Improved Upper Bounds for Several Variants of Group Testing

Von der Fakultät für Mathematik, Informatik und Naturwissenschaften
der Rheinisch-Westfälischen Technischen Hochschule Aachen
zur Erlangung des akademischen Grades eines Doktors
der Naturwissenschaften genehmigte Dissertation

vorgelegt von

Diplom-Mathematiker
Andreas Allemann
aus Bonn

Berichter: Universitätsprofessor Dr. Eberhard Triesch
Professor Dr. Lutz Volkmann

Tag der mündlichen Prüfung: 11. November 2003

Contents

Introduction

Group testing is a class of search problems, in which we typically have a set of n items, each of which is either good or defective. A test on an arbitrary group (subset) of items reveals either that all items in the group are good or that at least one of the items is defective, but not how many or which items are defective. The aim is to identify all items as either good or defective using as few tests as possible. We focus on the *sequential* case, in which the results of preceding tests may be used to determine the next test group. The *nonadaptive* case, in which all test groups have to be specified in advance, is a very different problem with few connections to the sequential case. We distinguish between *probabilistic* (average case) and *combinatorial* (worst case) group testing problems. In the former, we try to minimize the expected number of tests under a given probability distribution of the defectives. In the latter, the aim is to minimize the maximum number of tests required.

In this thesis, $\log x$ always denotes $\log_2 x$. Let $\lceil x \rceil$ ($\lfloor x \rfloor$) denote the smallest (largest) integer greater (less) than or equal to x.

The history of group testing begins with the probabilistic problem assuming that each item is defective with some probability p independent of all other items, also called the *binomial group testing* problem. In 1943, Dorfman [5] was the first to study group testing, motivated by the need to screen large populations for infectious diseases economically by testing pools of several blood samples for antibodies. A detailed account of the early history of group testing was given by Du and Hwang [6, Section 1.1].

Motivated by applications in industrial testing, Sobel and Groll [19] introduced a restricted class of algorithms called the *nested class*: if a group of at least two items is contaminated, that is, known to contain a defective, a nested algorithm always uses a subset of this group as the next test group. Sobel and Groll [19] gave a recursive definition of an optimal nested algorithm for the binomial group testing problem with parameter p, which was simplified later by Hwang [12]. Sobel [18] improved this algorithm by using special procedures involving overlapping test groups for contaminated groups containing 2 or 3 items. Chen, Hsu, and Sobel [3], see also Hsu [9], gave a construction based on choosing the next test group by one of two approaches: either trying to achieve a probability of a positive test result

as close to $\frac{1}{2}$ as possible or using the Huffman lower bound as a proxy for the expected number of remaining tests. They showed that the resulting algorithm is as good as or slightly better than Sobel's algorithm for selected values of p and $n \leq 6$. However, the optimal solution for binomial group testing remains unknown.

For the general average case with an arbitrary probability distribution on the defectives, Triesch [20] gave a construction and a cost estimate of a very good nested algorithm.

In combinatorial group testing, it is typically assumed that there are exactly d defectives among the n items, and we try to minimize the worst case number of tests. This is called the (d, n) *group testing* problem. It was first studied by Li [15] to determine which out of a large number of variables significantly influence the outcome of an experiment. As there are $\binom{n}{d}$ possible sets of defectives and in t tests at most 2^t cases can be differentiated, at least $\lceil \log \binom{n}{d} \rceil$ tests are needed. This is called the *information lower bound*. The best upper bound known in the literature was proven by Hwang [11] for his *generalized binary splitting algorithm*, which tests groups of size 2^a where the choice of the integer a depends on the proportion of defectives among the not yet identified items; whenever a test result is positive, a defective is identified by repeatedly halving the contaminated group. The generalized binary splitting algorithm belongs to the nested class, and the number of tests it needs exceeds the information lower bound at most by $d - 1$. Du and Hwang [6, Section 2.5] gave the recursive definition of an optimal nested algorithm.

In practical applications the exact number of defectives is hardly ever known. The case that d is an upper bound of the number of defectives was considered first by Hwang [11] and is called the *generalized* (d, n) *group testing* problem. Hwang, Song, and Du [13] showed that this problem can be solved with at most one additional test compared to the (d, n) group testing problem.

For the case that nothing is known about the number of defectives, Du and Hwang [7] formulated the *competitive group testing* problem. A competitive algorithm requires for every number of items n and number of defectives d at most a fixed multiple, called *competitive ratio*, of the number of tests used by the optimal algorithm for the (d, n) group testing problem plus a constant. The lowest competitive ratio achieved so far is $1.5 + \varepsilon$, for which Schlaghoff and Triesch [17] gave competitive algorithms for all positive ε.

Many variations of these group testing problems and numerous applications in a wide range of fields have been examined in the literature. Du and Hwang [6] give a comprehensive overview of the combinatorial side of group testing in their book.

In the first chapter of this thesis, we introduce a novel representation of the information obtained by group tests. The representation uses hypergraphs, following an idea of Triesch. We examine the cases in which the

number of defectives is unknown and in which lower and upper bounds for the number of defectives are known. For both cases, we analyse under which circumstances individual items are identified as either good or defective and under which conditions all items have been identified. The analysis applies to all group testing problems mentioned above.

In the second chapter, we discuss a new variant of combinatorial group testing, which we call the *complete group testing* problem: the number of defectives among n items is unknown and we try to minimize the worst case number of tests needed if there happen to be at most d defectives. This is similar to the generalized (d, n) group testing problem, in which it is known that there are at most d defectives, with the crucial difference that the case of more than d defectives is not excluded but has to be detected as well. This avoids the unfortunate property of the (d, n) and generalized (d, n) group testing problems that algorithms designed for these problems typically miss defectives or may even report good items as defective if more than d items happen to be defective in a practical application. We prove that the optimal algorithm for the complete group testing problem needs exactly one additional test compared to the optimal algorithm for the (d, n) problem. As the proof is constructive, it provides explicit instructions to transform algorithms between these two problems.

The third chapter contains the major result of this thesis, a new algorithm for combinatorial group testing. The *split and overlap algorithm* is defined for the complete group testing problem introduced in the second chapter, but can easily be adapted to the (d, n) or generalized (d, n) group testing problem by simply omitting all tests that become predictable due to the additional information about the number of defectives. In the beginning, the initial test group size m is chosen depending on the ratio $\frac{n}{d}$. The algorithm then repeatedly tests groups of size m and whenever a test result is positive at least one defective is identified in the contaminated group of size m by splitting it repeatedly, similarly to nested algorithms. However, unlike in nested algorithms, complex subalgorithms involving overlapping test groups are used on contaminated groups of certain sizes. These subalgorithms make it possible to deal efficiently with arbitrary test group sizes m, not just with powers of two, as in Hwang's generalized binary splitting algorithm.

The following estimates for the number of tests required by the split and overlap algorithm all refer to its application to the (d, n) group testing problem. We demonstrate that for the initial test group size m, the algorithm needs at most $\frac{1}{m}$ tests for each good item and a constant number of tests per defective identified plus 4 tests. This is less than $0.255d + \frac{1}{2}\log d + 5.5$ tests above the information lower bound $\left\lceil \log \binom{n}{d} \right\rceil$ for $\frac{n}{d} \geq 2$. For $\frac{n}{d} \geq 38$, the difference decreases to less than $0.187d + \frac{1}{2}\log d + 5.5$ tests. For $d \geq 10$, this is a considerable improvement over the $d - 1$ tests given by Hwang [11] for his generalized binary splitting algorithm. We conjecture that the behaviour

for large n and d of the difference between the number of tests required by the split and overlap algorithm and the information lower bound is optimal for $\frac{n}{d} \leq 4$. This implies that the $\frac{1}{2} - \log \frac{32}{27} = 0.255$ tests per defective given in the bound above are the best possible. Interestingly, leaving the nested class and using overlapping test groups seems to yield much bigger gains in combinatorial than in probabilistic group testing.

This thesis was typeset in LaTeX, using pdfTeX to generate portable output and the convenient graphical front end Scientific Workplace. All figures were produced with MetaPost, a close relative of the font generator MetaFont, which generates PostScript images from scripts and includes support for TeX expressions and cubic splines in PostScript output. The meta algorithm to find the best group testing algorithms was written in the advanced functional programming language Haskell, allowing me to concentrate on the algorithmic instead of the technical complexities and resulting in a small fraction of the source code size of a comparable Java program. I appreciate the efforts of the authors of all these excellent tools.

I wish to thank Prof. Dr. Eberhard Triesch for many fruitful discussions and helpful hints as well as for having introduced me to the fascinating area of group testing. I am particularly grateful to my wife Doris Allemann-Bulea for her steady support and for coping together with the strains and stresses inevitably linked to the writing of this thesis.

Chapter 1

On the Structure of Group Testing

This chapter contains an analysis of the precise circumstances under which items are identified as either good or defective in the common group testing model. To this end, we introduce a novel representation of the information obtained by group tests, which is used throughout the whole thesis.

