^* )</th>
<th>( k )</th>
<th>( \ell )</th>
<th>( (s_\cap, s_\cup) )</th>
<th>( m^* )</th>
</tr>
</thead>
<tbody>
<tr>
<td>100</td>
<td>4</td>
<td>(2, 2)</td>
<td>3444</td>
<td>100</td>
<td>4</td>
<td>(2, 2)</td>
<td>9661</td>
</tr>
<tr>
<td></td>
<td></td>
<td>(5, 1)</td>
<td>9114</td>
<td></td>
<td></td>
<td>(5, 1)</td>
<td>8063</td>
</tr>
<tr>
<td></td>
<td></td>
<td>(5, 2)</td>
<td>3772</td>
<td></td>
<td></td>
<td>(5, 2)</td>
<td>13543</td>
</tr>
<tr>
<td></td>
<td></td>
<td>(5, 3)</td>
<td>3634</td>
<td></td>
<td></td>
<td>(5, 3)</td>
<td>10700</td>
</tr>
<tr>
<td></td>
<td></td>
<td>(5, 4)</td>
<td>2869</td>
<td></td>
<td></td>
<td>(5, 4)</td>
<td>13353</td>
</tr>
<tr>
<td></td>
<td></td>
<td>(3, 5)</td>
<td>4128</td>
<td></td>
<td></td>
<td>(3, 5)</td>
<td>11701</td>
</tr>
<tr>
<td>200</td>
<td>10</td>
<td>(2, 2)</td>
<td>377</td>
<td>200</td>
<td>10</td>
<td>(2, 2)</td>
<td>1522</td>
</tr>
<tr>
<td></td>
<td></td>
<td>(5, 1)</td>
<td>981</td>
<td></td>
<td></td>
<td>(5, 1)</td>
<td>2559</td>
</tr>
<tr>
<td></td>
<td></td>
<td>(5, 2)</td>
<td>679</td>
<td></td>
<td></td>
<td>(5, 2)</td>
<td>2589</td>
</tr>
<tr>
<td></td>
<td></td>
<td>(5, 3)</td>
<td>709</td>
<td></td>
<td></td>
<td>(5, 3)</td>
<td>2469</td>
</tr>
<tr>
<td></td>
<td></td>
<td>(5, 4)</td>
<td>653</td>
<td></td>
<td></td>
<td>(5, 4)</td>
<td>2162</td>
</tr>
<tr>
<td></td>
<td></td>
<td>(3, 5)</td>
<td>545</td>
<td></td>
<td></td>
<td>(3, 5)</td>
<td>1992</td>
</tr>
</tbody>
</table>

From Section 4, it is desirable to set \((s_\cap, s_\cup)\) such that \( p_{\ell,i} \approx 1/k \). This suggests that \((s_\cap, s_\cup)\) whose curve comes near the point \((E[f_\ell], 1/k)\) in Figure 2 should perform well on average. Thus, when \( E[f_\ell] = p_M = 0.5 \) and \( k = 4 \), we expect to do well with, e.g., \((s_\cap, s_\cup) \in \{(5, 1), (5, 3)\}\), which is in fact the case in Table 2. Similarly, we see that \((5, 1)\) does well for \( k = 10 \), which also matches expectations. A similar phenomenon occurs for \( E[f_\ell] = p_M = 0.2 \) in Table 3, where curves \((2, 2)\) and \((3, 5)\) look like the best fit in Figure 2 and in fact perform the best for \( k \in \{4, 10\} \).

Note that the values of \( m^* \) in Tables 2 and 3 are worst-case in the sense that they are the maximum number of rows needed to recover all 100 mask set-target pairs. For a more refined analysis, we also plotted the
error of the recovered vector versus $m$ for $1 \leq m \leq 160$. For each value of $m$, for each of the ten mask sets, for each of the ten $\mathbf{x}^*$ vectors, we used our algorithm to build a matrix $A$, recovered an estimate of $\mathbf{x}^*$, and then computed the estimate’s error as its Hamming distance to $\mathbf{x}^*$ divided by $n$. Mean error rates plotted in Figure 3 for $p_M = 0.5$, $k = 4$ and $\ell \in \{100, 200\}$ show that not only does error drop very quickly, but also for $m = 80$, error is at or below 5% for all curves except $(3, 5)$. Thus, we see that if one is tolerant of a small amount of error in the estimate of $\mathbf{x}^*$, it suffices to use 1/4 to 1/3 of the number of rows implied by Table 2.

