Two New Perspectives on Multi-Stage Group Testing*

Peter Damaschke†, Azam Sheikh Muhammad
Department of Computer Science and Engineering
Chalmers University, 41296 Göteborg, Sweden
[ptr,azams]@chalmers.se
and
Eberhard Triesch
Lehrstuhl II für Mathematik
RWTH Aachen, 52056 Aachen, Germany
triesch@math2.rwth-aachen.de

Abstract

The group testing problem asks to find $d \ll n$ defective elements out of n elements, by testing subsets (pools) for the presence of defectives. In the strict model of group testing, the goal is to identify all defectives if at most d defectives exist, and otherwise to report that more than d defectives are present. If tests are time-consuming, they should be performed in a small constant number s of stages of parallel tests. It is known that a test number $O(d \log n)$, which is optimal up to a constant factor, can be achieved already in $s = 2$ stages. Here we study two aspects of group testing that have not found major attention before. (1) Asymptotic bounds on the test number do not yet lead to optimal strategies for specific n, d, s. Especially for small n we show that randomized strategies significantly save tests on average, compared to worst-case deterministic results. Moreover, the only type of randomness needed is a random permutation of the elements. We solve the problem of constructing optimal randomized strategies for strict group testing completely for the case when $d = 1$ and $s \leq 2$. A byproduct of our analysis is that optimal deterministic strategies for strict group testing for $d = 1$ need at most 2 stages. (2) Usually, an element may participate in several pools within a stage. However, when the elements are indivisible objects, every element can belong to at most one pool at the same time. For group testing with disjoint simultaneous pools we show that $\Theta(sd(n/d)^{1/s})$ tests are sufficient and necessary. While the strategy is simple, the challenge is to derive tight lower bounds for different s and different ranges of d versus n.

Keywords: group testing, randomization, Sperner family, Huffman code, parallel testing, transversal matroid

*Part of the paper has been presented by the first author at the *ICALP Workshop on Algorithms and Data Structures for Selection, Identification and Encoding ICALP2011GT*, Zürich.

†Corresponding author. Tel. 0046-31-772-5405. Fax 0046-31-772-3663.

1 Introduction

1.1 Multi-Stage Group Testing

In the *group testing* problem, a set of n elements is given, that are either *defective (positive)* or *non-defective (negative)*. Let P denote the set of positive elements. It is assumed that $|P| \leq d$, and usually d is very small compared to n. A *group test* takes any subset Q of elements, called a *pool*. The test (or pool) is positive if $Q \cap P \neq \emptyset$, and negative otherwise. In the latter case, obviously, all elements in Q are recognized as negative. The goal is to identify P using a minimum number of tests. A test strategy can be organized in s *stages*, where all tests within a stage must be executed in parallel. In *adaptive* group testing, s is not limited, hence the tests can be done sequentially. A *pooling design* is simply a set of pools. Throughout the paper, log means $\log_2$. We also define $L(n) := \lceil \log n \rceil$. Since the tests are binary, a trivial lower bound on the number of tests is log of the number of possible outcomes, which is roughly $d \log(n/d)$.

A small number s of stages is desirable when the tests are time-consuming. It has been known for a long time [15, 27] that $O(d \log n)$ tests are not sufficient if $s = 1$. It was a breakthrough result [12] that $O(d \log n)$ tests are sufficient already in $s = 2$ stages, and subsequent work [17, 4] has improved the constant factor. See also [10, 26] for complexity results on randomized group testing in a few stages.

Group testing has interesting applications, most notably in biological and chemical testing. For instance, in search for chemical samples that are contaminated with some substance, one can group-test any subset of samples by gathering parts of each sample and pouring them together. Applications in computer science include hardware diagnosis and network communication protocols [11, 18, 21]. We cannot possibly give a survey of the field. Instead we also refer to the textbooks [13, 14]. The present paper makes two independent new contributions to multi-stage group testing.

Randomized strict group testing:

Plenty of strategies are known for different s, assumptions on d in relation to n, and demands on the reliability of outcomes. The primary concern is to minimize the test number. However these results are asymptotic, designed for optimal behaviour when n becomes large. A question that received less attention is the design of *optimal* group testing strategies for *specific* n. In every application one is faced with a specific instance size, but asymptotic results (even optimal ones) do not necessarily entail the optimal strategy for just this n. Another motivation for considering moderate, specific n is that the pool sizes of asymptotically optimal strategies increase with n, but in some applications it may be practically infeasible to test arbitrarily large pools (because of technical obstacles, dilution of chemicals, etc.). Then one may split the given set into many small sets and treat them independently, but now, each one with the provably optimal test number.

We list our notation, along with some comments:

- P denotes the initially unknown set of defectives, and an upper bound d on the number of defectives is assumed.
- A group testing strategy is called *hypergeometric* if it always returns the correct P if $|P| \leq d$, but relies on the assumption that $|P| \leq d$ holds, i.e., this assumption is not checked. (A common phrase is that the property $|P| \leq d$ is "promised" to the searcher.)

- We call a group testing strategy *strict* if it always returns the correct P if $|P| \leq d$, and reports $|P| > d$ otherwise. In the latter case the strategy is not supposed to recognize any defective, but it might still return some partial information about possible sets P. We also refer to [8, Section V.6] for this notion. It was argued earlier in the group testing literature (e.g., [2]) that strict group testing is preferable. The understanding is that $|P| > d$ is unlikely but not impossible, such that in the "good" case $|P| \leq d$, we still need to confirm that no further defectives exist. This seems to be the most useful type of strategies, since they produce safe outcomes.

- We use $t(n, d, s)$ to denote the optimal number of tests needed, in the worst case, by a strict group testing strategy for n elements, at most d defectives, and s stages.

- Strategies may also be *randomized*, yet we want them to be strict. In the randomized case, $\bar{t}(n, d, s)$ denotes the optimal *expected* number of tests of a strict group testing strategy, conditional on $|P| = d$. (Only for technical reasons we take $|P| = d$ rather than $|P| \leq d$, anyway, the complexity is monotone in d.)

Group testing, like any combinatorial search problem, can be viewed as a game between a searcher and an *adversary*, also called *hider*. Here we consider only deterministic strategies. The searcher wants to identify the defectives using a minimum number of tests in the worst case. She chooses a test strategy and sticks to it. The adversary is aware of the searcher's strategy and responds to the tests in such a way that the searcher is forced to use as many tests as possible, for the chosen strategy. (We remark that the adversary must answer consistently, i.e., there must always exist some set P that would generate exactly the given answers if P were the true set of defectives.) Clearly, the test number enforced by the adversary is a lower bound on the worst-case test number for the chosen strategy. Hence, a test number that the adversary can enforce for *arbitrary* deterministic strategies is a lower bound on the deterministic test complexity of the problem. (A more formalized description is given in the Appendix. We also refer to [1] for a general introduction.) An additional standard proof technique is to allow the adversary to *reveal* extra information to the searcher, besides her answers to the tests. Since this only improves the searcher's situation, it cannot increase the lower bound. Hence it is correct to use such adversary actions in lower-bound proofs. The reason for revealing information is that it sometimes simplifies the description of the searcher's instantaneous knowledge and the subsequent arguments.

In this paper we give a complete characterization of the exact values of $t(n, 1, s)$ and $\bar{t}(n, 1, 2)$, as a starting point for tackling this question also for $d > 1$. (A more detailed description and motivation follows in Section 2.)

Group testing with disjoint simultaneous pools:

Group testing strategies with few stages and logarithmic test complexity must have pooling designs where the pools within a stage heavily overlap. Such strategies cannot be used at all if the elements are objects that cannot be split physically, thus an element must not occur in more than one pool at the same time. As an illustrating example imagine that we want to test light bulbs. When we turn on a set Q of bulbs in series connection, then light is on if all bulbs work properly, and light is off whenever some bulbs in Q are defective. Hence light bulbs are "group testable", but we cannot split them. This is probably not a serious application of group testing, however group testing could well be applied to checking other unsplittable items such as electronic chips or other components. For instance, [3] referred to a real-world

application that is best explained by quoting that passage: "The chips were group testable in two phases as follows. In the first phase, a set of chips is exposed to a heating process within a helium environment. If a chip in this set is pervious (defective) then some quantity of helium will penetrate into the chip; otherwise the chip is not affected by the helium. [The heating process does not damage the impervious (good) chips.] In the second phase, the same set is exposed to a helium sensor which records a leak of helium. If at least one chip is defective a leak will be recorded; otherwise no leak will be recorded." Obviously, simultaneous pools must be disjoint, and tests are time-consuming. Even in chemical testing it can be desirable not to split items too much, e.g., simply because only a very limited amount of each sample is available. In fact, the number of pools where an item participates is one of the quality measures for pooling designs in "classical" group testing. Anyway, group testing with disjoint simultaneous pools turns out to be an interesting and surprisingly deep problem in its own right. (A more general model where every item may be shared by at most a fixed number of simultaneous pools would be natural and interesting, too.) In this paper, we determine the asymptotic test complexity of the latter problem for a fixed number s of stages, up to constant factors that even tend to 1 under some conditions.

1.2 Overview of Results and Techniques

Throughout the paper we use n, d, s for the number of elements, defectives, and stages, respectively, and $L(n) := \lceil \log n \rceil$.

After some basic definitions and facts (Section 2.1) we show that only some "canonical" type of randomization, more specifically, a random permutation of the n elements, is needed in optimal group testing strategies (Section 2.2). Basically this is an application of Yao's minimax principle. The value of this observation is that it limits the search space for randomized strategies, for any given parameters n, d, s, to some finite set. Canonical randomizations have also found interest in related fields like property testing [20, 9]. A possible downside is that $O(n \log n)$ random bits are required to produce a random permutation.

Then we use this insight to derive the exact $t(n, 1, s)$ values for $s \leq 2$ (Section 2.3 ff.). The $t(n, 1, 1)$ values were already known (as we explain later in detail), thus it is natural to look at $t(n, 1, 2)$ next. We first settle the deterministic case (Section 2.4) by proving $t(n, 1, 2) = L(n) + 1$ for all n. While $t(n, 1, 2) < t(n, 1, 1)$ holds except for finitely many n, we show that a third stage can never improve the worst-case test number: $t(n, 1, 2) = t(n, 1, s)$ holds for all $s > 2$. Next we prove that $\bar{t}(n, 1, 2)$ is always achieved by one of two randomized strategies. One of them is structurally very simple, while the other one can be characterized in terms of some result in extremal combinatorics. Thus, the problem to figure out the optimal $\bar{t}(n, 1, 2)$ is solved as well. Moreover, the simpler of the two strategies is optimal for large ranges of n above each power of 2 (Section 2.5), and comparison with $t(n, 1, 2)$ exhibits significant savings for those n. We also consider adaptive randomized strategies and show that, essentially, $\bar{t}(n, 1, n)$ is no more than 0.5 expected tests away from the Huffman tree lower bound (Section 2.6).

A remark is that deterministic and hypergeometric group testing for $d = 1$ is trivial: We can encode n elements by strings of $L(n)$ bits, and test $L(n)$ pools, each consisting of the elements that have bit 1 at a fixed position. The answers uniquely determine the defective, and $L(n)$ matches the information-theoretic lower bound. Our optimal strict group testing is based on the same simple idea, but the details turn out to be more tricky.

We argue that the combination of (a) the 1-defective case, (b) strict group testing, and (c) randomization is of interest. Suppose that we have a huge instance with $N \gg n$ elements, but the pool size for tests is limited to n for technological reasons (as mentioned earlier). Further assume that defectives are rare, $d \ll N/n$, and we want to use only a few stages to save time. Then we would split the set randomly into N/n sets of size n, and treat them in parallel. Most likely, each of these instances has no more than one defective – point (a). Only a few instances that happen to have more defectives need to be solved afterwards. Here it is important to apply strict group testing – point (b) – in order to distinguish these two cases. Finally, since we work with *many* small instances, randomized tests that minimize the *expected* test number are advantageous compared to strategies that just optimize the worst-case test number – point (c). Moreover, the percentage of saved tests matters, rather than the absolute expected savings which amount to at most 1 test per instance. Hence it makes sense to "fight for fractions of 1 bit". Besides these practical considerations it is interesting to know the structure of the optimal randomized strategies.