1.1 The Group Testing Model

Consider a set I of n items each being either good or defective. The state of the items can be determined only by testing subsets of I. A test can yield a negative result, indicating that all tested items are good, or a positive result, indicating that the group is contaminated, that is, at least one of the tested items is defective. All tests are considered to be error-free.

The aim of group testing is to identify all defective items through a sequence of tests, thereby identifying all other items as good, using as few tests as possible. In the nonadaptive case all tests have to be specified in advance. We consider only the sequential (adaptive) case, in which the test group can be specified depending on the results of all previous tests.

The set of all defectives among the n items is called the *defective set*. All possible defective sets constitute the *search domain* S. In the probabilistic setting, a probability distribution on S is given, and we aim to minimize the expected number of tests (average case). In the combinatorial setting, we aim to minimize the maximum number of tests (worst case). The results in this chapter apply to both settings.

In the basic group testing problem, all subsets of the set of items I may be tested. However, it is possible to introduce restrictions on the tests allowed; for example, the maximum test size can be limited. In the nested class introduced by Sobel and Groll [19], if a group with at least two items is known to be contaminated, then the next test has to be performed on a

subset of this group. These restrictions do not affect the analysis performed in this chapter.

Most group testing problems are symmetric with respect to the items, that is, swapping any two items before the first test does not make a difference. Thus, if the search domain S contains a defective set with d elements, then all subsets of I with d elements belong to S. Therefore, in the symmetric case the only possible restrictions on the search domain are those concerning the number of defectives.

1.2 The Test Information Hypergraph

A *hypergraph* $H = (V, \mathcal{E})$ is an ordered pair of a set V of vertices and a set $\mathcal{E}$ of edges each consisting of a non-empty subset of the vertices. A *vertex cover* C of the hypergraph H is a set of vertices of H such that each edge of H contains at least one vertex from C. Denote by $C(H)$ the set of all vertex covers of H, and by $\tau(H)$ the minimum number of vertices forming a vertex cover of H. The *components* of H are the blocks of the finest partition of its vertex set V in which every edge lies completely in one block. We call a hypergraph $(V, \mathcal{E})$ an *antichain* if there are no $E_1, E_2 \in \mathcal{E}$ with $E_1 \subset E_2$.

In the following, we assume tacitly that we have chosen an algorithm and look at the sequence of tests for a fixed defective set.

Then, the information gathered after t tests can be represented as follows: The set G_t consists of all items that are known to be good because they belong to a group that has been tested with a negative result. In the *test information hypergraph* H_t on the vertex set $I \setminus G_t$ each edge is a group that is known to be contaminated.

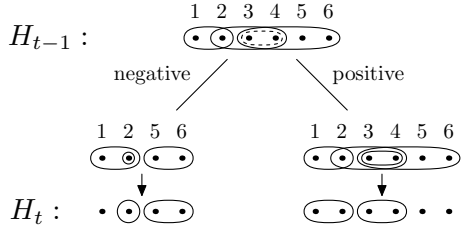
G_0 is empty and H_0 is the hypergraph on I without any edges. The step from $t-1$ to t works as follows: If the group T is tested with a positive result, let $G_t = G_{t-1}$ and H_t be H_{t-1} plus the edge $T \setminus G_{t-1}$. In the case of a negative test result, let $G_t = G_{t-1} \cup T$ and H_t on the vertex set $I \setminus G_t$ be H_{t-1} with all vertices in T removed from all edges. Then, regardless of the test result, all edges that include another edge are removed from H_t, which thereby becomes an antichain.

This procedure is based on the fact that a group T contains a defective if and only if $T \setminus G_{t-1}$ contains a defective. Furthermore, $T \subset T'$ being contaminated implies that T' is contaminated. Therefore, removing the edge corresponding to T' from the hypergraph H_t simplifies its structure without losing any information.

The step from $t-1$ to t is illustrated by the following example.

Example. Let $H_{t-1} = (\{i_1, \ldots, i_6\}, \{\{i_1, i_2\}, \{i_2, i_3, i_4\}, \{i_3, i_4, i_5, i_6\}\})$ and let $\{i_3, i_4\}$ be the next test group. If the test result is positive, then $\{i_3, i_4\}$ is added to the edges of H_{t-1}, and the edges $\{i_2, i_3, i_4\}$ and $\{i_3, i_4, i_5, i_6\}$ are removed, as they contain $\{i_3, i_4\}$, yielding $H_t = (\{i_1, \ldots, i_6\}, \{\{i_1, i_2\},$

$\{i_3, i_4\}\}$). If the test result is negative, $\{i_3, i_4\}$ is removed from all edges of H_{t-1}, yielding the hypergraph $(\{i_1, i_2, i_5, i_6\}, \{\{i_1, i_2\}, \{i_2\}, \{i_5, i_6\}\})$, from which $\{i_1, i_2\}$ is removed, as it contains $\{i_2\}$, resulting in $H_t = (\{i_1, i_2, i_5, i_6\}, \{\{i_2\}, \{i_5, i_6\}\})$.



Denote by D_t the union of all edges of H_t that contain just one item. All items in D_t are identified as defective. Denote by $U_t = I \setminus (G_t \cup D_t)$ the set of all items belonging neither to G_t nor to D_t. Then G_t, D_t and U_t are always disjoint with $G_t \cup D_t \cup U_t = I$.

If $T \subset G_{t-1}$, then the test result is known to be negative before the test is carried out; if, on the other hand, T includes an edge from H_{t-1}, the test result is known to be positive beforehand. In the following, only *reasonable* algorithms shall be considered, that is, those that do not perform any test whose outcome can be predicted. An algorithm that is not reasonable can be converted to a better reasonable algorithm by simply removing all predictable tests.

After t tests, the set G_t of good items and the test information hypergraph H_t, whose edges are contaminated groups, contain all information provided by the tests carried out so far. The defective sets consistent with the results of the first t tests are exactly the vertex covers $C(H_t)$. Therefore, an item is identified as defective if and only if it belongs to all vertex covers of H_t; likewise, it is identified as good if and only if it is not contained in any vertex cover of H_t.

The following theorem gives the conditions under which two hypergraphs have the same vertex covers.

Theorem 1.1 *Let $H = (V, \mathcal{E})$ be a hypergraph that is an antichain and $H' = (V, \mathcal{E}')$ a hypergraph on the same vertex set. Then $C(H) = C(H')$ if and only if $\mathcal{E} \subset \mathcal{E}'$ and for each $E' \in \mathcal{E}'$ there is an $E \in \mathcal{E}$ with $E \subset E'$.*

Proof. Suppose first $C(H) = C(H')$.

Let $E \in \mathcal{E}$. Then $V \setminus E \notin C(H)$, whereas $(V \setminus E) \cup i \in C(H)$ for all $i \in E$, as H is an antichain. Therefore $V \setminus E \notin C(H')$, which requires that there is an $E' \in \mathcal{E}'$ with $E' \subset E$. However, if E' is a proper subset of E, then for $i \in E \setminus E'$ the set $(V \setminus E) \cup i$ is no vertex cover of H', contradicting $C(H) = C(H')$. Hence $E \in \mathcal{E}'$ for all $E \in \mathcal{E}$, and thus $\mathcal{E} \subset \mathcal{E}'$.

Let $E' \in \mathcal{E}'$. Then $V \setminus E' \notin C(H')$ and therefore $V \setminus E' \notin C(H)$, which requires that there is $E \in \mathcal{E}$ with $E \subset E'$.

Now suppose that $\mathcal{E} \subset \mathcal{E}'$ and for each $E' \in \mathcal{E}'$ there is an $E \in \mathcal{E}$ with $E \subset E'$.

Then, every set that does not cover an edge of H does not cover the same edge of H', whereas every vertex cover of H covers also all edges of H'. Together, this yields $C(H) = C(H')$. □

Theorem 1.1 shows that the test information hypergraph H_t is minimal among the hypergraphs whose vertex covers are the defective sets consistent with the results of the first t tests.

1.3 Identifying an Unknown Number of Defectives

First we consider the case $S = 2^I$, in which the number of defectives is unknown. This includes the binomial group testing problem and all other probabilistic group testing problems in which no defective set in S has probability zero. On the combinatorial side, the competitive group testing problem falls under this case.

In the following, we examine under which conditions items are identified as either good or defective.

Lemma 1.2 *After t tests, an item is identified as good if and only if it is in G_t.*

Proof. All items in G_t belong to a group that has been tested with negative result and are therefore identified as good. All other items may be defective, as $I \setminus G_t$ is a vertex cover of H_t. □

Theorem 1.3 *Items are identified as good by the t^{th} test if and only if the test result is negative.*

Proof. By Lemma 1.2 exactly the items in $G_t \setminus G_{t-1}$ are identified as good in the t^{th} test. G_t contains items that are not in G_{t-1} precisely in the case of a negative test result. □

Lemma 1.4 *After t tests, an item is identified as defective if and only if it is in D_t.*

Proof. Every item $i \in D_t$ belongs to all vertex covers of H_t, as this is the only way to cover the edge $\{i\}$. It is therefore identified as defective.