### 5.1.2 Permutation-Based Masks

We then repeated our experiments with a new method to generate masks. Since Bernoulli trials are not realistic as masks in the chemical fractionation application, we used an alternative procedure to generate masks that is more appropriate. We generated 50 random permutations on the $n = 10^4$ compounds, to simulate different criteria for fractionating solutions, e.g., solubility is one, polarity is another, molecular weight yet another, etc. Then, assuming that one could vary the thresholds of these hypothetical fractionation processes, we specified 20 thresholds for each permutation, distributed uniformly. For a specific (permutation, threshold) pair, all indices $i$ in the permutation that are above the threshold get a one in their corresponding position in their masks, and the others get a zero. E.g., consider the case when $n = 5$ and the random permutation under consideration is $(4, 5, 1, 2, 3)$. Then, if the threshold is 4, the mask is $(1, 1, 0, 0, 0)$ since indices 1 and 2 are at or above the threshold 4 in the permutation. We had 20 such thresholds for each permutation, yielding $\ell = 1000$ masks. We then repeated our experiments, generating ten sets of masks with $\ell = 1000$ masks each, and computed the number of rows $m$ needed to fully recover the same target vectors $\mathbf{x}^*$ from the previous experiments. Table 4 shows the results. If the value for $m^*$ is given as "$ > 2^{14n}$", then that means that the process quit early due to memory constraints and no sufficient value of $m$ was found. Since $m^* > 1000$ for all experiments, we see that the permutation-based masks are more difficult to work with than those based on Bernoulli trials, though positive results are still obtained. More analysis is needed to better understand how our algorithm responds to these types of masks.

### Table 4: Value of $m^*$ to recover support of random $\mathbf{x}^*$ using permutation-based masks and $n = 10^4$.

<table>
<thead>
<tr>
<th>$k$</th>
<th>$(s_{[1]}, s_{(j)})$</th>
<th>$m^*$</th>
<th>$(s_{[1]}, s_{(j)})$</th>
<th>$m^*$</th>
</tr>
</thead>
<tbody>
<tr>
<td>4</td>
<td>$(2, 2)$</td>
<td>4097</td>
<td>$(2, 2)$</td>
<td>$&gt; 2^{14}$</td>
</tr>
<tr>
<td></td>
<td>$(5, 1)$</td>
<td>1701</td>
<td>$(5, 1)$</td>
<td>8127</td>
</tr>
<tr>
<td></td>
<td>$(5, 2)$</td>
<td>1579</td>
<td>$(5, 2)$</td>
<td>10745</td>
</tr>
<tr>
<td></td>
<td>$(5, 3)$</td>
<td>1527</td>
<td>$(5, 3)$</td>
<td>11079</td>
</tr>
<tr>
<td></td>
<td>$(5, 4)$</td>
<td>1803</td>
<td>$(5, 4)$</td>
<td>14363</td>
</tr>
<tr>
<td></td>
<td>$(3, 5)$</td>
<td>3175</td>
<td>$(3, 5)$</td>
<td>$&gt; 2^{14}$</td>
</tr>
</tbody>
</table>

### 5.2 Chemistry Dataset

We then evaluated our method on an existing chemical dataset of 249 compounds provided by Mercier et al. [10]. The target vector $\mathbf{x}^*$ was determined by lab data measuring whether each compound experimentally displayed a specific binding response to a target protein. Six attributes which form the basis of several commonly used chemical separation techniques [13] were used to construct the mask matrix: molecular weight (MW), isoelectric point (pI), log-transformed partition ratio (logP), and
three values of log-transformed distribution ratio (logD) computed at pH \( \in \{5, 7, 9\} \). Predicted values of each of these attributes were calculated using OpenBabel’s obprop and ChemAxon Calculator Plugins [14]. Each of these attributes is real-valued and compound mixtures can be sorted and then separated into two subsets by partitioning at a particular threshold value, similar to the permutation-based mask model of Section 5.1.2. For each sorting of the compounds we chose 10 thresholds to separate on. Specifically, we chose thresholds at the 10 largest differences between adjacent values in each sorted list, which would model the cleanest separations of compounds. The six permutations times ten thresholds each results in a total of 60 ways to separate a solution. For each case, we can choose either separated sub-solution from each of the 60 separations, resulting in \( \ell = 120 \) masks.