Still it would be nice to get similar results also for general d. As indicated, calculating exact test numbers is already subtle for $d = 1$, and in ongoing work we see a further jump in the difficulty when we go to $d = 2$. We are able to get a bunch of exact $t(n, d, s)$ results also for $d > 1$, but this requires completely different combinatorial methods and is out of the scope of the present paper.

To our best knowledge, only very partial results on exact test numbers exist so far [19, 23], and the case of fixed $s > 1$ is not explored at all. A challenging open question is: Does there exist some constant s such that, for all n and d, the optimal test number can already be achieved in s stages, that is, $t(n, d, n) = t(n, d, s)$? As outlined earlier, the answer is affirmative when it comes to the asymptotic test complexity, but this does not imply anything for exact test numbers.

Our main result for group testing with disjoint simultaneous pools is that its test complexity is essentially $sd(n/d)^{1/s}$. Actually the upper bound is an old and simple result [24]. Our contribution is an almost matching lower bound, that holds at least if $d = o(n^{1/s})$. We emphasize that we consider the problem for any constant number s of stages, and particularly for small s, having in mind that tests are often time-consuming, such that one cannot afford many stages. For easier accessibility we divided the proof into small steps, going from easy to complex, where each step introduces another technical idea. First we define the terminology (Section 3.1). Once more, the case of one defective is central: We first solve the case $d = 1$ (Section 3.2). Here, a balancing argument shows that it is optimal to use, essentially, an equal number of equally sized pools in every stage, and this pattern appears also later for general d. The solution for $d = 1$ suggests already a simple strategy with recursively nested pools. In order to show asymptotic optimality for any d, we have to prove that a deviation from this nested scheme cannot save many tests. This claim is void for $s = 1$ and easy to prove for $s = 2$. We use the idea of revealing a set of defectives that hits all positive pools. These defectives "explain" all positive pools and thus destroy all structure in the description of the searcher's instantaneous knowledge. This simplifies things and allows us to prove a lower bound of $\Omega(d(n/d)^{1/s})$ tests for any fixed s, although first with an underestimated constant factor (Section 3.3). In the following sections we specify this factor better. By a graph-theoretic argument, we prove an almost tight bound, first for $s = 3$, provided that $d = o(n^{1/3})$. Then, it is a small step to extend the result to any fixed number s (Section 3.4).

Up to that point we assumed the number d to be known in advance. If d is unknown, we can get, in stage 1, a randomized estimate of d (Section 3.5). Finally we solve one problem towards raising our lower bound for $s = 3$ stages, for the case that d comes closer to $n^{1/3}$. This part may find independent interest as we develop (to our best knowledge) a new piece of matroid theory: We prove a guarantee for the weight of maximal independent sets in a certain class of weighted matroids stemming from bipartite graphs (Appendix).

Some comparison to related literature is interesting. The test complexity $\Theta(sd(n/d)^{1/s})$ is similar to that of interval group testing [5, 6]. In that version of group testing, the elements are in a fixed total order, and pools must be intervals of consecutive elements. Despite similar complexity, the two group testing versions are not comparable: (a) Our pools are arbitrary subsets, rather than members of some restricted family, and (b), in interval group testing, pools within a stage may overlap. Thus, the two problems also need very different combinatorial tools. Furthermore, we studied the subproblem of estimating the number d of defectives already for classical group testing [10], but for disjoint pools it works quite differently.

2 Randomized Strict Group Testing

2.1 Preliminaries

An *antichain*, also known as *Sperner family*, is a family of sets none of which is subset of another one.

A *pool hypergraph* represents a pooling design as follows: Vertices are pools, edges are elements, and a vertex Q belongs to an edge e if and only if $e \in Q$. At any moment during the execution of a group testing strategy, a *candidate element* is an element that has appeared so far in positive pools only.

The last stage of an optimal strict group testing strategy is always deterministic, even within a randomized strategy with optimal $\bar{t}(n, d, s)$. This is a simple consequence of strictness: The actual pooling design in the last stage must guarantee identification of the defectives if $|P| \leq d$, and recognition of the case $|P| > d$. Hence the best we can do is to use the smallest design with that property, and randomization has no benefits. In particular, optimal strict group testing strategies for $s = 1$ are deterministic, without loss of generality.

A pooling design is *d-disjunct* if its pool hypergraph is *d-cover free*, that is, no union of d edges contains another edge as subset. Equivalently, for every set C of size $|C| \leq d$ and every element $v \notin C$, there exists a pool Q with $v \in Q$ and $C \cap Q = \emptyset$. It is well known that d-disjunct pooling designs recognize up to d defectives and also recognize the presence of more than d defectives. That is, they solve the strict group testing problem for $s = 1$. For the sake of completeness we insert the proof of this fact:

Proposition 1 *A pooling design solves the strict group testing problem in* 1 *stage if and only if it is d-disjunct.*

Proof. Consider a d-disjunct pooling design, and any set C, $|C| \leq d$. If the true set P of defectives equals C then every $v \notin C$ appears in some negative pool and, trivially, no element of C appears in a negative pool. Hence the searcher recognizes C as the complement of the union of all negative pools. The case $|P| > d$ is recognized, too, because in this case, more than d elements remain outside all negative pools. Now, consider a pooling design that is not

d-disjunct. Hence there exist C, $|C| \leq d$, and $v \notin C$, such that no pool Q satisfies $v \in Q$ and $C \cap Q = \emptyset$. Now it may be the case that $P = C$. Since either no pool at all contains v, or every pool containing v also intersects C (and is therefore positive), obviously the searcher cannot confirm $v \notin P$. $\diamond$

That is, calculating $t(n, d, 1)$ and $\bar{t}(n, d, 1)$ is equivalent to the construction of optimal d-disjunct pooling designs. Asymptotic constructions have been studied extensively, however, exact numbers are known for a few n only, and hard to calculate, see [23]. As opposed to $s = 1$, randomization saves tests on average already when $s = 2$, as we shall see later.

2.2 A Canonical Form of Randomized Group Testing

We say that a randomized group testing strategy is *deterministic up to random permutation* if it is obtained as follows: prior to stage 1, the n elements are randomly permuted, that is, all $n!$ permutations are equally likely. Then, some deterministic strategy is applied. This seems very natural, since, in the group testing problem, no set of possible candidates is preferred over others *a priori*. The remarkable fact is that no other randomizations are needed to obtain an optimal expected number of tests. As opposed to this, known strategies (e.g., in [4]) do apply other randomizations to keep the asymptotic analysis simple.

Before we state and prove the theorem it is useful to recapitulate some general remarks. Our $\bar{t}(n, d, s)$ was defined to be the optimal expected number of tests, conditional on $|P| = d$. Any randomized strategy can be viewed as a probability distribution on deterministic strategies. (All possible randomized steps can be decided right from the beginning, and then applied depending on the test answers.) Any deterministic strategy confronted with any defective set P of size d runs with a specific number of tests. Moreover, the number of deterministic strategies is finite for trivial reasons: In every stage, the number of possible pooling designs and test outcomes is limited by some function of n, and s is fixed.

Theorem 2 *For any fixed n, d, s there is always a randomized strict group testing strategy with optimal $\bar{t}(n, d, s)$ which is deterministic up to random permutation.*

Proof. Consider a matrix where each row represents a deterministic strategy σ, and each of the $m := \binom{n}{d}$ columns represents a set P of d elements. Every matrix entry $q_{\sigma,P}$ denotes the test number of strategy σ, if P is the set of defectives.

Consider some row ρ where the average of test numbers $t := (1/m) \sum_P q_{\rho,P}$ is minimal. We show that no randomized strategy can yield an expected test number below t for all P. Since a randomized strategy is a probability distribution on the rows, let $\pi(\sigma)$ denote the probability of choosing strategy σ. Then we have $\sum_\sigma \pi(\sigma)(1/m) \sum_P q_{\sigma,P} \geq t$, due to the minimality of t. Hence there exists some column P where $\sum_\sigma \pi(\sigma) q_{\sigma,P} \geq t$. By linearity of expectation, this weighted sum is also the expected test number when this P is the true set of defectives.

Now we give a strategy that has the claimed form and yields an expected test number t on every P, and is therefore optimal. We apply the deterministic strategy ρ corresponding to our specified row with minimum t, but first we randomly permute the n elements. The resulting strategy corresponds, in our matrix, to a set S of $n!$ rows, each of which is chosen equiprobably and has the same row average t. We claim that every column restricted to S contains exactly the same multiset of test numbers, hence their average is t again: $(1/n!) \sum_{\sigma \in S} q_{\sigma,P} = t$ for

every P. This claim is evident from a symmetry argument: Since the subsets P of the same size d are not distinguished by the group of all permutations of the n elements, they cannot behave differently in the devised randomized strategy. $\diamond$

We remark that the proof did not really need the stage number s and the strictness property, but we have formulated Theorem 2 in this way, just to serve our specific purpose. Based on Theorem 2 we will now design provably optimal strategies for one defective in two stages.

2.3 Some Lemmas for the One-Defective Case

Pooling designs that solve the strict group testing problem for $d = 1$ in one stage are exactly those with the pool hypergraph being an antichain. We also refer to them as *antichain designs*. Hence, minimizing $t(n, 1, 1)$ and $\bar{t}(n, 1, 1)$ amounts to the problem of constructing maximal antichains, which is solved by Sperner's theorem. Thus we will immediately go on to the study of $t(n, 1, 2)$ and $\bar{t}(n, 1, 2)$.

We observe that, for $s = 1$, an antichain design is still necessary if the existence of some defective is already guaranteed, i.e., this extra information does not help the searcher. This holds because, if some edge e contains another edge f in the pool hypergraph, and element e happens to be positive, we cannot infer the status of f. Another trivial but useful observation is that, whenever $|P| \leq 1$ and $n > 1$, in any antichain design that solves the strict group testing problem, at least one pool will be negative.

A pooling design, say with t pools, divides the set of elements into 2^t disjoint subsets of elements that we refer to as *cells*, where the elements in each cell belong to exactly the same pools. Note that a cell may be empty. The cell of elements being in no pool is called the *uncovered cell*. All other cells are *covered*. Provided that $|P| = 1$, a positive (negative) pool Q tells us that the only defective is in Q (is in the complement of Q). Hence the answers from a pooling design in stage 1 specify exactly one cell that may contain the defective. We call it the *candidate cell*. This should not be confused with the earlier notion of a candidate element: There may exist candidate elements outside the candidate cell, since the searcher is not yet sure that $|P| \leq 1$. Furthermore, if the uncovered cell is the candidate cell, then possibly $|P| = 0$.

Lemma 3 *Let $d = 1$, and consider the last stage of a strict group testing strategy. If the candidate cell C resulting from the previous stages has more than one element, then it is optimal to do the last stage as follows: Take an optimal antichain design in C and add to each pool the complement of C.*

Proof. As stated earlier, the last stage can be assumed to be deterministic.

First we show sufficiency of the tests in the last stage. If $|P| \leq 1$ then at least one pool of the antichain design is negative (as observed above), and since every pool contains the complement of C, all elements outside C will be confirmed to be negative. Moreover, the antichain design solves the strict group problem restricted to C. Note that the cases $|P| = 0$ and $|P| > 1$ are detected as well.

Secondly we show that fewer pools are not sufficient. Assume that a helper reveals that no defective is outside C. Even with this extra information, the searcher still has to solve the strict group testing problem in C. Thus, the intersections of our pools (in the last stage) with C must form an antichain design, and we took an optimal one. $\diamond$

The following cases are easy to check directly:

Lemma 4 *Let $d = 1$, and consider the last stage of a strict group testing strategy. If the candidate cell C resulting from the previous stages has exactly one element, then it is optimal to do the last stage as follows:*
If C is a covered cell, and all other elements are already recognized as negative, then the only candidate element must be positive, thus no further tests are needed. We call this case a "verified defective".
If C is a covered cell, but further candidate elements exist, then exactly one further test is needed to falsify these other candidate elements.
If C is the uncovered cell, then exactly one further test is needed to check whether the candidate element is positive. $\diamond$

We call a cell with k elements a *k-cell*. Let $a(k)$ be the size of an optimal antichain design in a k-cell, for any $k \geq 1$. For trivial reasons, $a(k)$ is monotone increasing in k. We also remark, for later use, that $a(k) = k$ for $k \leq 4$. Sperner's classical theorem [28, 25] implies that $a(k)$ is the minimum integer a where $\binom{a}{\lfloor a/2 \rfloor} \geq k$.