Every edge in H_t contains either one item from D_t or at least two items from U_t. For every item $i \notin D_t$ it follows that $(D_t \cup U_t) \setminus i$ is a vertex cover of H_t. Thus i may be good. □

Theorem 1.5 *Items are identified as defective by the t^{th} test if and only if*

1. *the test result is positive and the test group contains exactly one item from* $I \setminus G_{t-1}$*, or*
2. *the test result is negative and the test group contains all items but one of a contaminated group with at least two items.*

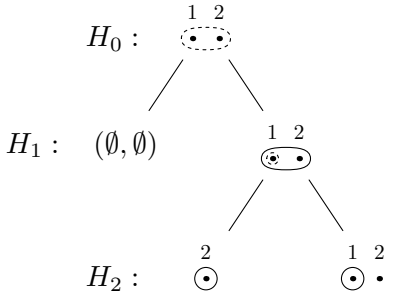
Proof. If condition 1 holds, then H_t has an edge with one item that is not present in H_{t-1}, as the test result must not be predictable. If condition 2 is fulfilled, then an edge in H_{t-1} with at least two items is reduced to one item in H_t. Therefore, in both cases D_t contains an item not contained in D_{t-1}, and by Lemma 1.4 this item is identified as defective.

Assume that neither condition 1 nor condition 2 holds. Then $D_t = D_{t-1}$, and by Lemma 1.4 no new item is identified as defective. The conditions given in the theorem are therefore necessary. □

If condition 1 holds, then just one defective is identified, whereas condition 2 allows several defectives to be identified by one test, as well as good items.

The following simple example serves to illustrate the application of Theorems 1.3 and 1.5.

Example. Let $I = \{i_1, i_2\}$, and let $\{i_1, i_2\}$ be the first test group. If the test result is negative, then $G_1 = \{i_1, i_2\}$ and $H_1 = (\emptyset, \emptyset)$; i_1 and i_2 are identified as good and no items are identified as defective. Now assume the result of the first test to be positive. Then $G_1 = \emptyset$ and $H_1 = (\{i_1, i_2\}, \{\{i_1, i_2\}\})$; no items are identified as good or defective. Let the second test be performed on $\{i_1\}$. If the test result is positive, then $G_2 = \emptyset$ and $H_2 = (\{i_1, i_2\}, \{\{i_1\}\})$; no items are identified as good by Theorem 1.3, and i_1 is identified as defective by Theorem 1.5, condition 1. If the result of the second test is negative, then $G_2 = \{i_1\}$ and $H_2 = (\{i_2\}, \{\{i_2\}\})$; i_1 is identified as good by Theorem 1.3 and i_2 as defective by Theorem 1.5, condition 2.



Now we give the condition under which an algorithm has finished after t tests if the number of defectives is unknown.

Theorem 1.6 *All items are identified as either good or defective after* t *tests if and only if* $G_t \cup D_t = I$*.*

Proof. The statement follows directly from Lemmas 1.2 and 1.4. □

1.4 Identifying a Bounded Number of Defectives

We consider now the group testing problem for which it is known that there are at least $d_{\min}$ and at most $d_{\max}$ defectives, with $0 \leq d_{\min} \leq d_{\max} \leq n$. This includes the (d, n) group testing problem with $d_{\min} = d_{\max} = d$ and the generalized (d, n) group testing problem with $d_{\min} = 0$ and $d_{\max} = d$.

The restriction on the total number of defectives translates into a restriction on the cardinality of the vertex covers of H: the remaining defective sets after t tests, that is, those consistent with the results of the first t tests, are all vertex covers of H_t that consist of at least $d_{\min}$ and at most $d_{\max}$ items. This implies that $\tau(H_t) \leq d_{\max}$ for all t. The direct methods of identifying items in Theorems 1.3 and 1.5 hold as well for a bounded number of defectives. Yet the restrictions on the number of defectives lead to additional indirect methods of identifying items, as we describe in the following theorems.

Theorem 1.7 *After t tests, an item $i \in U_t$ is identified as good if and only if $\tau(H_t) = d_{\max}$ and no minimum vertex cover of H_t contains i.*

Proof. If $\tau(H_t) = d_{\max}$, then the remaining defective sets are just the minimum vertex covers of H_t. Therefore exactly the items given in the theorem are identified as good.

If $\tau(H_t) < d_{\max}$, then every item i from U_t can be defective. To see this, add i to a minimum vertex cover, and add further items from U_t to reach a cardinality of at least $d_{\min}$, if necessary. The resulting vertex cover of H_t belongs to the remaining defective sets. □

If the conditions given in Theorem 1.7 are met for one item, then all items in U_t not contained in any edge of H_t are identified as good, as they never belong to a minimum vertex cover.

If $\tau(H_t) = d_{\max}$, then $\tau(H_{t'}) = d_{\max}$ for all $t' \geq t$, as $\tau(H_t)$ is nondecreasing in t and bounded from above by $d_{\max}$. Then in all remaining tests items can be identified as good by positive test results using Theorem 1.7.

Theorem 1.8 *After t tests, an item $i \in U_t$ is identified as defective if and only if*

1. *$|G_t| = n - d_{\min}$, or*
2. *all vertex covers of H_t without i contain at least $\tau(H_t) + k + 1$ items where $k = d_{\max} - \tau(H_t)$.*

Proof. If $|G_t| = n - d_{\min}$, the only remaining defective set is $I \setminus G_t = D_t \cup U_t$, in which all items are identified as defective.

From now on suppose $|G_t| < n - d_{\min}$.

If an item i satisfies condition 2, then all vertex covers of H_t consisting of at most $\tau(H_t)+k = d_{\max}$ items contain i. Thus i is identified as defective.

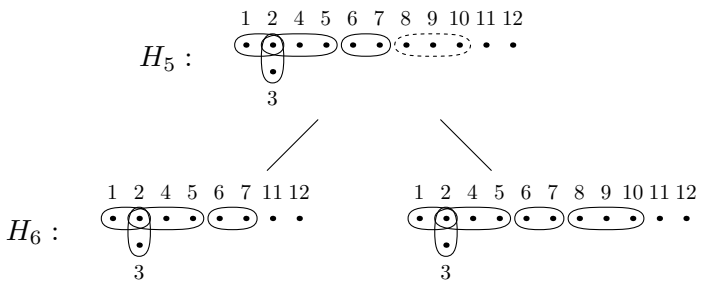
If, to the contrary, i does not fulfil condition 2, then there is a vertex cover of H_t that does not contain i and consists of at most $d_{\max}$ items. If necessary, add further items from $(I \setminus G_t) \setminus i$ to reach a cardinality of at least $d_{\min}$, which is always possible as $|(I \setminus G_t) \setminus i| = n - |G_t| - 1 \geq d_{\min}$. The resulting vertex cover has a cardinality of at least $d_{\min}$ and at most $d_{\max}$ and does not contain i. Hence i is not identified as defective. □

Each item i that satisfies condition 2 of Theorem 1.8 belongs to at least $k+2$ edges of H_t. Otherwise, a vertex cover of H_t without i with no more than $\tau(H_t)+k$ items could be constructed by removing i from a minimum vertex cover and adding one item distinct from i from every edge of H_t that contains i. If the algorithm belongs to the nested class, this condition is never met, because no two edges of H_t share any item.

Minimum vertex covers can be found by regarding all components of the hypergraph independently. Each minimum vertex cover is the union of minimum vertex covers of all components and vice versa. Therefore, if the condition $\tau(H_t) = d_{\max}$ of Theorem 1.7 is fulfilled, then the items not contained in any minimum vertex cover of H_t and thus known to be good can be identified separately for each component of H_t. Likewise, in Theorem 1.8, condition 2, the minimum vertex cover without i of the component containing i has to contain $k+1$ additional items compared to the minimum vertex cover including i, as the other components are not affected by the omission of i from the vertex covers.

The following example demonstrates the use of Theorems 1.7 and 1.8.

Example. Let $n = 12$, $I = \{i_1, \ldots, i_{12}\}$, $d_{\min} = 0$, and $d_{\max} = 3$. Suppose $G_5 = \emptyset$ and $H_5 = (I, \{\{i_1, i_2\}, \{i_2, i_3\}, \{i_2, i_4, i_5\}, \{i_6, i_7\}\})$, which implies $D_5 = \emptyset$. The minimum vertex covers of H_5 are $\{i_2, i_6\}$ and $\{i_2, i_7\}$, thus $\tau(H_5) = 2$. Since $\tau(H_5) < d_{\max}$, by Theorem 1.7 no items are identified as good. Condition 1 of Theorem 1.8 is not fulfilled, but i_2 satisfies condition 2: all vertex covers of H_5 without i_2 contain i_1, i_3, at least one of i_4 and i_5, and at least one of i_6 and i_7, thus at least 4 items. Therefore i_2 is identified as defective. The components $\{i_1, i_2, i_3, i_4, i_5\}$ and $\{i_6, i_7\}$ of H_5 can as well be looked at separately.