To test the case of \( k = 1 \), we selected one of the 18 compounds that displayed a binding response and added it to the 231 compounds that did not. We then calculated the number of rows, \( m \), in each of the 18 cases such that the value yielded no error in recovering the target vector, \( x^* \), as in Section 5.1. These values are in Table 5 and mean error curves are in Figure 4. Table 5 lists each of the 18 compounds that has a binding response (i.e., each of the 18 positives), referenced by an internal database ID. The supplementary material maps each such ID to its IUPAC International Chemical Identifier (InChIKey).

We see in the table that a good selection of \((s_{11}, s_{12})\) brings median \(m^*\) to within about 5 times the value of the theoretical unconstrained minimum of \(\log_2(232)\) = 8 when \(k = 1\). One test randomly managed to beat that number, and surprisingly, the \((s_{11}, s_{12})\) pair that performed best of the combinations tried was \((3, 5)\), a value that visibly underperformed in the synthetic permutation-based experiments of Table 4. In Figure 4, we see that for most values of \((s_{11}, s_{12})\), less than 10% average error is achieved for \(m = 30\).

We took a similar approach to simulate the \(k = 2\) cases, computing \(m^*\) for each of the \(\binom{18}{2}\) pairs of binding compounds. The results are shown in Table 6. Note that doubling \(k\) from 1 to 2 requires median \(m^*\) to increase by a factor of \(< 4\), which is better than the quadratic dependence of \(m\) on \(k\) presented in Section 4.

![Figure 4](image-url)
6 Conclusions

Group testing is a well-studied area with numerous applications, but most approaches assume that arbitrary tests are possible. To address cases when there are constraints on what kinds of tests are feasible (such as chemical analysis via fractionation), we introduced a new group testing model called mask-based constrained group testing, in which constraints on tests are limited to unions of intersections of bit vectors from a given set of masks. We then gave a randomized algorithm to work in our new model and proved that, under the right conditions, it is guaranteed with high probability to identify the active compounds of a solution with a number of tests that grows with $O(k^2 \log n)$, where $n$ is the number of items under test and $k$ is the number of those that are active. Empirical analyses on both synthetic and real data corroborated the efficacy and efficiency of our approach.

Future work on this problem includes a detailed analysis of our algorithm for different mask generation processes that more closely represent reality than an i.i.d. Bernoulli process. In this regard, we are currently analyzing the more realistic random permutation-based mask generation model introduced in Section 5.1.2. In a simplified version of this model, to generate a mask vector $M$, we pick a permutation $\pi$ on $\{1, 2, \ldots, n\}$ and a single threshold $t$ where $1 \leq t \leq n$, both uniformly at random. Then the $i$th bit of $M$ is set to 1 if and only if $\pi(i) \leq t$. A proof that this process also leads to test matrices with parameters comparable to that generated by a Bernoulli process is forthcoming and we plan to include it in a future version of the paper. Also, we note that in some real chemistry applications, one might need to generalize the model from binary masks to real-valued, to indicate the strength of presence of each compound after each fractionation process. Future work is to extend our results to that case.

There are connections between the unconstrained version of our model and the problem of learning monotone $k$-disjunctions in the exact learning model [1] using only membership queries. Specifically, one can view row $i$ of the matrix $A$ as an instance in learning, and the $i$th entry of vector $b$ is that instance’s label, i.e., the output of a membership query. While there are several fundamental differences between exact learning and our mask-based constrained group testing model, it would be interesting to explore whether results from one field can be adapted to the other.

Acknowledgment

The authors thank Robert Powers for his discussions. This work was supported in part by a grant from the Layman Foundation, as well as the National Science Foundation under Grant No. 1422668. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of the NSF.

References