Lemma 5 *For all $k > 1$ it holds that $a(k-1) \leq a(k) < a(2k-1) \leq a(2k)$.*

Proof. If a is even, we have $\binom{a+1}{a/2}/\binom{a}{a/2} = (a+1)/(a/2+1) < 2$. If a is odd, we have $\binom{a+1}{(a+1)/2}/\binom{a}{(a-1)/2} = (a+1)/((a+1)/2) = 2$. That is, the largest k with a given a value is at most doubled when we increment a. Now consider any fixed k. Trivially, the largest argument with function value $a(k)-1$ is at most $k-1$. Thus the largest argument with function value $a(k)$ is at most $2k-2$, which yields $a(2k-1) > a(k)$ as claimed. The other inequalities follow by monotonicity. $\diamond$

2.4 Optimal Deterministic Strategies for One Defective

The previous lemmas have already determined much of the structure of a strict group testing strategy that minimizes $t(n, 1, 2)$. What remains to be determined is the number t of tests in stage 1, and the sizes of the 2^t cells. Lemmas 3 and 4 say that, if the candidate cell is a k-cell, then $a(k)$ tests are needed in stage 2, unless we get a verified defective. Remember $L(n) := \lceil \log n \rceil$.

Theorem 6 *For all n, the optimal deterministic strict group testing strategy for 1 defective in 2 stages requires $t(n, 1, 2) = L(n) + 1$ tests in the worst case. For almost all n, the second stage is actually needed to accomplish this test number, with the following exceptions where one stage suffices: $n = 1, 2, 3, 5, 6, 9, 10, 17, 18, 19, 20, 33, 34, 35, 65, 66, 67, 68, 69, 70$.*
Moreover, more stages do not improve the worst-case test number, that is, we have $t(n, 1, 2) = t(n, 1, s)$ for all n and all $s \geq 2$.

Proof. In stage 1 we use a pooling design with $L(n)$ tests that produces 0-cells and 1-cells only, where the 1-cells are placed arbitrarily. (In other words: We encode the n elements arbitrarily by distinct strings of $L(n)$ bits. Then every bit string that is actually used yields a 1-cell, while the others yield 0-cells.) Then, by Lemma 4, at most one further test suffices in stage 2.

In order to prove optimality of the test number (even if more stages were permitted), we define $t'(n)$ as the optimal worst-case test number of an adaptive strict group testing strategy for one defective, where the searcher even gets the additional *a priori* information that some defective is present. We have $t'(1) = 0$ by definition, and we claim that $t'(n) \geq L(n) + 1$ for $n > 1$.

We prove the claim by induction, with $n = 2$ as the base case. Denote the two elements e and f. Then the pool $\{e, f\}$ is always positive, thus not informative. When we test exactly one element, say e, and e happens to be the defective, then the positive answer leaves us uncertain about f: it could be defective or not, as $|P| = 1$ is not known in advance. Hence $t'(2) = 2 = L(2) + 1$.

For the induction step suppose that the claim holds for $1 < n \leq 2^t$, for some t. Consider $2^t < n \leq 2^{t+1}$, and test a pool of k elements first. If the test is positive, we have to solve the problem for k elements (and confirm that no defective is in the complement). If the test is negative, we have to solve the problem for $n - k$ elements. (Note that we gave the searcher the extra knowledge that some defective is present, hence if no defective is in the pool, there must be some in the complement set.) Thus we have $t'(n) \geq 1 + \min_k \max\{t'(k), t'(n-k)\}$. Since t' is monotone, the best choice is $k = \lfloor n/2 \rfloor$, and the claim follows by induction.

Finally we determine those n where $t(n, 1, 2) = t(n, 1, 1)$. Recall that a strict 1-stage strategy must apply an antichain design, which needs $a(n)$ tests. It remains to figure out all n with $a(n) \leq L(n)+1$. For $n \leq 70$ this is easy to do with little calculation (we skip the details), which leads to the list above. We show that $a(n) > L(n) + 1$ for $n > 70$. Observe $a(71) = 9$ and $L(71) + 1 = 8$, and further $L(128) + 1 = 8$, which establishes the strict inequality for $71 \leq n \leq 128$. Next observe $a(129) = 10$, whereas $L(129) + 1 = 9$. The rounded logarithm $L(n)$ grows only after each power of 2, hence Lemma 5 yields that $a(n)$ grows at least that fast, which maintains the strict inequality. $\diamond$

Theorem 6 solves the problem of optimal deterministic strict group testing strategies for $d = 1$ and all s completely. Using randomization we can still achieve better *expected* query numbers, which is the next issue.

2.5 Optimal Randomized Strategies for One Defective in Two Stages

In the following we calculate strategies that minimize $\bar{t}(n, 1, 2)$ and enjoy the structure specified in Theorem 2. Our next step is to determine the optimal cell sizes in stage 1. We remind the reader that a pooling design with t pools can be viewed, equivalently, as an assignment of t-bit vectors to each element, and every pool comprises the elements that have a 1 at a specific position. Elements that are assigned the same t-bit vector form a cell. To "move" an element from a cell to another cell (which is a very convenient expression in the following proof) simply means to change the bit vector of that element.

Lemma 7 *There is a strict group testing strategy minimizing $\bar{t}(n, 1, 2)$ that produces, in stage 1, either 1-cells and 2-cells only, or 0-cells and 1-cells only, and in the latter case the 1-cells are the majority.*

Proof. Consider a $(b + 1)$-cell and a c-cell, where $b > c$. Let us move one element from the larger to the smaller cell, thus obtaining a b-cell and a $(c + 1)$-cell. Due to Theorem 2, in the case $|P| = 1$, every element after applying the random permutation is the defective with the

same probability $1/n$. We study how the move improves the expected number of tests. To this end we use Lemmas 3 and 4, and for convenience we omit the common factor $1/n$ in the test number.

If $b \geq 1$ and $c = 0$ then the improvement is at least $b \cdot (a(b+1) - a(b)) + (a(b+1) - 1) \geq 0$. Hence we can fill all 0-cells with elements from cells of more than one element, without deteriorating the expected test number. If $b \geq 2$ and $c = 1$ then the improvement is at least $b \cdot (a(b+1) - a(b)) - 2 + (a(b+1) - 2)$. This expression is also nonnegative, since $a(2) = 2$, $a(3) = 3$, $a(4) = 4$. It follows that we can enlarge all 1-cells without making the expected test number worse. We repeat this step until only 1-cells and 2-cells remain, or every cell has at least 2 elements.

In the latter case we invest one more test in stage 1, thus doubling the number of cells, and we split every $2k$-cell in two k-cells, and every $(2k-1)$-cell in a k-cell and a $(k-1)$-cell. Remember that $a(k)$ tests are needed if the defective happens to be in a k-cell, $k \geq 2$, and at most 1 test if $k = 1$. Using Lemma 5 we get that the split improves the expected number of tests in stage 2 by at least 1. This compensates for the extra test in stage 1. We conclude that we always reach one of the two cases in the statement of the Lemma.

In the case of 0-cells and 1-cells, assume that the 1-cells are not the majority of cells. Then we do the opposite action: We save one test in stage 1 and redistribute the elements arbitrarily to the cells, placing at most one element in each cell. Since 0 or 1 tests are needed if the defective is in a 1-cell (by Lemma 4), the expected number of tests for stage 2 is raised by at most 1, therefore the total expected test number does not increase either. $\diamond$

Definition 8 *The* k-lattice *is the set of all subsets of a fixed set of* k *elements. Let* F *be a family of exactly* n *sets in the* k*-lattice. A set in* F *is* minimal *(with respect to inclusion) if it contains no other set of* F *as a subset. Clearly, the minimal sets in* F *always form an antichain. For* n *not being a power of* 2*, we define* $M(n)$ *to be the largest possible number of minimal sets in a family of exactly* n *sets in the* $L(n)$*-lattice. For* n *being a power of* 2*, we set* $M(n) := 0$*.*

About calculating $M(n)$:

By virtue of $M(n)$ we will characterize the optimal randomized 2-stage strict group testing strategies for one defective. The $M(n)$ and corresponding optimal antichains in the $L(n)$-lattices can be efficiently computed for all n, even in the more general lattices of multisets [7]. We emphasize that $M(n)$ and an optimal F can be precomputed for any considered n, and this calculation is not part of the test complexity.

To explain how the $M(n)$ are obtained, consider n between any fixed consecutive powers of 2, that is, $2^{k-1} < n < 2^k$, where $k = L(n)$. Note that $n > 2^{L(n)}/2$. According to Definition 8 we want to choose a family F of n sets in the $L(n)$-lattice so as to maximize the number of sets being minimal in F. By Sperner's theorem, a largest antichain in the $L(n)$-lattice is given by the sets of size $\lfloor L(n)/2 \rfloor$; we refer to it as the *Sperner family*. In particular, F itself is not an antichain, since the Sperner family is smaller than $2^{L(n)}/2 < n = |F|$. It also follows that, if the $L(n)$-lattice contains at most n sets of size at least $\lfloor L(n)/2 \rfloor$, an optimal choice for F is simply the Sperner family filled up with any selection of $n - \binom{L(n)}{\lfloor L(n)/2 \rfloor}$ larger sets. This simple case applies to some interval of values n above 2^{k-1}. As $n < 2^k$ is getting larger, the Sperner family and all larger sets together do not contain enough sets any more; we will call this case an *overflow*. Then we must gradually insert in F some sets with less than $\lfloor L(n)/2 \rfloor$ elements. Specifically, these sets added to F are always sets of the largest

cardinality and in colexicographic order, that were not yet in F. This guarantees a maximum antichain, due to the Kruskal-Katona theorem. (To define the colexicographic order $<$ of subsets, let be $S := \{1, \ldots, k\}$. For any two subsets A and B of S, we say $A < B$ if and only if $\sum_{a \in A} 2^a < \sum_{b \in B} 2^b$.) We skip further details and refer to [7] and the pointers therein. We remark that, for any fixed k, the $M(n)$ are monotone non-increasing for $2^{k-1} < n < 2^k$, which follows straight from Definition 8: Given an optimal F of n sets, simply remove some non-minimal set (F is not an antichain), in order to see $M(n-1) \geq M(n)$.

We demonstrate that, in fact, the $M(n)$ are now easy to calculate. The $M(n)$, $n \leq 11$, in Table 1 are easily checked. The first overflow case is $n = 12$, since $\binom{4}{2} + \binom{4}{3} + \binom{4}{4} = 11$. Verify that the antichain of minimal sets in F cannot have exactly 5 members, but the 1-element sets obviously form an antichain of size 4, which yields $M(12) = \ldots = M(15) = 4$. To discuss the interval $16 < n < 32$, observe that $\binom{5}{2} + \binom{5}{3} + \binom{5}{4} + \binom{5}{5} = 26$, hence $M(17) = \ldots = M(26) = 10$. The optimal antichain of minimal sets for the first overflow case $|F| = 27$ in the 5-lattice consists of $\{1\}, \{2,3\}, \{2,4\}, \{2,5\}, \{3,4\}, \{3,5\}, \{4,5\}$, hence $M(27) = 7$. Similarly as above, the antichain of minimal sets in F cannot have exactly 6 members, but the 1-element sets obviously form an antichain of size 5, which yields $M(28) = \ldots = M(31) = 5$, and so on.