Let the next test be performed on $\{i_8, i_9, i_{10}\}$. Assume the result to be positive. Then $G_6 = \emptyset$ and $H_6 = (I, \{\{i_1, i_2\}, \{i_2, i_3\}, \{i_2, i_4, i_5\}, \{i_6, i_7\}, \{i_8, i_9, i_{10}\}\})$, which implies $\tau(H_6) = 3$. Thus i_1, i_3, i_4, i_5, i_{11}, and i_{12} satisfy the conditions of Theorem 1.7 and are identified as good, as no minimum vertex cover of H_6 contains them.

Now we can establish exactly under which circumstances the algorithm has finished after t tests.

Theorem 1.9 *All items are identified as either good or defective after t tests if and only if*

1. $G_t \cup D_t = I$, *or*
2. $|G_t| = n - d_{\min}$, *or*
3. $\tau(H_t) = d_{\max}$ *and there is exactly one minimum vertex cover of* H_t.

Proof. All items in G_t are identified as good, all items in D_t as defective. For all items to be identified, therefore all items in $I \setminus (G_t \cup D_t) = U_t$ must fulfil the conditions of either Theorem 1.7 to be identified as good or Theorem 1.8 to be identified as defective.

Since condition 1 implies $U_t = \emptyset$, it is clearly sufficient.

If condition 2 holds, all items in U_t are identified as defective by Theorem 1.8, condition 1. Otherwise, condition 1 of Theorem 1.8 is never met.

If condition 3 holds, all items in U_t that belong to the minimum vertex cover of H_t are identified as defective by Theorem 1.8, condition 2, whereas all other items in U_t are identified as good by Theorem 1.7.

Assume now that none of the conditions holds. Then $\tau(H_t) < d_{\max}$ or there are at least two minimum vertex covers of H_t.

First suppose $\tau(H_t) < d_{\max}$. Then no item in U_t is identified as good by Theorem 1.7. If all items in U_t were identified as defective and therefore fulfilled condition 2 of Theorem 1.8, then all vertex covers of H_t containing not all items from U_t and thereby all vertex covers would have at least $d_{\max} + 1$ elements, which would leave no defective sets consistent with the test results. Thus, not all items in U_t are identified as good or defective.

Now suppose the existence of at least two minimum vertex covers of H_t with $d_{\max}$ elements. Then all items contained in one vertex cover but not in the other cannot be identified as good or defective. □

The following lemma is a generalization of a lemma by Hwang, Song, and Du [13] that treated the case $d_{\min} = d_{\max} = d$. It is used in the proof of Lemma 2.3 below.

Lemma 1.10 *Suppose $d_{\min} < n$. If an algorithm identifies a defective set D after t tests with $|G_t| = n - d_{\min}$ and $|U_t| \geq 1$, then there is a defective set in S for which the first $t-1$ test results are the same as for D but the t^{th} test result is positive instead of negative and for which the algorithm needs at least $t+|U_t|-1$ tests if $d_{\min} = d_{\max}$ and at least $t+|U_t|$ tests if $d_{\min} < d_{\max}$.*

Proof. The condition $d_{\min} < n$ implies $t \geq 1$. For the defective set D the result of the t^{th} test is negative. Otherwise $G_t = G_{t-1}$ would hold and the algorithm would already have finished after $t-1$ tests by Theorem 1.9, condition 2. Furthermore $D = I \setminus G_t = D_t \cup U_t$. For each $i \in U_t$ the set $D \setminus i$ is a vertex cover of H_t, as $D \in C(H_t)$ and all edges of H_t containing i contain also another element of D. Since $C(H_t) \subset C(H_{t-1})$, it follows that $D \setminus i \in C(H_{t-1})$ for all $i \in U_t$.

Denote the t^{th} test group by T. Let the set G'_t and the information test hypergraph H'_t describe the case that the result of the t^{th} test is positive. Then $G'_t = G_{t-1} = G_t \setminus (T \setminus G_{t-1})$, and H'_t is H_{t-1} plus the edge $T \setminus G_{t-1}$. Choose arbitrarily $i_T \in T \setminus G_{t-1}$. Then $i_T \notin D$. For each $i \in U_t$ the set $(D \setminus i) \cup i_T$ is a vertex cover of H'_t with $d_{\min}$ elements, as $D \setminus i$ covers H_{t-1} and i_T covers the additional edge $T \setminus G_{t-1}$.

Now consider the sequence of tests that results if a test result is positive if and only if the test group contains at least one item from $D \cup i_T$. Denote by t' the number of tests after which the algorithm has finished in this case. Then by Theorem 1.9 one of the following conditions is fulfilled.

1. $G_{t'} \cup D_{t'} = I$.
   This implies $t' \geq t + |U_t|$, as the items in U_t cannot be identified as good and at most one item in U_t can be identified as defective by one test.

2. $|G_{t'}| = n - d_{\min}$.
   This never holds, as $G_{t'} \subset I \setminus (D \cup i_T)$ and therefore $|G_{t'}| < n - d_{\min}$.

3. $\tau(H'_{t'}) = d_{\max}$ and there is exactly one minimum vertex cover of $H'_{t'}$.
   Therefore, if $d_{\min} = d_{\max}$, then all but one, else all of the $|U_t|$ vertex covers $(D \setminus i) \cup i_T$ for $i \in U_t$ belonging to $C(H'_t)$ have been excluded by the last $t' - t$ tests. Of these, at most one can be excluded by each test, as $(D \setminus i) \cup i_T$ is excluded if and only if the test group contains i but no other item from $D \cup i_T$. Thus $t' \geq t + |U_t| - 1$ if $d_{\min} = d_{\max}$ and $t' \geq t + |U_t|$ if $d_{\min} < d_{\max}$.

□

Hence, condition 2 of Theorem 1.9 does not influence the worst case number of tests needed by an algorithm.

Chapter 2

Complete Group Testing

In this chapter, we introduce a new variant of combinatorial group testing, the complete group testing problem. Drawing on the analysis in the first chapter, we show that its solution requires exactly one more test than that of the (d, n) group testing problem.

2.1 Variants of Combinatorial Group Testing

If nothing is known about the number of defectives, the search domain $S = 2^I$ has cardinality 2^n. Thus, in the worst case at least n tests are needed, which can be achieved by simply testing all items individually. To get a meaningful problem, we therefore have to impose some restrictions on the number of defectives.

As in the first chapter, only reasonable algorithms A, that is, those without predictable tests, shall be allowed.

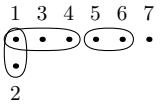
In the (d, n) *group testing problem* as defined by Du and Hwang [6, Section 1.2] the number of defectives is known to be exactly d, with $0 \leq d \leq n$. Denote by $M_A(d, n)$ the number of tests needed by the algorithm A in the worst case. Then $M(d, n) = \min_A M_A(d, n)$ denotes the maximum number of tests needed by a minimax algorithm.

However, the assumption that the number of defectives is known exactly rarely holds in practical applications. In the *generalized* (d, n) *group testing problem* it is only known that there are at most d defectives. We denote by $\bar{M}_A(d, n)$ the number of tests needed by the algorithm A in the worst case and by $\bar{M}(d, n)$ the worst case number of tests of a minimax algorithm. Hwang, Song, and Du [13] proved $\bar{M}(d, n) \leq M(d, n) + 1$ for $d < n$.

If an algorithm designed for the generalized (d, n) group testing problem encounters more than d defectives in a practical application, it typically identifies just d defectives and then stops. In this case, it lets pass defective items as good ones, but it may as well label good items as defective, as in the following example.

Example. Consider the test information hypergraph $H_t = (\{i_1, \ldots, i_7\}, \{\{i_1, i_2\}, \{i_1, i_3, i_4\}, \{i_5, i_6\}\})$. Under the assumption that there are at most 2 defectives, the item i_1 is identified as defective. But allowing for 3 defectives, i_1 can as well be good if i_2 and either i_3 or i_4 are defective.

H_t :



We address these limitations by formulating the *complete group testing problem*: The number of defectives is unknown and an algorithm must identify all items as good or defective. Denote by $\bar{M}_A(d|n)$ the maximum number of tests needed by the algorithm A in those cases in which there are at most d defectives. Equivalently, A solves the generalized (d, n) group testing problem, but establishes in addition that there are no more than d defectives, or states that there are more than d defectives in $\bar{M}_A(d|n)$ tests. Let $\bar{M}(d|n) = \min_A \bar{M}_A(d|n)$ denote the worst case number of tests of a minimax algorithm. Similarly, denote by $M_A(d|n)$ and $M(d|n) = \min_A M_A(d|n)$ the maximum number of tests needed by the algorithm A and a minimax algorithm respectively in those cases in which there are exactly d defectives instead of at most d defectives. Since $\bar{M}(d|n) = M(d|n)$ by Theorem 2.1 below, we call both $\bar{M}(d|n)$ and $M(d|n)$ the complete group testing problem.

Colbourn [4] formulated a problem called the strict group testing problem for the nonadaptive case, which is similar to the complete group testing problem.

2.2 Complete Related to (d, n) Group Testing

The following theorem establishes the relationship between the complete and (d, n) group testing problems.

Theorem 2.1 *For $d < n$,*

$$\bar{M}(d|n) = M(d|n) = M(d, n) + 1.$$

The condition $d < n$ is necessary, as complete group testing for $d = n$ requires individual testing of all items and thus $\bar{M}(n|n) = M(n|n) = n$, but on the other hand obviously $M(n, n) = 0$.

Interestingly, it makes no difference whether the complete group testing problem is examined for at most d or exactly d defectives.

For the proof of the theorem we need the following lemmas, in which A always denotes an algorithm for the (d, n) group testing problem, and A' always denotes an algorithm for the complete group testing problem.

Lemma 2.2 *For all A',*

$$M_{A'}(d|n) \leq \bar{M}_{A'}(d|n).$$

Proof. The lemma follows directly from the definitions of $M_{A'}(d|n)$ and $\bar{M}_{A'}(d|n)$. □

Lemma 2.3 *Suppose $d < n$. For each A there exists an A' so that*

$$\bar{M}_{A'}(d|n) \leq M_A(d, n) + 1.$$

Proof. Let A' perform the same tests as A. As A' lacks information about the number of defectives, it has not necessarily finished when A has. The additional tests are specified below.

Consider the sequence of tests that A performs for the defective set D with d elements. Let t be the number of tests needed by A in this case. Then by Theorem 1.9 at least one of the following conditions is fulfilled.

1. $G_t \cup D_t = I$.
   In this case, no further tests are necessary, as A' has finished by Theorem 1.6.

2. $|G_t| = n - d$.
   If $|U_t| = 0$, refer to case 1. Otherwise, let A' test all items in U_t individually unless they are identified as defective in the meantime. Thus A' requires at most $t + |U_t|$ tests to identify all items as good or defective. On the other hand, the application of Lemma 1.10 on A with $d_{\min} = d_{\max} = d < n$ shows that there is a defective set for which A needs at least $t + |U_t| - 1$ tests.

3. $\tau(H_t) = d$ and D is the only minimum vertex cover of H_t.
   In the $(t+1)^{\text{th}}$ test, let A' test the group $I \setminus (D \cup G_t)$.
   If the test result is negative, then $G_{t+1} = I \setminus D$ and $D_{t+1} = D$, as each $i \in D \setminus D_t$ belongs to an edge of H_t with at least two elements that contains no other items from D, which is reduced to $\{i\}$ in H_{t+1}. Hence $G_{t+1} \cup D_{t+1} = I$, and by Theorem 1.6 A' has finished after $t+1$ tests.
   If the result of the $(t+1)^{\text{th}}$ test is positive, there are at least $d+1$ defectives. This case does not influence $\bar{M}_{A'}(d|n)$, and A' can be continued in an arbitrary way till all items are identified as good or defective.

□

Lemma 2.4 *For each A' there exists an A so that*

$$M_A(d, n) + 1 \leq M_{A'}(d|n).$$

Proof. Derive the algorithm A from A' by simply omitting all tests that become predictable due to the knowledge that there are exactly d defectives.

Let D be a defective set for which A needs the worst case number $t = M_A(d, n)$ of tests and for which the last test result is positive. This can always be found, because if A has not finished after $t-1$ tests, it finishes with the t^{th} test regardless of the test result. Then $|G_{t-1}| < n - d$, as otherwise A would have finished after $t-1$ tests by Theorem 1.9, condition 2. As the result of the t^{th} test is positive, $G_t = G_{t-1}$ and therefore $|G_t| < n - d$.

If, for the defective set D, one of the first t tests in A' has been omitted in A, then A' obviously needs at least $t+1$ tests. Otherwise, if the first t tests of A and A' are the same, the condition $G_t \cup D_t = I$ of Theorem 1.6 is not fulfilled for A'. Therefore, the algorithm A' has not finished after t tests and needs at least $t+1$ tests. □

Proof of Theorem 2.1. Lemmas 2.2, 2.3, and 2.4 transfer to the number of tests needed by minimax algorithms for the same problems. For $d < n$, this yields

$$M(d|n) \leq \bar{M}(d|n) \leq M(d, n) + 1 \leq M(d|n).$$

□

Hence, solving the complete group testing problem always needs exactly one more test than solving the (d, n) problem. The proofs of Lemmas 2.3 and 2.4 give explicit instructions for converting algorithms between the complete and (d, n) problems. Therefore, existing algorithms for the (d, n) group testing problem can be easily modified to solve the complete problem needing just one additional test.

Furthermore, nested algorithms remain in the nested class when converted between the complete and (d, n) problems. Hence, Theorem 2.1 holds as well restricted to the nested class.

As obviously $\bar{M}(d, n) \leq \bar{M}(d|n)$, the result from Hwang, Song, and Du [13] for the generalized (d, n) group testing problem can be derived as a corollary from Theorem 2.1.

Corollary 2.5 *For $d < n$,*

$$\bar{M}(d, n) \leq M(d, n) + 1.$$

Chapter 3

The Split and Overlap Algorithm

The final chapter is devoted to a new algorithm for the combinatorial group testing problems discussed in the last chapter. We build up the split and overlap algorithm modularly. After a brief overview, we introduce new efficient subalgorithms and a general method of scaling up arbitrary subalgorithms by powers of two. Using these as building blocks, we construct algorithms that repeatedly test groups of the same size and always follow the same procedure if a group is found to be contaminated. The split and overlap algorithm then works by choosing the best of these algorithms depending on $\frac{n}{d}$. At each stage, estimates of the maximum number of tests used are provided, and finally the estimate for the whole algorithm is shown to be significantly lower than the best upper bound for the (d, n) group testing problem in the literature.

3.1 Nested Algorithms

The test information hypergraph introduced in Section 1.2 represents the information gathered by the group tests. The edges of the hypergraph are the minimal groups known to be contaminated. We call an item *free* if it has not been identified as either good or defective and does not belong to any edge in the test information hypergraph, that means, if there is no information about the item.

Sobel and Groll [19] introduced the *nested class*, a subclass of simple algorithms for the group testing problem with the following property. If the test information hypergraph has an edge with at least two items, the next test is performed on a proper subset of this edge. Initially, a nested algorithm may test arbitrary groups until a test yields a positive result. Denote the contaminated group by L. We test $K \subset L$ next. If the result is positive, the edge K replaces the edge L in the test information hypergraph.

Otherwise, if the result is negative, removing the items in K from the edge L results in the new edge $L \setminus K$. We continue by successively testing subsets of the resulting edge, until the edge contains only a single item, which is identified as defective. Now we start again by testing arbitrary groups until a test yields a positive result.

Chang, Hwang, and Weng [2] gave the following binary splitting algorithm to identify one defective in a contaminated group L of $l \geq 2$ items.

Subalgorithm B_l

Let $k = \max\left(l - 2^{\lceil \log l \rceil - 1}, 2^{\lceil \log l \rceil - 2}\right)$.
Test a group $K \subset L$ of size k.
If the result is positive,
  apply B_k to K,
else
  apply B_{l-k} to $L \setminus K$.

B_1 identifies a defective without any further tests. If l is a power of 2, the algorithm B_l repeatedly halves the size of the contaminated group.

Lemma 3.1 *The subalgorithm B_l identifies a defective in at most $\lceil \log l \rceil$ tests. If $\lceil \log l \rceil$ tests are actually used, then at least $2^{\lceil \log l \rceil} - l$ good items are identified as well.*

The proof of Lemma 3.1 is by induction and is detailed in [2].

Hwang [11] gave the following generalized binary splitting algorithm for the (d, n) group testing problem.

Algorithm G

Denote by I the set of n items containing d defectives. Always remove items from I when they are identified as good or defective.

While $d \geq 1$:
  If $|I| \leq 2d - 2$, test all items in I individually and then stop.
  Let $k = \left\lfloor \log \frac{|I|-d+1}{d} \right\rfloor$.
  Test a group of size 2^k.
  If the result is positive,
    apply B_{2^k} to the contaminated group and set $d := d - 1$.

The only general lower bound that is known for the (d, n) group testing problem is the information lower bound $\left\lceil \log \binom{n}{d} \right\rceil$, which is based on the fact that each test divides the search domain of cardinality $\binom{n}{d}$ in two.

Du and Hwang [6, Section 2.4] proved that the generalized binary splitting algorithm G exceeds the information lower bound by at most $d-1$ tests for $d \geq 2$. Interestingly, the application of a much more general result for hypergraphs by Triesch [20] to complete hypergraphs of rank d yields the

same upper bound. The corresponding algorithm also belongs to the nested class, but differs considerably from Hwang's generalized binary splitting algorithm.