The test strategy:

First we calculate $\min\{L(n) + 2 - 2^{L(n)}/n, L(n) + 1 - M(n)/n\}$. If the first term is the minimum, we set up, in stage 1, the pooling design with bit vectors of length $L(n) - 1$, where every bit vector is assigned once or twice. This is possible since $n/2 \leq 2^{L(n)-1} < n$. That is, only 1-cells and 2-cells are generated; a nice detail is that the 2-cells can be chosen arbitrarily. Depending on the candidate cell size we apply Lemma 3 or 4 in stage 2. If the second term is the minimum, we set up a pooling design with bit vectors of length $L(n)$, where every bit vector is assigned at most once, thus generating only 0-cells and 1-cells. The 1-cells are arranged such that the pool hypergraph, which is a family of n subsets of the $L(n)$-lattice, has $M(n)$ minimal edges. Then, Lemma 4 is applied in stage 2. (If both terms are equal, we can go either way.) Now we are going to prove optimality of this strategy.

Theorem 9 *An optimal randomized strict group testing strategy finding* 1 *defective in* 2 *stages uses*

$$\bar{t}(n, 1, 2) = \min\{L(n) + 2 - 2^{L(n)}/n, L(n) + 1 - M(n)/n\}$$

tests on average. In particular, we have

$$L(n) < \bar{t}(n, 1, 2) \leq L(n) + 1.$$

Proof. By Theorem 2, every element is the defective with probability $1/n$, provided that $|P| = 1$. Now consider the two options in Lemma 7.

If we decide on 1-cells and 2-cells, then we never get a verified defective (see Lemma 4). This holds because all cells contain elements, in particular the uncovered cell does. Hence there are always candidates outside the (covered) candidate cell. It follows from Lemma 3 and 4 that it does not matter in which way we place one or two elements in the cells. We need exactly k tests in stage 2 if the defective is in a k-cell ($k = 1, 2$). Since n elements are located in $2^{L(n)-1}$ cells, the number of elements in 2-cells is $2(n - 2^{L(n)-1})$. This yields the expected test number $(L(n) - 1) + 1 + 2(n - 2^{L(n)-1})/n = L(n) + 2 - 2^{L(n)}/n$.

If we decide on 0-cells and 1-cells, then a defective is verified if and only if it corresponds to a minimal edge in the pool hypergraph. By Lemma 4, no further test is needed for them, while

Table 1: Optimal expected test numbers for 1 defective in 2 stages.

n	1	2	3	4	5	6	7	8	9	10	11	12
$\bar{t}(n,1,2)$	1.00	2.00	2.33	3.00	3.40	3.50	3.57	4.00	4.22	4.40	4.45	4.67
$M(n)$	0	0	2	0	3	3	3	0	6	6	6	4
strategy	1,2	1,2	0,1	1,2	1,2	0,1	0,1	1,2	1,2	1,2	0,1	0,1
$\bar{t}/t$	1.00	1.00	0.78	1.00	0.85	0.88	0.89	1.00	0.84	0.88	0.89	0.93
n	13	14	15	16	17	18	19	20	21	22	23	24
$\bar{t}(n,1,2)$	4.69	4.71	4.73	5.00	5.12	5.22	5.32	5.40	5.48	5.55	5.57	5.58
$M(n)$	4	4	4	0	10	10	10	10	10	10	10	10
strategy	0,1	0,1	0,1	1,2	1,2	1,2	1,2	1,2	1,2	0,1	0,1	0,1
$\bar{t}/t$	0.94	0.94	0.95	1.00	0.85	0.87	0.89	0.90	0.91	0.92	0.93	0.93
n	25	26	27	28	29	30	31	32	33	34	35	36
$\bar{t}(n,1,2)$	5.60	5.62	5.74	5.82	5.83	5.83	5.84	6.00	6.06	6.12	6.17	6.22
$M(n)$	10	10	7	5	5	5	5	0	20	20	20	20
strategy	0,1	0,1	0,1	0,1	0,1	0,1	0,1	1,2	1,2	1,2	1,2	1,2
$\bar{t}/t$	0.93	0.94	0.96	0.97	0.97	0.97	0.97	1.00	0.87	0.87	0.88	0.89

non-verified defectives need exactly one further test. This immediately yields the expected test number $L(n) + 1 - M(n)/n$, when we use a pooling design with the maximum number $M(n)$ of minimal edges. $\diamond$

For brevity we call the two options the $1,2$-strategy and the $0,1$-strategy. Note that the $0,1$-strategy is optimal if and only if $M(n) \geq 2^{L(n)} - n$. We report the optimal results for $n \leq 36$ in Table 1. The $\bar{t}(n,1,2)$ values are rounded to two decimals. The strategy row indicates which of the two specified strategies from Theorem 9 is optimal, and the $\bar{t}/t$ row gives the ratio of the randomized and deterministic test number. We see that several percent of costs are saved, in particular if n is slightly above a power of 2.

The strategies in Theorem 9 do not only minimize the expected test number. Moreover, they never use more than $L(n) + 1$ tests in the worst case, which is also optimal due to Theorem 6. This property is not self-evident for combinatorial search problems: For instance, the problem of sorting by comparisons can be viewed as a combinatorial search problem (find an unknown permutation). Quicksort is a fast randomized strategy, but its worst-case bound is far higher than a deterministic upper bound one can achieve in a different way.

2.6 Randomized Adaptive Strategies for One Defective

With the help of Theorem 2 we also get optimal randomized adaptive group testing strategies in the case of one defective. In strict group testing, the task was to identify the single defective and confirm that it is the only one. In the adaptive case we want to characterize the expected $\bar{t}(n,1,n)$. (It does not harm to set $s = n$, although actually some $\log n$ queries and stages are sufficient.) By Theorem 2 it suffices to study deterministic strategies up to random permutation, and due to this random permutation we can assume that every element is the defective with probability $1/n$, provided that $|P| = 1$.

First let us consider a simpler problem: Suppose the searcher does already know that $|P| = 1$. Then every group test on a pool Q obviously divides the set of possible outcomes: The defective is Q (in the complement of Q) if Q was positive (negative). Thus, a deterministic

strategy can be viewed as a binary tree with the elements as leaves, and minimizing the expected number of tests is nothing else than constructing an optimal Huffman tree [22] for the uniform distribution. It is well known that the optimal Huffman trees in this case are the full binary trees characterized as follows: All levels are full (the kth level has 2^k nodes), except the last level if n is not a power of 2. The leaves in the last level form pairs of siblings, but apart from that, their placement is arbitrary. Let $T(n)$ denote the average path length of any such tree. We have $T(n) = \log n$ if n is a power of 2, but $T(n)$ is also very easy to express analytically for general n; we skip these details as we do not need them in the sequel.

Now we approach our original problem where the searcher is not sure in advance that $|P| = 1$. Clearly $\bar{t}(n, 1, n) \geq T(n)$, but the question is what is the price for verifying the existence of only one defective. We can construct the optimal strategies for all n, and a nice upper bound holds:

Theorem 10 *We have* $\bar{t}(n, 1, n) < T(n) + 0.5 + O(1/n)$.

Proof. Let us first assume that $|P| \geq 1$ is already known. In the end we will adjust the result to possible absence of defectives. For notational clarity we use again the notion of candidate cell C. After each test with a pool Q, the new candidate cell is $C \cap Q$ if the test was positive, and $C - Q$ else. When the search has narrowed down the candidate cell C to 2 elements, say $C = \{e, f\}$, we need another 1.5 expected queries (provided that $|P| = 1$). This is accomplished as follows. In the next pool we put e, together with all candidate elements that may still exist outside C. With probability 0.5 the test is negative, thus we also learn that f is defective. With probability 0.5 the test is positive, and then we have to test the other elements in one further pool, in order to verify that all are negative. Hence 1.5 expected queries are enough. This is also optimal in the specified situation: Since C is the candidate cell, it contains the defective, and elements outside C are not defective (provided that $|P| = 1$), but the latter must be confirmed by the searcher. Hence a pool including none or both elements of C would just be wasted, and including all elements outside C is not a mistake.

Now consider any binary tree with n leaves, representing an adaptive test strategy that is deterministic up to random permutation. Let u be any inner node, and v, w its children. Let $C(u)$ denote the set of leaves in the subtree rooted at u, and note that $C(u)$ is the candidate cell when the search has reached node u. Either of $C(v)$ and $C(w)$ could be used as the pool corresponding to u. If $|P| = 1$ were known a priori, both choices would be equally good, but since we only know $|P| \geq 1$, it is important how we choose the pools. Specifically we do the following. If both v and w are leaves, we proceed as in the previous paragraph, and we are done with 1.5 expected further queries. If exactly one child of u, say v, is a leaf, the next pool consists of all candidate elements except v. Hence, if the test is positive, we proceed to w, and if the test is negative, we can stop and output v as the defective. If no child of u is a leaf, the next pool is any of $C(v)$ and $C(w)$. This strategy is optimal for the given tree, since optimality for a node with two leaf children was shown above, and no further verification tests have been introduced at other leaves.

We can interpret the result for the fixed tree as follows. For every pair of sibling leaves, 0.5 expected further tests are needed in addition to the path length, if some of them is the defective. No further tests are needed if some leaf with a non-leaf sibling is the defective. Now we are also able to optimize the shape of the tree. Suppose that u has two leaves v and w as children, and some leaf x is closer to the root than u. Then we attach v and w as children

to x, while u becomes a leaf. This reduces the expected path length by at least $2/n$. The verification queries at v and w are the same as before. In the worst case we create a new pair of sibling leaves (u and its sibling), which causes $1/n$ expected new verification queries. Still we have improved the tree. By iterating this step it follows that the optimal tree is a full binary tree. Next, remember that the pairs of sibling leaves in the last level can be placed arbitrarily, without affecting the expected path length. We attach them to the nodes of the second last level in such a way that a maximum number of leaves therein has a non-leaf as sibling. (The greedy procedure for this is trivial.) Since any full binary tree is a Huffman tree, in particular, the expected number of tests is $T(n) + 0.5$ or better.

We have to make some final adjustments to accommodate the possibility that $|P| = 0$. Since $|P| = 0$ can occur, the unique tree path with only negative answers ends in a leaf u for which the searcher does not know whether it is defective. Thus we need one extra test there, in addition to the path length. That is, if u is the defective, we must confirm this as well, but this happens only with probability $1/n$ and adds $1/n$ to the expected test number. Furthermore, if really $|P| = 0$, then we get to this leaf with probability 1. In order to avoid that the path length plus 1 exceeds the average test number that we got for $|P| = 1$, we can move u one level higher and, in exchange, move $O(1)$ nodes to the last level, which obviously adds $O(1/n)$ to the expected test number, too. $\diamond$

Since $T(n) = L(n) - (1/n) * (2^{L(n)} - n) \leq \log_2 n + 0.086$ is known [1, Theorem 1.9], our bound can be stated more explicitly as $\bar{t}(n, 1, n) < \log_2 n + 0.586 + O(1/n)$.

3 Group Testing with Disjoint Simultaneous Pools

3.1 Notation and Basic Facts

We start defining the terminology needed in this section. The notions of positive (defective) and negative elements, group testing problem, pools, and stages were already given in Section 1.1. A *candidate* element is an element that has not appeared so far in a negative pool, thus it may still be positive. Parameters n, d, s are used as before. We explicitly mention the difference between hypergeometric and strict group testing (see Section 1.1) only when it is relevant for the complexity results. To avoid heavy notation we will sometimes silently omit lower-order terms in complexity bounds, in particular, rounding brackets for fractional numbers are omitted.

From now on we exclusively deal with the version of group testing where pools within the same stage must be disjoint: Whenever two pools Q and Q' are tested simultaneously in a stage, they must satisfy $Q \cap Q' = \emptyset$. Due to this demand, any stage naturally induces a partitioning of the candidate elements: The *parts* of this partitioning are the pools used in that stage, and the *rest*, which is the set of elements being in no pool in that stage.

A set D is called a *hitting set* of a given family of sets, if D has a nonempty intersection with every set of this family. For our proofs we will also need some standard graph-theoretic notion. A *graph* is a pair $G = (V, E)$ of a vertex set V and an edge set E, where every edge is an unordered pair of vertices. An edge between vertices x and y is denoted xy. An edge xy and each of its vertices x and y are said to be *incident*, and the vertices x and y are called *adjacent*. A graph is *bipartite* if V can be split in two disjoint sets X and Y, called the *partite sets*, such that every edge is incident to vertices from both X and Y. In order to stress that a graph is bipartite we may write it as $G = (X, Y; E)$. A *path* is a sequence of vertices such

that any two consecutive vertices are adjacent. A *cycle* is a path starting and ending in the same vertex. In a *connected* graph, any two vertices are joined by some path. A connected and cycle-free graph is a *tree*, and a cycle-free graph (not necessarily connected) is a *forest*. If $G = (V, E)$ is connected, a *spanning tree* of G is any tree $T = (V, F)$ with $F \subseteq E$. Similarly, a *spanning forest* of an arbitrary graph G is any forest $T = (V, F)$ with $F \subseteq E$. The attribute "spanning" refers to the demand that T must contain all vertices of G. A *matching* is a set $M \subseteq E$ of edges that are pairwise disjoint, i.e., do not share vertices. We say that M *covers* the $2|M|$ vertices appearing in the edges of M.

3.2 The One-Defective Case and a Simple General Strategy

Adapting an old result [24], we first get the asymptotic complexity in the case $d = 1$ (which will be a building block of our reasoning for general d), and a simple strategy for any d. Then, almost the whole remainder of Section 3 is devoted to proving asymptotic optimality of the resulting upper bound, if d grows slowly compared to n.

Proposition 11 *Any strategy for $d = 1$ needs at least $sn^{1/s} - s$ tests in the worst case.*

Proof. As introduced above, every stage divides the candidate elements into parts. The adversary reacts as follows in every stage. She chooses some part Q of maximum size. If Q is a pool, the answer to Q is positive, and all other pools are negative. If Q is the rest, then all pools are negative. In the case of hypergeometric group testing it is obvious that, after any stage, the candidate elements are exactly the elements in Q. In the strict group testing model, however, the rest may contain further defectives when some pool was positive. In this case our adversary also reveals that no defective is in the rest, such that we arrive at the same statement: After any stage, the candidate elements are exactly the elements in Q.

Against this adversary it is always optimal for the searcher to make the parts equally large (with differences of at most 1), because this minimizes the size of the maximum part Q specified above. Let p_i be the number of parts in stage i. Then stage i reduces the number of candidate elements to a fraction $1/p_i$. Hence the searcher must choose the p_i such that $p_1 \cdot \ldots \cdot p_s \geq n$, and she wants to minimize $p_1 + \ldots + p_s$. By routine calculation, her optimal choice is to set all $p_i = n^{1/s}$, from which the claimed lower bound follows. Note that we subtract s since the number of tests in every stage is $p_i - 1$. $\diamond$

The lower bound in Proposition 11 is tight, up to minor additive gaps.

Proposition 12 *We can identify one defective using $sn^{1/s} - s$ and $sn^{1/s} - s + 2$ tests, in the hypergeometric and the strict group testing model, respectively.*

Proof. In every stage, divide the candidates into $n^{1/s}$ parts of equal size. If some pool is positive, we hand it over to the next stage, otherwise we hand the rest over to the next stage. Obviously we will eventually find the defective. In the case of strict group testing we also have to check that only one defective is present: We proceed as above in all stages but the last one. In the last stage we test *all* elements from the received part individually, and we test one pool consisting of all rests of earlier stages that exhibited some positive pool. The latter test is done in order to rule out further defectives that may be hidden in these rests. Thus we need two more tests than in the hypergeometric model. $\diamond$

We call a strategy *nested* if every pool in a stage is a subset of some part (pool or rest) from the previous stage. The above strategy for one defective is nested and suggests the following nested strategy for the general case, cf. [24].

Theorem 13 *We can identify up to d defectives using at most $sd(n/d)^{1/s}$ tests.*

Proof. First consider the hypergeometric model. In stage 1 we partition the elements into p pools of size n/p, where p will be specified below. First suppose that d pools are positive. Then we treat them as d instances of the problem with one defective, to be solved in the remaining $s-1$ stages. By Proposition 12, we can solve each with $(s-1)(n/p)^{1/(s-1)}$ tests. (For simplicity we omit the minor additive term $s-1$ and take a slightly worse bound.) Specifically, we choose p so as to minimize $p + (s-1)d(n/p)^{1/(s-1)}$. Standard calculation yields $p = d(n/d)^{1/s}$, hence $n/p = (n/d)^{1-1/s}$ and $(s-1)d(n/p)^{1/(s-1)} = (s-1)d(n/d)^{1/s}$. We remark that the same number of tests is used in every stage, and that exactly d pools in every stage are positive.

If fewer than d pools are positive in stage 1, we simply continue on these positive pools, using the same pool sizes as above, in every stage. What is new is that positive pools may now contain several defectives. Thus, when we split a positive pool into p smaller pools in the next stage, several of them may answer positively. But since all pools in a stage are disjoint, trivially, the total number of positive pools in every stage is still bounded by d. Hence the total number of tests in all stages together gets only smaller than in the case when d pools were positive already from stage 1 on.

In the strict group testing model, nothing changes in the strategy, because we only discard elements in negative pools. Here the only difference is that more than d disjoint pools can be positive, but in this event we stop. $\diamond$

3.3 Revealing Defectives and a Preliminary Lower Bound

One might think that $sd(n/d)^{1/s}$ from Theorem 13 (actually from [24]) is also an immediate lower bound: An adversary reveals a partitioning into d sets of size n/d with one defective each, and by Proposition 11 roughly $s(n/d)^{1/s}$ tests are needed to find every defective. However, note that the searcher is not bound to search the d sets independently. Actually, the previous argument proves a matching lower bound only for nested strategies. Without this restriction, the searcher may use pools that overlap several earlier parts. The question is if such intersections can save tests. As our next step, we actually establish a simple lower bound of $\Omega(d(n/d)^{1/s})$ for every fixed s, that is, with some hidden factor depending on s only.

Observe that the possible sets D of defectives at any moment during the execution of a search strategy are characterized by three conditions: $|D| \leq d$; D contains only candidate elements; and D intersects every positive pool, in other words, D is a hitting set of the family of positive pools restricted to the candidate elements.

The technique of revealing extra information was explained in Section 3.1. In particular, the adversary may reveal a defective v, and then we can forget about all positive pools that contain v, because v "explains" these positive pools, i.e., they do not supply the searcher with any further information.

Lemma 14 *Consider the family $\mathcal{F}$ of positive pools encountered in all stages prior to the last stage. Then the following holds for each candidate element v: If $\mathcal{F}$ has a hitting set D, such*

that $|D| < d$ and $v \notin D$, then v must be tested individually in the last stage. Consequently, if $\mathcal{F}$ has a hitting set D with less than d elements, then all candidate elements in the complement of D must be tested individually.

Proof. Let the adversary reveal a hitting set D of defectives, as specified. Then all positive pools are explained, and the searcher has to check if more defectives exist among the other candidate elements. Consider the pooling design used for this task in the last stage. Every candidate element $v \notin D$ must appear in some pool, since otherwise the searcher cannot decide the status of v. Furthermore, no pool Q may contain several elements, because, if Q is positive, the searcher cannot decide which element in Q is defective. Since no intersecting pools are allowed, the elements in a pool are not distinguishable. Thus, the singleton $\{v\}$ must be a pool, which was the claim. $\diamond$

The proof essentially uses that overlaps are prohibited, and, in fact, Lemma 14 does not hold in unrestricted group testing. Now we get already a reasonable lower bound. For $s = 1$, trivially n individual tests are needed, by arguments as in the proof of Lemma 14. In the following we always assume $s \geq 2$.

Theorem 15 *Any strategy for a fixed $s \geq 2$ and general d needs at least $(1/(s-1))^{1-1/s} \cdot sd(n/d)^{1/s} - s - d$ tests in the worst case.*

Proof. In all stages but the last one, the adversary sets the $d/(s-1)$ largest parts positive and also reveals one defective in every such part. Actually, a slightly smaller number of parts is selected, such that at most $d-1$ defectives are revealed in total. The set D of these defectives is a hitting set of the family of positive pools. Hence, by Lemma 14, the remaining candidate elements, except the at most $d-1$ elements of D, must be tested individually in stage s. As earlier, for a searcher playing against this adversary it is optimal to use parts of equal size within every stage. Again, let p_i denote the number of parts in stage i. In particular, p_s candidate elements still exist before stage s. Then we have for every $i < s$: In stage i, the number of candidate elements is multiplied by a factor $d/((s-1)p_i) < 1$. Hence the searcher must choose the p_i such that $n(d/(s-1))^{s-1}/(p_1 \cdot \ldots \cdot p_{s-1}) \leq p_s$, and she wants to minimize $p_1 + \ldots + p_s$. The constraint can be written as $n(d/(s-1))^{s-1} \leq p_1 \cdot \ldots \cdot p_s$. By routine calculations, in the optimal choice the inequality becomes an equation, and all p_i become equal to some p, giving $n(d/(s-1))^{s-1} = p^s$ and $sp = (1/(s-1))^{1-1/s} \cdot sd(n/d)^{1/s}$. $\diamond$

Note that the bound in Theorem 15 is already tight when $s = 2$, subject to an additive gap that does not depend on n. For any $s > 2$ there remains a multiplicative gap of $(s-1)^{1-1/s}$ between the upper and lower bounds in Theorem 13 and 15. In the following we will narrow down this gap, using more sophisticated techniques.

3.4 An Almost Matching Lower Bound

Until now we have obtained tight lower bounds for $s = 1$ and $s = 2$ only. Next we introduce a structural argument that enables us to solve our problem to optimality also for $s = 3$, at least if $d \ll n^{1/3}$. Eventually we extend this result to any fixed s, but the main difficulty is the step from 2 to 3 stages. One of the new ideas we need is to defer the adversary's decision on the defectives. Another prerequisite is the following graph-theoretic lemma.

Lemma 16 *Let B be any bipartite graph with positive integer edge weights. Define the weight of a vertex v to be the sum of weights of the edges incident to v. Then B has a spanning forest T with (changed, positive, integer) edge weights such that every vertex has in T the same weight as in B.*

Proof. If B has no cycle then set $T := B$. Otherwise let C be any cycle of edges in B. We will change the edge weights by integer amounts and destroy at least one edge in C, without changing the vertex weights. Thus iterated application of this step will yield a spanning forest as claimed.

Let e be an edge in C with minimum weight, say w. We subtract w from the weight of e, add w to the weight of the next edge in C, and so on, that is, subtraction and addition are done alternatingly while traversing the cycle. Since C has an even number of edges, obviously this manipulation is well defined and preserves the vertex weights. Edge e, as well as any other possible edge whose weight has been reduced to 0, is removed from B. $\diamond$

In the following we will need that some number p of parts, all of equal size, have been used in stage 1. The adversary will make d' of them positive, where d' is some number with $d' \leq d$. (In more detail: She gives a positive answer either to d' pools, or to $d'-1$ pools and reveals that the rest contains a defective.) If all parts actually have size n/p, this results in d' positive parts of that size. If the sizes are unequal, obviously it is optimal for the adversary to make the d' largest parts positive, thus retaining more than nd'/p candidate elements. But from this we cannot simply conclude that equally sized parts are optimal for the searcher. (It might be true, there is just no simple argument.) The catch is that some of the d' largest parts may be smaller than n/p, and then it is not clear whether the situation is worse for the searcher, than a scenario with equal sizes. To overcome this difficulty we guarantee some minimum size which is only slightly smaller than n/p:

Lemma 17 *If $p > d'$ pools are used in stage 1, then the adversary can leave the searcher with d' disjoint positive parts, each containing at least $(1-d'/p)n/p$ elements.*

Proof. Let $x := (d'/p)(n/p)$. A part with at least $n/p - x$ elements is called large, otherwise it is called small. In every large part we mark $n/p-x$ of the elements, and in every small part we mark all elements. If at least d' parts are large, the statement of the Lemma is true. If the d'-th largest part is small, then so are the $p-d'$ smallest parts. Together they have at most $(p-d')(n/p-x)$ elements, hence the d' largest parts together have at least $d'n/p + (p-d')x$ elements. Since at most $d'(n/p-x)$ of them are marked, at least px elements in the d' largest parts are unmarked. Note that unmarked elements can occur only in large parts. For every large part, the adversary answers positively and also announces that some marked element in the part is defective. Thus the searcher has no knowledge about the status of the unmarked elements there. The adversary collects the at least px unmarked elements and groups them into new fictitious parts of size $n/p-x$. Thus she obtains $px/(n/p-x) > px/(n/p)$ fictitious large parts. In the worst case for the adversary, only one real part is large, and all (but one) large parts are fictitious. Still the adversary can make d' parts (real or fictitious ones) positive, and they have the claimed size $n/p - x = (1-d'/p)(n/p)$. $\diamond$

Remember that we sometimes silently ignore lower-order terms. We also adopt the hypergeometric group testing model; the lower bound for the strict model cannot be smaller.