Du and Hwang [6, Section 2.5] gave a set of recursive equations that describe the number of tests required by a minimax nested algorithm and a procedure to find such an algorithm based on computing the solution to these equations. This algorithm repeatedly tests a group of some size l depending on the actual values of n and d and applies B_l if the group is found to be contaminated. Unfortunately, no estimate of the number of tests needed by this minimax nested algorithm is provided.

3.2 Overview of the Split and Overlap Algorithm

We formulate the split and overlap algorithm for the complete group testing problem introduced in Section 2.1. This supposes that there are n items to be classified as either good or defective and the number of defectives is unknown; we aim to minimize the maximum number $M_A(d|n)$ of tests needed if there are at most d defectives. The algorithm can be adapted to the (d, n) and generalized (d, n) group testing problems, in which the number of defectives is known to be exactly respectively at most d, by simply omitting all tests that become predictable due to the extra information about the number of defectives.

At the start of the split and overlap algorithm, we choose the *initial test group size* m depending only on the ratio $\frac{n}{d}$ of all items to the defectives.

If we do not know any contaminated group, we always test a group of m items as long as there are enough free items left. If the test result is negative, we test the next group of m items. Otherwise, if the result is positive, we proceed to identify at least one defective in the group of m items. As the nested algorithms in the last section, we always test subsets of the current contaminated group. However, contaminated groups of certain sizes are dealt with by special subalgorithms, which are more efficient due to the use of overlapping test groups. Some of these subalgorithms identify at least two or three defectives starting with as many disjoint contaminated groups of the same size l. For these subalgorithms we collect the required number of contaminated groups of size l by setting them aside as they occur. After identifying at least one defective or setting aside a contaminated group of size l we continue by testing the next group of the same size m as before.

When there are not enough free items left to repeat this procedure, we use binary splitting till the end in a way specified in Section 3.6.

We show in Section 3.7 that the algorithm with initial test group size m requires at most $\frac{1}{m}$ tests for the identification of each good item and a constant number of tests to identify each defective plus 5 tests.

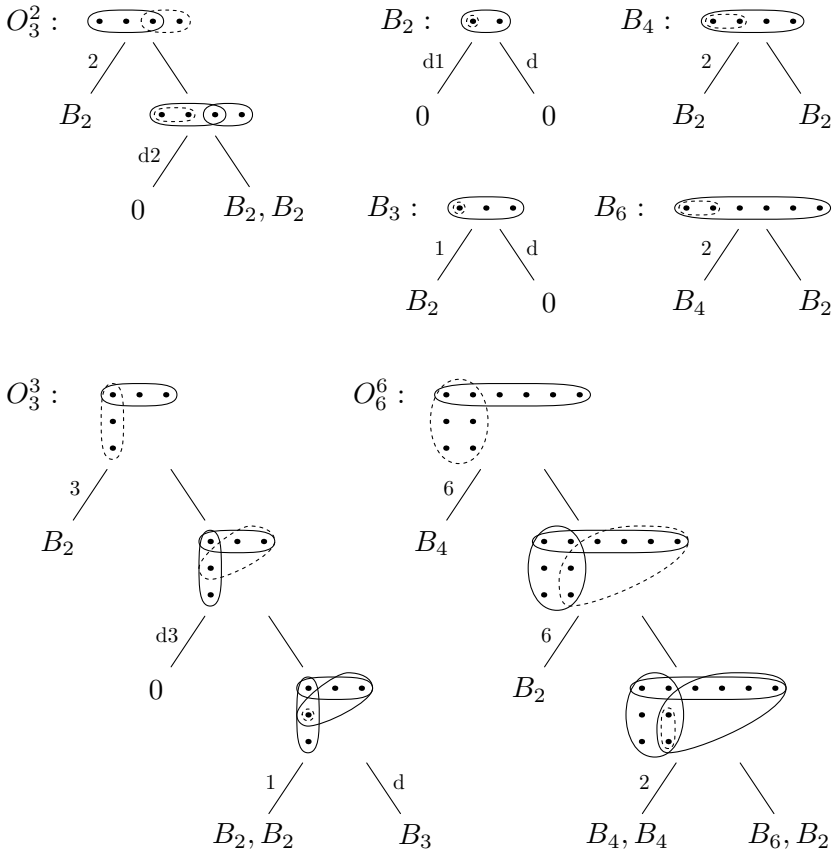


Figure 3.1: The subalgorithms O_3^2,O_3^3, and O_6^6

3.3 Overlapping Subalgorithms

We can represent a subalgorithm as a binary tree in the following way. Each test is represented by a node that has two subtrees for the cases of a negative and positive test result, respectively. By convention, we always depict the negative test result on the left and the positive test result on the right side. The root corresponds to the start of the subalgorithm, when there are one or more contaminated groups of size l. Each leaf marks the end of the subalgorithm, when at least one defective has been identified and only contaminated groups of size l may be left.

At each node we display the part of the information test hypergraph that is relevant to the subalgorithm and specifically omit edges containing just a single item. Additional edges that are disjoint to the displayed edges can be omitted safely, as they do not interfere with the others as long as no items from them are included in the test group. The next test group is always indicated by a dashed line.

The number of good items identified in the case of a negative test result,

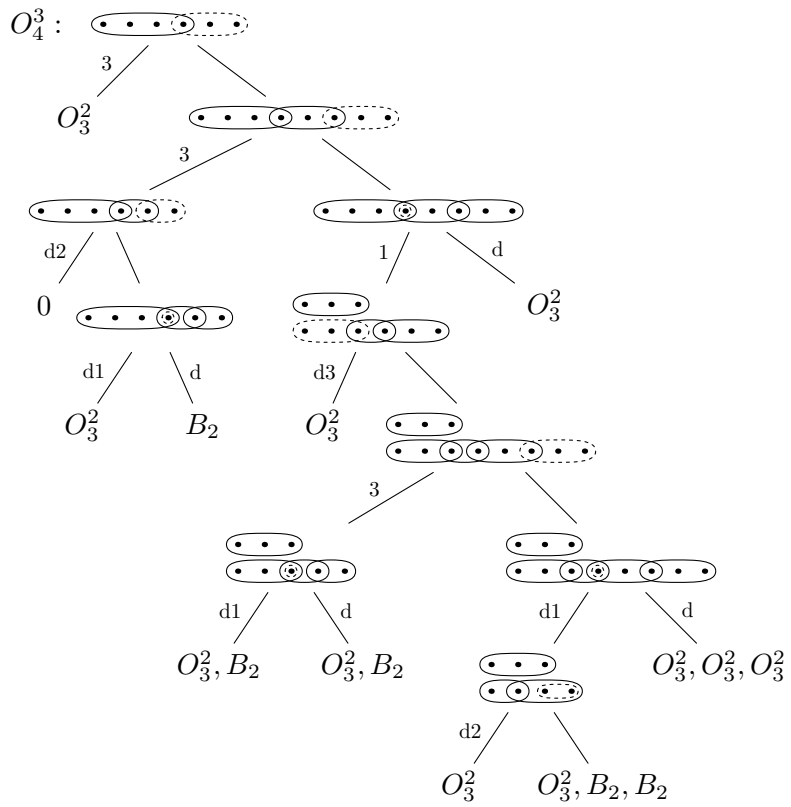


Figure 3.2: The subalgorithm O_4^3

which always coincides with the size of the test group, is displayed above the connection to the left child. An item identified as defective is indicated by the letter 'd' above the connection to the corresponding child.

A zero on a leaf of the binary tree indicates that no contaminated group containing at least two items is left. Subtrees identical to subalgorithms already shown are represented by the name of the corresponding subalgorithm instead of the contaminated group; for example, B_2 indicates that an edge containing two items is left of which one is tested next. Several subalgorithms separated by commas indicate that the corresponding edges are dealt with independently one after the other. A contaminated group that is left over is represented by a number indicating its size.

Items from a contaminated group that become free again are reused for further tests whenever possible to reduce the number of free items required by the subalgorithms.