Theorem 18 *Any strategy for d defectives and $s = 3$ stages needs at least $(1-o(1))3d'(n/d')^{1/3}$ tests in the worst case, where $d = o(n^{1/3})$ and $d' := d - 1$.*

Proof.

Outline: The plan of the proof is as follows. We construct an adversary that will force every 3-stage strategy to execute the claimed number of tests. Every 3-stage strategy corresponds to an edge-weighted bipartite graph representing stage 2, in a way that we specify below. We speak of a *forest strategy* if this bipartite graph is a forest. We will analyze the number c of remaining candidate elements after stage 2. It indicates, subject to lower-order terms, the number of tests that our adversary can impose on a considered strategy. We show that, for any strategy σ, there exists a forest strategy σ' with the same or smaller c. Finally we prove a lower bound on the test number of forest strategies, helped by the fact that forests have fewer edges than vertices. Due to the previous point, this lower bound then applies to all 3-stage strategies. Now we detail these steps.

Adversary: In stage 1, the searcher tests some number p of pools. Our adversary gives positive answers to $d-1$ parts with sizes guaranteed by Lemma 17. Let U_i, $i = 1, \ldots, d-1$, denote these positive parts, and let $V_1, \ldots, V_q$ be the parts used by the searcher in stage 2. Only in stage 2, our adversary reveals $d-1$ defectives, exactly one in every U_i. The adversary chooses these revealed defectives so as to maximize the total number c of elements in all V_j that contain revealed defectives. Then, the adversary gives positive answers to these parts V_j, and negative answers to all other parts.

First analysis steps: The revealed defectives form a $(d-1)$-elements hitting set of the family of positive parts after stage 2. Hence Lemma 14 applies. Since all elements in the positive parts V_j are candidate elements, it follows that at least $c-d$ tests are needed in stage 3, and we can establish a lower bound $p+q+c-d$. Note that the searcher has already fixed the number p of tests of stage 1. Thus, for any fixed number q of tests in stage 2, it suffices to consider such pooling designs (in stage 2) that minimize the maximum c that the adversary can enforce.

Feasible families and forest strategies: We call a family F of parts V_j *feasible* if there exist $d-1$ elements, one from each U_i, that intersect exactly the sets in F. (To motivate the definition, note that feasible families are exactly those that can be made positive by revealing $d-1$ defectives, as above.)

In the following we do not notationally distinguish between parts (sets of elements) and vertices representing those parts in a certain graph; this should not cause confusion. To any pooling design in stage 2 we assign a bipartite graph with vertex sets $U = \{U_1, \ldots, U_{d-1}\}$ and $V = \{V_1, \ldots, V_q\}$, where vertices U_i and V_j are joined by an edge of weight $|U_i \cap V_j|$ whenever $U_i \cap V_j \neq \emptyset$. Trivially, $|U_i| = \sum_{j=1}^{q} |U_i \cap V_j|$ holds for all i. Vertex weights are defined as in Lemma 16.

Conversely, to any bipartite graph on vertex sets $U = \{U_1, \ldots, U_{d-1}\}$ and $V = \{V_1, \ldots, V_q\}$, with integer edge weights, and vertex weights $|U_i|$ at all vertices U_i, we can assign a pooling design for stage 2 as follows. Simply put in $U_i \cap V_j$ as many elements as the given edge weight indicates; this defines V_j (which equals $\bigcup_{i=1}^{d-1}(U_i \cap V_j)$ for each j). Clearly, our edge-weighted bipartite graphs and pooling designs are therefore in one-to-one-correspondence up to isomorphisms, i.e., permutations of elements.

Now consider any pooling design in stage 2, and let B denote the assigned bipartite graph. Let T be a spanning forest of B as given by Lemma 16. Consider the pooling design assigned

to T. Since the weights of all U_i in T are the same as in B, this pooling design is, in fact, valid and may be used in stage 2 instead of B. Since T possesses only edges that also exist in B, no feasible families are newly created when we replace B with T. Finally, since the weights of all V_j in T are the same as in B, too, every feasible family in T comprises the same number of candidates as it would have in B. Together it follows that c does not increase. Thus, for proving our lower bound of the form $p+q+c-d$, it suffices to consider forest strategies. Next we observe that a forest strategy has at most $q+d-2 < q+d$ nonempty sets $U_i \cap V_j$, since these intersections correspond to the edges of a forest with $q+d-1$ vertices. In other words, all V_j, with fewer than d exceptions, are subsets of parts U_i from stage 1. (That is, a forest strategy is almost nested.)

Finishing the analysis: Since we are proving a lower bound, we can now make the adversary weaker and give away some candidate elements: Instead of maximizing c for the given pooling design from stages 1 and 2, let the adversary simply choose the largest $U_i \cap V_j$ for every i, and fix some defective there. Assume for a moment that $|U_i| = n/p$ for all i. Now we are able to express how many candidates the adversary can secure before stage 3. At best, the searcher can split the $d-1$ parts U_i of size n/p into, in total, $q+d$ smaller pieces, upon which the adversary selects the largest piece from each U_i. Define $q' := q+d$ for brevity. By routine extremal value calculations, the searcher achieves the minimum by using the same number $q'/(d-1)$ of equally sized pieces in every U_i, hence every piece has $n(d-1)/(pq')$ elements, which yields $c = n(d-1)^2/(pq')$ candidates.

The number of tests in all 3 stages is therefore at least $p + q' + n(d-1)^2/(pq')$, subject to lower-order terms. This expression is minimized if $p = q' = (d-1)^{1-1/3}n^{1/3}$, hence we can write $p = q+d = (d-1)^{1-1/3}n^{1/3} + o(n^{1/3})$. In a final step this also yields asymptotically $n(d-1)^2/(pq) = (d-1)^{1-1/3}n^{1/3}$. Actually the U_i may be smaller by a factor $(1-d'/p)$ due to Lemma 17, but since we assumed $d = o(n^{1/3})$, this does not affect the constant factor in the main term of the query complexity. This completes the proof. $\diamond$

Let us rephrase the key observation in this proof. In stage 2, the adversary can keep at least a $d'/(d'+q-1)$ fraction of the candidate elements. Actually we used that, in an edge-weighted bipartite graph with partite sets U and V of size d' and q, respectively (where the edge weights are numbers of candidate elements), some spanning forest exists where the edge weights are in general changed but the vertex weights are preserved. Then we have chosen the maximum-weight edge incident to each vertex in U and shown that these edges together hold the desired fraction of the weight. Note that, for the entire proof to work, the adversary has to select, for each of the d' vertices in U, exactly one incident edge. Also note that all vertex weights in U are equal. However, our weak adversary in Theorem 18 gives away many candidate elements, thus it should be possible to raise the guaranteed fraction of candidate elements. By studying examples we came up with the following conjecture. (For convenience let $k := d'$ and $m := q$.)

Claim. Let $B = (U, V; E)$ be a bipartite graph, $k := |U|$ and $m := |V|$ with $k < m$, and let $f : E \to R^+$ be a weight function on the edges, such that all $u \in U$ satisfy $\sum_{v:\, uv \in E} f(uv) = 1/k$. Due to the last property we call the weight function *U-balanced*. Define $\pi(v) := \sum_{u:\, uv \in E} f(uv)$; note that $\sum_{v \in V} \pi(v) = 1$. We will call any nonnegative function π on V with sum 1 a *distribution*, and say that π is *induced* by f. For a fixed distribution π, the weight of a set $S \subseteq V$ is naturally defined as $\pi(S) := \sum_{v \in S} \pi(v)$. – Then B has a matching M that covers some set $S \subseteq V$ of weight $\pi(S) \geq k/m$.

Clearly, the scaling factor $1/k$ is arbitrary. We remark that the U-balanced property is essential (when it is dropped, one easily finds counterexamples already for $k = 2$), and that k/m is optimal for trivial reasons.

The claim implies that the adversary can even keep a d'/q fraction of the candidate elements: She takes the matching M, fixes one defective in every corresponding set $U_i \cap V_j$, and gives positive answers to all these V_j, hence all elements therein remain candidates. Since M is a matching, we get that only $|M| \leq d'$ defectives are fixed, and $d' - |M|$ further defectives can be fixed in order to hit all U_i (which have to be positive parts!).

In fact, we can prove the claim under mild assumptions: $m \geq 2k-1$ is sufficient. Since this proof is much more complex and uses matroid theory, we give it separately in the Appendix. Using a fraction d'/q instead of $d'/(d'+q-1)$ avoids the loss of d tests in our lower-bound analysis. The resulting improvement of the lower bound is minor when $d = o(n^{1/3})$, as considered in Theorem 18 (also, the technical issue addressed in Lemma 17 is still in place), however, it becomes more significant as d grows. We also believe that this graph-theoretic result is structurally interesting in itself, and it removes one obstacle on the way to a future extension of the $(1 - o(1))3d'(n/d')^{1/3}$ lower bound to larger d. We must leave the latter as an open problem.

We conclude this section with the generalization to any fixed s. No further technical ideas are needed, we only have to combine the earlier ingredients.

Theorem 19 *Any strategy for d defectives and s stages needs at least $(1 - o(1))sd'(n/d')^{1/s}$ tests in the worst case, where $d = o(n^{1/s})$ and $d' := d - 1$.*

Proof. In stage 1, the adversary generates disjoint positive sets $U_1^1, \ldots, U_{d-1}^1$ as in Lemma 17. The method in the proof of Theorem 18 shows that, in every stage $j = 2, \ldots, s-1$, the searcher can minimize the number of remaining candidate elements by applying a forest strategy, yet the adversary can keep the number of candidates as large as it would be in a nested strategy with only d additional tests per stage, and these candidates form disjoint positive sets $U_1^j, \ldots, U_{d-1}^j$, with $U_i^j \subset U_i^{j-1}$ for all i. Eventually, the searcher has to test all but $d-1$ candidates in $U_1^{s-1} \cup \ldots \cup U_{d-1}^{s-1}$ individually in stage s. Again, straightforward extremal value calculation shows that now the total number of tests is minimized if the searcher uses the same number of tests in every stage, and parts of equal size within every stage. Thus the test number is, subject to lower-order terms, the same as in an optimal nested strategy, as calculated earlier. $\diamond$

We conjecture that our test number is asymptotically optimal for all ranges of parameters n, d, s, and merely our proof techniques are still too weak, see the discussion above.

3.5 When the Number of Defectives is Unknown

So far we assumed that the number d of defectives, or an upper bound on d, is known in advance. If we have no clue how large d could be, we can first estimate d in stage 1, and then append our optimal strategy with $s-1$ stages. However we need randomization to estimate d. Again we neglect lower-order terms for simplicity.

Theorem 20 *We can identify d defectives, where d is not known beforehand, using $t := (s-1)d(n/d)^{1/(s-1)}$ tests. If d happens to be small compared to $n^{1/s}$, then $(s-1)s^{1/s}dn^{1/s}$*

tests are sufficient. (We can either find all defectives for sure, with an expected number t of tests, or find them with high probability using t tests.)

Proof. We will argue below that we can estimate d in one stage using $o(n^{1/s})$ randomized tests, with a multiplicative error that becomes, with arbitrarily high probability, arbitrarily small when $n \to \infty$. Then, the number of tests in stage 1 is negligible compared to the tests needed to actually identify the defectives. To this end we apply the strategy from Theorem 13 in stages $2, \ldots, s$, with the exception that we need not stop if more than the estimated number of defectives are present. Note that this strategy still works correctly even in the case of a false d, as the input parameter d only affects the number of tests.

For stage 1 we apply a random permutation to the n elements[1] and then divide them into p disjoint pools, such that the i-th pool has size $i^{s-1+\epsilon}$, with an arbitrarily small $\epsilon > 0$. Since $n \approx \sum_{i=1}^{p} i^{s-1+\epsilon} \approx p^{s+\epsilon}/s$, we get $p = o(n^{1/s})$. Moreover, the pool sizes are below $(sn)^{1-1/s}$.

Since the pool sizes start with 1 and grow slowly (as a power function), in some prefix of the sequence only a minority of pools is positive, and the defectives are, with high probability, located in different pools. Thus we can easily estimate d/n by the number of positive pools divided by the number of elements in that prefix. The larger n is, the larger is the prefix we can use, and the more accurate is the estimate of d. (Since we only state the asymptotic result, no analysis of the variance of the estimator is needed here; it suffices to notice that it tends to 0 when $n \to \infty$.)

Theorem 13 applied with $s - 1$ instead of s stages gives the test complexity. If d is small compared to $n^{1/s}$, then even the largest pool size $s^{1-1/s} n^{1-1/s}$ is small compared to n/d. More importantly, the number of candidates after stage 1 is then bounded by $d(sn)^{1-1/s}$. Thus we can replace n/d in Theorem 13 with $(sn)^{1-1/s}$ and obtain the claimed test complexity. $\diamond$

The multiplicative error of our estimator gets arbitrarily small as n grows. For moderate problem sizes, however, the variance of the estimate of d cannot be neglected, so that more analysis, theoretically or experimentally, is needed. Next, it would be interesting to figure out the price of ignorance of d, that is, the optimal competitive ratio. Note that, compared to the case of known d, essentially one stage is lost. The competitive ratio is well investigated for classical adaptive group testing [29]. For the present problem we believe that it cannot be constant but must be some function of parameters d and s.

Acknowledgments

The work of the first two authors has been supported by the Swedish Research Council (Vetenskapsrådet), through grant 2010-4661, "Generalized and fast search strategies for parameterized problems". The first author also received support from RWTH Aachen during a visit in 2011. We thank the referees of ICALP2001GT and of this Special Issue for their careful remarks and suggestions.

[1] Note that this strategy is deterministic up to random permutation, however we do not need optimality as in Theorem 2.

References

[1] Aigner, M.: Combinatorial Search. Wiley-Teubner (1988)

[2] Balding, D.J., Torney, D.C.: Optimal pooling designs with error detection. J. Comb. Theory A 74, 131–140 (1996)

[3] Bar-Lev, S.K, Boneh, A., Perry, D.: Incomplete identification models for group-testable items. Naval Res. Logistics 37, 647–659 (1990)

[4] Cheng, Y., Du, D.Z.: New constructions of one- and two-stage pooling designs. J. Comp. Biol. 15, 195–205 (2008)

[5] Cicalese, F., Damaschke P., Vaccaro, U.: Optimal group testing strategies with interval queries and their application to splice site detection. Int. J. Bioinf. Res. Appl. 1, 363–388 (2005)

[6] Cicalese, F., Damaschke P., Tansini, L., Werth, S.: Overlaps help: Improved bounds for group testing with interval queries. Discr. Appl. Math. 155, 288–299 (2007)

[7] Clements, G.F.: The minimal number of basic elements in a multiset antichain. J. Comb. Theory A 25, 153–162 (1978)

[8] Colbourn, C.J., Dinitz, J.H.: The CRC Handbook of Combinatorial Designs. CRC Press (1996)

[9] Dachman-Soled, D., Servedio, R.: A canonical form for testing Boolean function properties. In: Goldberg, L.A., Jansen, K., Ravi, R., Rolim, J.D.P. (eds.) Approximation, Randomization, and Combinatorial Optimization. Algorithms and Techniques APPROX-RANDOM 2011. LNCS, vol. 6845, pp. 460–471. Springer, Heidelberg (2011)

[10] Damaschke, P., Sheikh Muhammad, A.: Randomized group testing both query-optimal and minimal adaptive. In: Bieliková, M., Friedrich, G., Gottlob, G., Katzenbeisser, S., Turán, G. (eds.) 38th International Conference on Current Trends in Theory and Practice of Computer Science SOFSEM 2012. LNCS, vol. 7147, pp. 214–225. Springer, Heidelberg (2012)

[11] De Bonis, A., Di Crescenco, G.: Combinatorial group testing for corruption localizing hashing. In: Fu, B., Du, D.Z. (eds.) Computing and Combinatorics COCOON 2011. LNCS vol. 6842, pp. 579–591. Springer, Heidelberg (2011)

[12] De Bonis, A., Gasieniec, L., Vaccaro, U.: Optimal two-stage algorithms for group testing problems. SIAM J. Comp. 34, 1253–1270 (2005)

[13] Du, D.Z., Hwang, F.K.: Combinatorial Group Testing and Its Applications. Series on Appl. Math. vol. 18. World Scientific (2000)

[14] Du, D.Z., Hwang, F.K.: Pooling Designs and Nonadaptive Group Testing. Series on Appl. Math. vol. 18. World Scientific (2006)

[15] Dyachkov, A.G., Rykov, V.V.: Bounds on the length of disjunctive codes. Problems of Info. Transmission (in Russian) 18, 7–13 (1982)

[16] Edmonds, J.: Matroids and the greedy algorithm. Math. Progr. 1, 127–136 (1971)

[17] Eppstein, D., Goodrich, M.T., Hirschberg, D.S.: Improved combinatorial group testing algorithms for real-world problem sizes. SIAM J. Comp. 36, 1360–1375 (2007)

[18] Fang, J., Jiang, Z.L., Yiu, S.M., Hui, L.C.K.: An efficient scheme for hard disk integrity check in digital forensics by hashing with combinatorial group testing. Int. J. Digital Content Technol. Appl. 5, 300–308 (2011)

[19] Fischer, P., Klasner, N., Wegener, I.: On the cut-off point for combinatorial group testing. Discr. Appl. Math. 91, 83–92 (1999)

[20] Goldreich, O., Trevisan, L.: Three theorems regarding testing graph properties. Random Structures and Algor. 23, 23–57 (2003)

[21] Goodrich, M.T., Hirschberg, D.S.: Improved adaptive group testing algorithms with applications to multiple access channels and dead sensor diagnosis. J. Comb. Optim. 15, 95–121 (2008)

[22] Huffman, D.A.: A method for the construction of minimum-redundancy codes. Proc. IRE 40, 1098–1101 (1952)

[23] Huang, S.H., Hwang, F.K.: When is individual testing optimal for nonadaptive group testing? SIAM J. Discr. Math. 14, 540–548 (2001)

[24] Li, C.H.: A sequential method for screening experimental variables, J. Amer. Statist. Assoc. 57, 455–477 (1962).

[25] Lubell, D.: A short proof of Sperner's lemma. J. Comb. Theory 1, 299 (1966)

[26] Mézard, M., Toninelli, C.: Group testing with random pools: Optimal two-stage algorithms, IEEE Trans. Info. Theory 57, 1736–1745 (2011)

[27] Ruszinkó, M.: On the upper bound of the size of the r-cover-free families. J. Combin. Theory A 66, 302–310 (1994)

[28] Sperner, E.: Ein Satz über Untermengen einer endlichen Menge (in German). Math. Zeitschrift 27, 544–548 (1928)

[29] Schlaghoff, J. Triesch, E.: Improved results for competitive group testing. Comb. Prob. and Comput. 14, 191–202 (2005)

[30] Welsh, D.J.A.: Matroid Theory. Academic Press (1976)

Appendix

Interpretation of Group Testing as a Game

Here we give a more formal presentation of the game-theoretic setting we use for proving lower bounds. In the group testing problem we are searching for some unknown subset P of an n-element set I of elements, and we can choose pools $Q \subset I$ in order to test whether $Q \cap P$ is empty or not.

A deterministic s-stage group testing strategy A is a function which successively selects sets $T_i := \{Q_{i,1}, \ldots, Q_{i,t_i}\}$ of pools, $1 \leq i \leq s$. Given some subset $P \subset I$, the result of test stage i is given by the vector $r_i := (r_{i,1}, \ldots, r_{i,t_i})$, where $r_{i,j} := 0$ if $Q_{i.1} \cap P = \emptyset$, and $r_{i,j} := 1$ else. The strategy may use the results of stages $1, \ldots, i-1$ when selecting T_i, hence T_i may depend on P if $i > 1$. The cost of stage i is the number of tests, $t_i = t_i(P)$.

A strategy A solves the strict (d, n) group testing problem if, for each $P \subset I$, $|P| \leq d$, the sequence of results r_i, $1 \leq i \leq s$, uniquely determines P as a subset of I. (In particular, if the strategy is applied to a set P of cardinality larger than d, the test results show that $|P| > d$).

We denote by $\mathcal{A}$ the finite set of all deterministic s-stage strategies solving the strict (d, n) group testing problem. Moreover, we denote by $t(A, P) := \sum_{i=1}^{s} t_i(P)$ the cost of A, given P, and by $t(A) := \max\{t(A, P) : P \subset I, |P| \leq d\}$, the worst-case cost of strategy A. Now, $t(n, d, s) := \min_{A \in \mathcal{A}} t(A)$.

An adversary strategy S is a function which, given a sequence r_j, $1 \leq j \leq i-1$, of possible results for an s-stage group testing algorithm and some test set T_i of pools, chooses some vector $r_i = S(r_1, \ldots, r_{i-1}, T_i)$ of results for the tests in T_i. In what follows, we are considering adversary strategies generating results which are consistent with at least one subset $P \subset I$ and denote the (finite) set of all such strategies by $\mathcal{S}$.

Given some adversary strategy $S \in \mathcal{S}$ and an s-stage group testing strategy $A \in \mathcal{A}$, we define $t(A, S)$ as the total number of tests used if A is applied to the results generated by S, while S is applied to the test sets generated by A.

The number $t(A, S)$ can be interpreted as the total number of tests used in a game between two players, the searcher and the hider. The searcher wants to find P, and the tests she uses are given by algorithm A. The hider provides results for the tests by using her strategy S. The number $t(S) := \min_{\mathcal{A}} t(A, S)$ denotes the worst-case cost that the hider can enforce by using strategy S.

Clearly, we have $t(A) = \max_{S \in \mathcal{S}} t(A, S)$, hence $\min_{\mathcal{A}} t(A) \geq \max_{\mathcal{S}} t(S)$. In fact, it is well known that equality holds:

$$\min_{\mathcal{A}} \max_{\mathcal{S}} t(A, S) = \max_{\mathcal{S}} \min_{\mathcal{A}} t(A, S) = t(n, d, s).$$

This can be proved by induction on s by slightly modifying the proof of Proposition 1.31 in [1] and is a special case of the Minimax Theorem of Game Theory. Hence $t(S)$ is a lower bound for $t(n, d, s)$ for each adversary strategy S.

In our arguments, we also use a variant of the strategies discussed above: Imagine that, in stage i, the pool set T_i is tested. Instead of providing results for the tests $Q_{i,j}$ only, the hider might be allowed to reveal more information for free, that is, add positive or negative test sets $Q_{i,j}$ to T_i, which are not counted as tests in computing $t(A, S)$. (Of course, the answers must remain consistent with some set P). For a strategy S in this generalized sense, the inequality $t(S) \leq t(n, d, s)$ holds *a fortiori*.