The nested algorithms in Section 3.1 use subalgorithm B_3 on contaminated groups of three items, that is, they first test a single item from the group. In the worst case, B_3 identifies one defective and one good item in two tests. The subalgorithms O_3^2 and O_3^3 depicted in Figure 3.1 improve

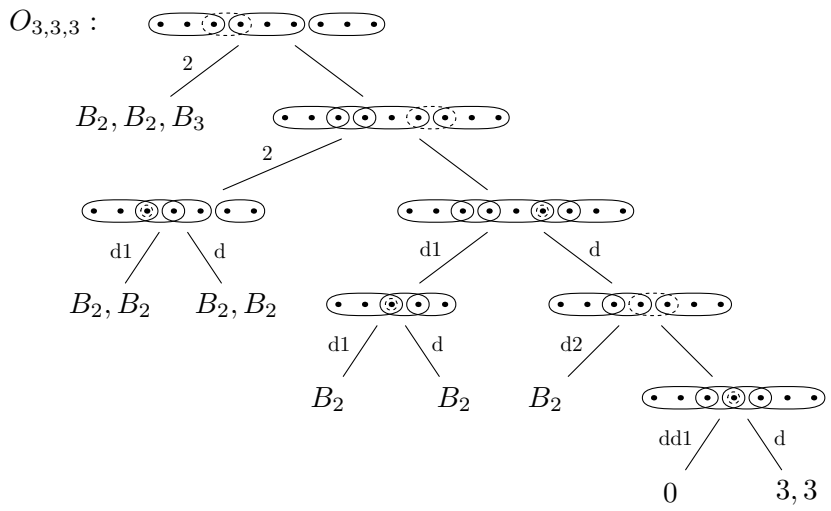


Figure 3.3: The subalgorithm $O_{3,3,3}$

on this by first testing a group of size two respectively three that contains exactly one of the three items in the original contaminated group. If one of the first two test results is negative, O_3^2 and O_3^3 identify one defective in two tests, like B_3, but at least two respectively three good items instead of one. Otherwise, in the case of two consecutive positive test results, they need four respectively five tests, but identify two defectives instead of one. In addition to the contaminated group of size three, O_3^2 requires one and O_3^3 requires two free items. The technique to derive O_6^6 in Figure 3.1 from O_3^3 is explained in Section 3.5.

On a contaminated group of four items, nested algorithms always use subalgorithm B_4, that is, they halve the group two times. In the worst case, B_4 identifies one defective and no good items in two tests. The algorithm that always tests a group of four items and then uses B_4 if the test result is positive needs three tests for the identification of each defective. Surprisingly, we can do better than this. The subalgorithm O_4^3 depicted in Figure 3.2 starts by testing a group of size three that contains exactly one of the four items in the original contaminated group. It typically needs more tests than B_4 to identify a defective, but more than compensates for this by identifying good items as well. In addition to the contaminated group of size four, O_4^3 requires up to ten free items.

If a subalgorithm is used in an algorithm with the initial test group size m, the identification of each good item needs $\frac{1}{m}$ tests. The preceding subalgorithms improve on B_3 and B_4 only by identifying more good items. They are therefore best for small m. To construct subalgorithms on a contaminated group of size l that are good for large m, we have to reduce the number of tests needed to identify a defective. As we have to distinguish at least the

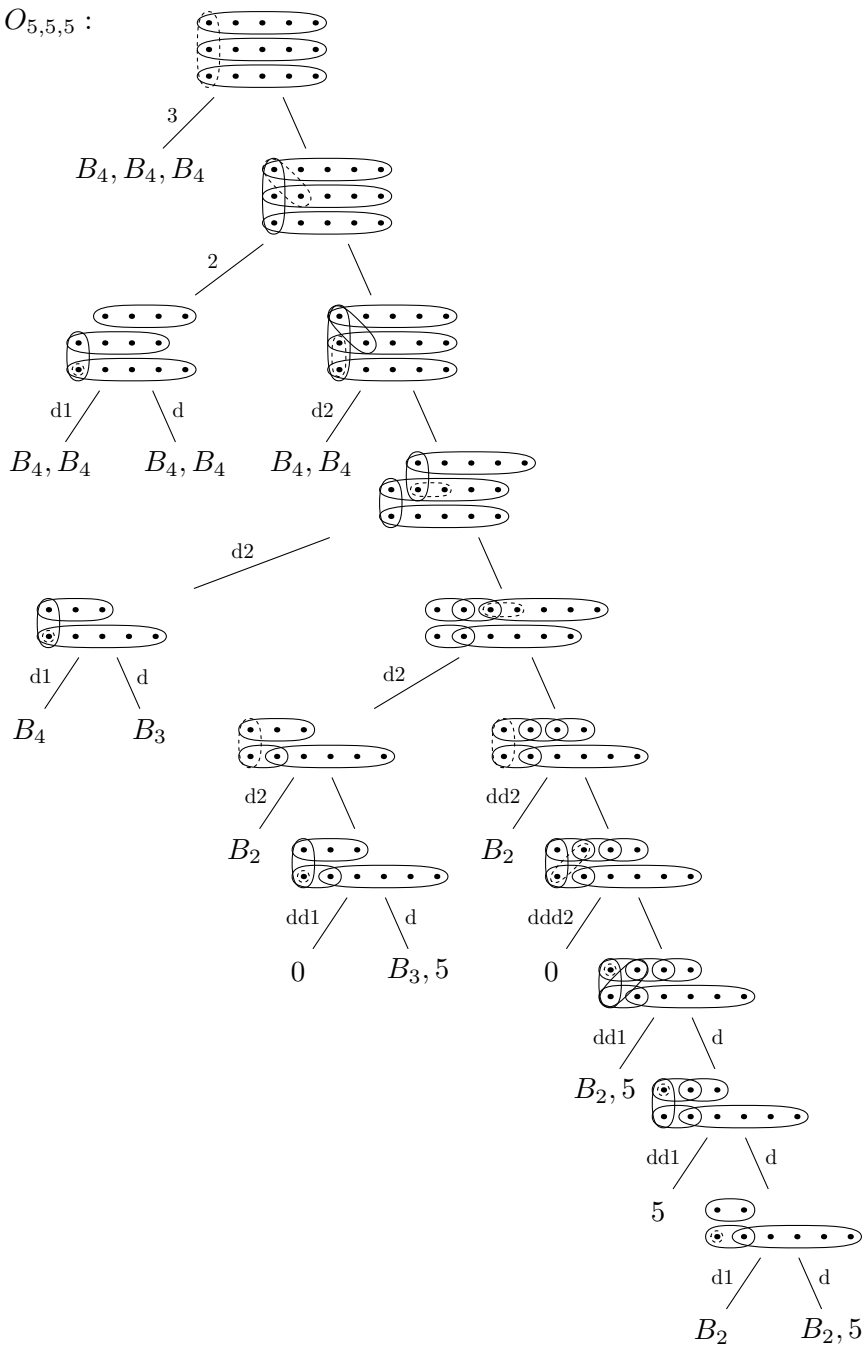


Figure 3.4: The subalgorithm $O_{5,5,5}$

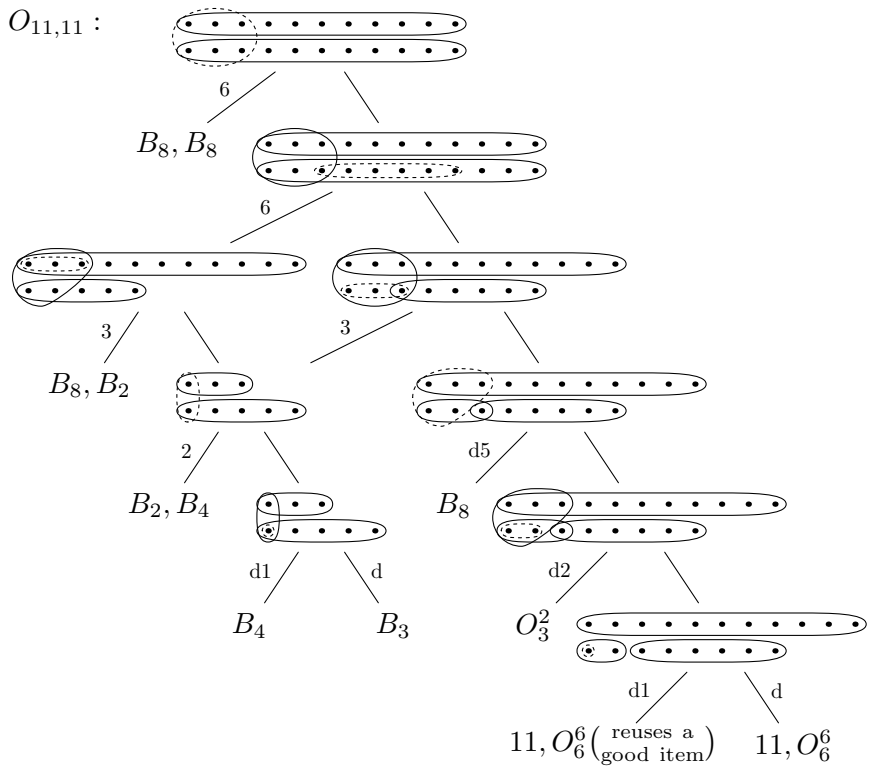


Figure 3.5: The subalgorithm $O_{11,11}$

l cases in which exactly one of the items in the group of size l is defective, we need at least $\lceil \log l \rceil$ tests, which is already achieved by subalgorithm B_l. To do better, we simultaneously identify at least one defective in each of k contaminated groups of size l. This requires at least $\lceil \log l^k \rceil$ tests in the worst case, that is, $\frac{1}{k} \lceil k \log l \rceil$ tests per defective.