A randomized strategy is now given by a probability distribution π on $\mathcal{A}$. Its expected costs are defined as $E[\pi] := \sum_{A \in \mathcal{A}} \pi(A)t(A)$. With similar definitions for randomized adversary strategies τ as probability distributions on $\mathcal{S}$, the Minimax Theorem of Game Theory yields:

$$\min_{\pi} E[\pi] = max_{\tau} E[\tau].$$

Extremal Matchings in Balanced Weighted Bipartite Graphs

In this section we prove the Claim from Section 3.4, provided that $m \geq 2k-1$ (equivalently, $d \leq (q+3)/2$). It was already stated and motivated there, such that we can now focus on the proof. We presume familiarity with the notion of a matroid, otherwise we refer to [30].

The *transversal matroid* of a bipartite graph $B = (U, V; E)$ has as *independent sets* exactly those sets $S \subseteq V$ for which there exists a matching that covers S. It is well known that, in any matroid, an independent set, also called a *basis*, of maximum weight can be computed by a *greedy algorithm* due to [16]. It works as follows in our case of a transversal matroid, for a given $B = (U, V; E)$ and π: Sort the vertices in V by non-ascending weights $\pi(v)$, and start with $S := \emptyset$. Scan this sorted list and always put v into the basis S if and only if the resulting set $S := S \cup \{v\}$ remains independent, otherwise let S unchanged. (The greedy algorithm has to call an "oracle" that checks whether some matching covers S.) We denote by $g(\pi)$ the weight $\pi(S)$ of a maximum-weight basis S.

We suppose in the following that B possesses a matching of size k, that is, a matching that covers U. Then we have $|S| = k$, since in a matroid all maximal independent sets have the same cardinality.

Let Δ denote the set of all distributions π on V. Since Δ is compact and g, with argument $\pi \in \Delta$, is a continuous function on Δ, the minimum $\min_{\pi \in \Delta} g(\pi)$ exists. Denote by Δ^* the set of all distributions $\pi^* \in \Delta$ where $g(\pi^*)$ attains this minimum. Δ^* is compact as well.

In the following we adopt the convention $x \log_2 x = 0$ for $x = 0$. The *entropy function* $h(\pi) := -\sum_{v \in V} \pi(v) \log_2 \pi(v)$ is continuous, in particular on Δ^*. Hence some distribution $\sigma \in \Delta^*$ maximizes the entropy. We claim that $g(\sigma) \geq k/m$. Of course, the Claim then follows for all distributions from Δ^* and hence from Δ, and then the k/m lower bound is proved for any graph B as specified above. We just need the entropy as an auxiliary measure to prove this Claim. A basic property of the entropy $h(\pi)$ is that it increases if we replace two items in π, say x and y such that $x < y$, with $x + \epsilon < y - \epsilon$, where $\epsilon > 0$.

When applying the greedy algorithm to σ, we may assume that the vertices of V are sorted in such a way that, for every existing value a of vertex weights $\sigma(v)$, those vertices v from $\{v \in V : \sigma(v) = a\}$ that enter the basis S appear before the vertices of equal weight that do not enter the basis. Thus we obtain the following structure. We can split S into $S = S_1 \cup S_2 \cup \ldots \cup S_r$, such that all vertices in each S_i have the same weight, and the weights in each S_i are strictly larger than the weights in S_{i+1}. Furthermore we can split $L := V \setminus S$ into $L = L_1 \cup L_2 \cup \ldots \cup L_r$, such that $S_1, L_1, S_2, L_2, \ldots, S_r, L_r$ is a partition of the whole V sorted by non-ascending weights, into consecutive subsets, and the weights in each L_i are strictly larger than the weights in S_{i+1}. Note that some L_i may be empty.

We also fix some matching that covers S (which exists since S is a basis) and divide U into $U = W_1 \cup W_2 \cup \ldots \cup W_r$, where W_i comprises the matching partners of vertices in S_i.

Lemma 21 *The vertices in each $S_j \cup L_j$ have all their neighbors in $W_1 \cup \ldots \cup W_j$.*

Proof. Any edge incident to a vertex in $v \in L_j$ has its second vertex in some W_i with $i \leq j$, since otherwise the greedy algorithm would have added v to the basis. The main work is to prove the Lemma also for S_j.

Assume that an edge uv exists with $u \in W_i$, $v \in S_j$, and $i > j$. Let $w \in S_i$ be the matching partner of u. We increase $f(uw)$ and decrease $f(uv)$ by some $\epsilon > 0$; we will specify below how small it has to be. Let σ' be the resulting distribution on V. Clearly, the change does not affect the sum of weights of the edges incident with $u \in U$, and we get $\sigma'(w) = \sigma(w) + \epsilon$ and $\sigma'(v) = \sigma(v) - \epsilon$. Moreover, σ' has a higher entropy than σ.

Since all σ weights in S_i were equal, we can assume w to be the first vertex in S_i, and since the σ weights in $S_{i-1} \cup L_{i-1}$ were strictly larger, the position of w in the order of σ' weights remains the same as in σ, for a small enough ϵ. Similarly we can assume that v is the last vertex in S_j. If v keeps its position as well, then the greedy algorithm returns the same basis containing both v and w, hence $g(\sigma') = g(\sigma)$. Now assume that $g(\sigma') > g(\sigma)$. Then there must be some $v' \in L_j$ with $\sigma(v') = \sigma(v)$ that replaces v in the greedy basis (otherwise the greedy basis cannot change any more). That means, $S_1 \cup \ldots \cup S_j \cup \{v'\} \setminus \{v\}$ is an independent set in the transversal matroid of B. Accordingly, let M' be a matching that covers $S_1 \cup \ldots \cup S_j \cup \{v'\} \setminus \{v\}$. Let M be a matching that covers $S_1 \cup \ldots \cup S_j$ and maps this set to $U = W_1 \cup \ldots \cup W_j$. Note that M exists, and u does not belong to any edge in M.

Consider the alternating path A in $M' \cup M$ that starts in v' with an edge of M'. If A ends in U, then we can use the M'-edges in A to cover v' and those vertices of $S_1 \cup \ldots \cup S_j$ that appear in A. The other vertices of $S_1 \cup \ldots \cup S_j$ are still covered by their M-edges. If A ends in V, then the last vertex of A is necessarily v, since otherwise we could continue A with another M'-edge. But now we can append the edge vu to A instead, since u is not involved in any M-edge. We are back to the previous situation and can cover v' and those vertices of $S_1 \cup \ldots \cup S_j$ that appear in A, using M'-edges and the extra edge vu. Finally, A cannot be a cycle, i.e., return to v', since v' is not incident with any M-edge. In either case we found a matching that covers $S_1 \cup \ldots \cup S_j \cup \{v'\}$. But this contradicts the fact that the greedy algorithm applied to the original σ weights did not put v' in the basis.

This contradiction disproves the assumption $g(\sigma') > g(\sigma)$. Hence we always obtain $g(\sigma') = g(\sigma)$, since $g(\sigma)$ was minimal. Thus we have $\sigma' \in \Delta^*$, contradicting the choice of σ as the distribution with maximum entropy in Δ^*. This contradiction, in turn, shows that the assumed edge uv with $u \in W_i$, $v \in S_j$, and $i > j$ cannot exist, which completes the proof. $\diamond$

Define $k_i := |S_i|$, and let $\sigma(X)$ denote the weight of any subset $X \subseteq V$. Consider any prefix $S_1, L_1, \ldots, S_i, L_i$ of our sequence $S_1, L_1, \ldots, S_r, L_r$. The edges incident to the $k_1 + \ldots + k_i$ vertices in $W_1 \cup \ldots \cup W_i$ have together the weight $(k_1 + \ldots + k_i)/k$, and the vertices in $S_1 \cup L_1 \cup \ldots \cup S_i \cup L_i$ get their weights from such edges only, due to Lemma 21. It follows $\sigma(S_1 \cup L_1 \cup \ldots \cup S_i \cup L_i) \leq (k_1 + \ldots + k_i)/k$. Recall that $k = |S|$, $m = |V|$, and we want to prove $\sigma(S) \geq k/m$.

At this point we can forget about the graph and consider $S_1, L_1, \ldots, S_r, L_r$ as a sequence of sets of weighted items which enjoys the following properties:
(i) All weights in each S_i are equal.
(ii) All weights in S_i are at least as large as all weights in L_i.
(iii) All weights in S_i are at least as large as all weights in S_{i+1}.
(iv) All prefixes satisfy $\sigma(S_1 \cup L_1 \cup \ldots \cup S_i \cup L_i)/\sigma(V) \leq (k_1 + \ldots + k_i)/k$.

Our claim is $\sigma(S)/\sigma(V) \geq k/m$ for such sequences. From this, the original claim for the

graph would follow. Note that we have ignored some stronger properties that are no longer needed: The entire sequence of weights is not necessarily monotone, and $\sigma(V) = 1$ is not required. (Obviously, ratios do not change if we multiply all weights with a common scaling factor.) Now we can proceed without much calculation, just using monotonicity arguments.

Lemma 22 *$\sigma(S)/\sigma(V) \geq k/m$ holds for sequences that satisfy (i)-(iv).*

Proof. Let V' be any prefix, $S' = S \cap V'$, and $k' = |S'|$. By property (iv), $\sigma(V')$ is below average, and by monotonicity property (iii), $\sigma(S')$ is above average (for a selection of sets S_i with totally k' vertices, together with their L_i). Hence $\sigma(S')/\sigma(V') \geq \sigma(S)/\sigma(V)$.

Now we multiply all weights in V' with a scaling factor smaller than 1, until the weights in the last S_i in V' equal those in S_{i+1}. Scaling down the prefix obviously does not affect (i) and (ii), moreover we have respected (iii), and (iv) remains true because diminishing some prefix reduces the relative weight of every prefix. Hence the invariants (i)–(iv) are preserved.

Moreover, $\sigma(S')/\sigma(V') \geq \sigma(S)/\sigma(V)$ ensures that the new ratio $\sigma(S)/\sigma(V)$ after the scaling can only get smaller, as we have reduced the relative weight of V' in V. Thus, by repeating this manipulation we can equalize all weights in S without increasing $\sigma(S)/\sigma(V)$. But once all items in S have reached the same weight, $\sigma(S)/\sigma(V) \geq k/m$ is true because of property (ii). $\diamond$

Altogether we can conclude:

Theorem 23 *Let $B = (U, V; E)$ be any bipartite graph which is U-balanced with respect to the edge weight function f with sum 1, let π be the distribution on V induced by f, and $k = |U|$, $m = |V|$. If B has a matching of size k then B also has a matching that covers some $S \subseteq V$ with $\pi(S) \geq k/m$.* $\diamond$

The additional assumption that B has a matching of size k is not that satisfactory, but finally we prove the statement with a precondition only on the sizes of the partite sets of B.

Theorem 24 *Let $B = (U, V; E)$ be any bipartite graph which is U-balanced with respect to the edge weight function f with sum 1, let π be the distribution on V induced by f, and $k = |U|$, $m = |V|$. If $m \geq 2k - 1$ then B has a matching that covers some $S \subseteq V$ with $\pi(S) \geq k/m$.*

Proof. Let $\mu < k$ be the size of a maximum matching. There exists $U_0 \subset U$, such that the induced subgraph $B[U_0 \cup V]$ does have a matching that covers U_0, and U_0 is a maximal set with this property. In particular, $|U_0| = \mu$. The total weight of edges in $B[U_0 \cup V]$ is μ/k. By Theorem 23, V contains an independent set S, in the transversal matroid of $B[U_0 \cup V]$, with weight at least $(\mu/m)(\mu/k) = \mu^2/(mk)$. Since U_0 was maximal, the $k - \mu$ vertices in $U \setminus U_0$ have all their neighbors in S. Thus, in the entire graph B, the basis S has a weight at least $(k - \mu)/k + \mu^2/(mk) = ((k - \mu)m + \mu^2)/(mk)$. Observe that $((k - \mu)m + \mu^2)/(mk) < k/m$ would imply $(k - \mu)m + \mu^2 < k^2$, thus $(k - \mu)m < k^2 - \mu^2$ and $m < k + \mu$. Consequently, S has weight at least k/m provided that $m \geq k + \mu$, in particular if $m \geq 2k - 1$. $\diamond$