The subalgorithm $O_{3,3,3}$ depicted in Figure 3.3 needs five tests to identify three defectives in three contaminated groups of size three. The $1\frac{2}{3}$ tests per defective compare favourably with the two tests needed by B_3, O_3^2, and O_3^3. Additionally, $O_{3,3,3}$ identifies at least one good item. If all five test results are positive, then no good items can be identified, and the subalgorithm identifies two defectives in one of the groups of size three and leaves the other two groups unchanged.

It would be even better to use eight tests to identify five defectives in five contaminated groups of size three, yielding $1\frac{3}{5}$ tests per defective. This would require the $3^5 = 243$ cases in which each of the five groups contains exactly one defective to be split by the first test in such a way that neither after a positive nor a negative test result more than $2^7 = 128$ cases remain, which cannot be accomplished by a group test.

Similarly to $O_{3,3,3}$, the subalgorithm $O_{5,5,5}$ depicted in Figure 3.4 starts

with three contaminated groups of size five and typically identifies three defectives and at least two good items in seven tests. The $2\frac{1}{3}$ tests per defective are much better than the three tests required by B_5. Alternatively, $O_{5,5,5}$ may identify three defectives and at least one good item in nine tests while leaving one of the original groups of size five unchanged. All other possibilities are not relevant for the worst case, as we show in the next section.

The subalgorithm $O_{11,11}$ depicted in Figure 3.5 needs seven tests to identify two defectives and at least four good items in two contaminated groups of eleven items. Again, the $3\frac{1}{2}$ tests per defective are less than the four tests required by B_{11}. Alternatively, the subalgorithm may identify two defectives and at least six good items in nine tests while leaving one of the original groups of size eleven unchanged, whereas all other possibilities turn out not to be relevant for the worst case. If the results of the first five tests are positive and of the sixth test negative, only three of the items in the two groups of size eleven are not contained in any contaminated group, but subalgorithm O_6^6 needs four free items. To avoid requiring the existence of another free item, O_6^6 reuses an item already identified as good in the next test, which may reduce the number of good items identified by one but does not affect the worst case behaviour of $O_{11,11}$.

3.4 Cost Estimate of Subalgorithms

In the binary tree representation, each sequence of test results in the subalgorithm corresponds to a path from the root to a leaf. We denote the set of all these paths by T. From now on, we refer to paths from the root to a leaf simply as paths.

For the path $P \in T$ denote by d_P the number of defectives and by g_P the number of good items identified along the path; denote by t_P the number of tests needed; denote by s_P the number of contaminated sets of size l present in the root of the tree but not in the leaf at the end of the path, that is, the number of groups of size l used up to identify defectives. Let $\bar{g}_P = \frac{g_P}{d_P}$, $\bar{t}_P = \frac{t_P}{d_P}$, and $\bar{s}_P = \frac{s_P}{d_P}$ denote the corresponding values per identified defective.

In the following, we assume that the main algorithm always tests a group of size m if no contaminated groups are known and that all contaminated groups of size l required by the subalgorithm are obtained along the same path in the algorithm. Denote by $t_{m,l}$ the number of tests needed on this path until a contaminated group of size l is known including the first test on m items with positive result and denote by $g_{m,l}$ the number of good items identified in the process. If we assign a cost of $\frac{1}{m}$ tests to the identification of each good item, then $c_{m,l} = t_{m,l} - \frac{g_{m,l}}{m}$ describes the cost to get a contaminated group of size l.

If the tests follow the path P inside the subalgorithm, then d_P defectives are identified using $s_P t_{m,l}+t_P$ tests while identifying $s_P g_{m,l}+g_P$ good items. Then we get $f\left(c_{m,l}, \frac{1}{m}\right)$, the worst case cost to identify a defective, by taking the maximum over all paths in the subalgorithm.

$$\begin{aligned} f\left(c_{m,l}, \tfrac{1}{m}\right) &= \max_{P\in T} \frac{1}{d_P}\left(s_P t_{m,l} + t_P - \frac{s_P g_{m,l}}{m} - \frac{g_P}{m}\right) \\ &= \max_{P\in T} \bar{s}_P c_{m,l} + \bar{t}_P - \frac{\bar{g}_P}{m} \\ &= \max_{P\in T} f_P\left(c_{m,l}, \tfrac{1}{m}\right). \end{aligned}$$

Here, let

$$f_P(x,y) = \bar{t}_P + \bar{s}_P x - \bar{g}_P y$$

be the cost to identify a defective if path P is followed in the subalgorithm where $x = c_{m,l}$ and $y = \frac{1}{m}$ are determined by the main algorithm. This can be depicted as a plane in R^3. Then $f(x,y)$ is the maximum over the planes of every path in the subalgorithm. As $m \geq l$ and $c_{m,l} \geq 1$ always hold, we are interested only in the domain $D_l = \left\{(x,y) | 1 \leq x, 0 \leq y \leq \frac{1}{l}\right\}$.

The following lemma gives the conditions under which the plane belonging to the path $Q \in T$ is below the plane of the path $P \in T$ on the whole domain D_l.

Lemma 3.2 *$f_P(x,y) \geq f_Q(x,y)$ on D_l if and only if $\bar{s}_P \geq \bar{s}_Q$ and $\bar{s}_P+\bar{t}_P \geq \bar{s}_Q + \bar{t}_Q$ and $\bar{s}_P + \bar{t}_P - \frac{\bar{g}_P}{l} \geq \bar{s}_Q + \bar{t}_Q - \frac{\bar{g}_Q}{l}$.*

Proof. $\bar{s}_P + \bar{t}_P \geq \bar{s}_Q + \bar{t}_Q$ is equivalent to $f_P(1,0) \geq f_Q(1,0)$, and $\bar{s}_P + \bar{t}_P - \frac{\bar{g}_P}{l} \geq \bar{s}_Q + \bar{t}_Q - \frac{\bar{g}_Q}{l}$ is equivalent to $f_P(1, \frac{1}{l}) \geq f_Q(1, \frac{1}{l})$, whereas $\bar{s}_P$ and $\bar{s}_Q$ are the gradients of $f_P(x,y)$ and $f_Q(x,y)$ in x-direction. Therefore the conditions on the right side are clearly sufficient. Conversely, $\bar{s}_P \geq \bar{s}_Q$ is necessary, as else $f_P(x,y) \geq f_Q(x,y)$ would be violated for x sufficiently large, whereas the other conditions are obviously necessary. □

We call $W \subset T$ a *worst case set* of paths if on the whole domain D_l

$$\max_{P\in W} f_P(x,y) = \max_{P\in T} f_P(x,y),$$

that is, W shows the same worst case behaviour as T. We represent a path by the sequence of the test results with 0 standing for a negative and 1 for a positive test result. Table 3.1 lists paths constituting a worst case set for all subalgorithms presented in the last section and for B_{2^k} with $k \geq 1$ together with the parameters of their planes. This is shown in the following Lemma.

Lemma 3.3 *For each subalgorithm the paths given in Table 3.1 constitute a worst case set.*

	path P	$\bar{s}_P$	$\bar{t}_P$	$\bar{g}_P$	path Q	$\bar{s}_Q$	$\bar{t}_Q$	$\bar{g}_Q$
B_{2^k}	$1\ldots1$	1	k	0				
O_3^2	01	1	2	2	1111	$\frac{1}{2}$	2	0
O_3^3	01	1	2	3	11011	$\frac{1}{2}$	$2\frac{1}{2}$	$\frac{1}{2}$
$O_{3,3,3}$	11110	1	$1\frac{2}{3}$	$\frac{1}{3}$	11111	$\frac{1}{2}$	$2\frac{1}{2}$	0
O_4^3	001	1	3	5	110110001	$\frac{1}{3}$	3	2
$O_{5,5,5}$	1011111	1	$2\frac{1}{3}$	$\frac{2}{3}$	111111110	$\frac{2}{3}$	3	$\frac{1}{3}$
$O_{11,11}$	1101101	1	$3\frac{1}{2}$	2	111111011	$\frac{1}{2}$	$4\frac{1}{2}$	3

Table 3.1: Worst case costs of subalgorithms

Proof. For each subalgorithm, the application of Lemma 3.2 shows that the planes of all other paths are below one of the listed paths on the whole domain. There is only one exception in subalgorithm $O_{5,5,5}$. Denote by R the path 11111111101. Then $\bar{s}_R = \frac{3}{4}$, $\bar{t}_R = 2\frac{3}{4}$, and $\bar{g}_R = \frac{1}{4}$. With $y \leq \frac{1}{5}$ follows $f_P(x,y) \geq f_R(x,y)$ for $x \geq 2$ and $f_Q(x,y) \geq f_R(x,y)$ for $x \leq 2$. Hence, the plane of R is below the maximum of the planes of P and Q from Table 3.1 on the whole domain D_5. □

3.5 Scaling up Subalgorithms

We can get subalgorithms for further test sizes by scaling up known subalgorithms. To do this, we choose a subalgorithm and $k \geq 1$. In all group tests we substitute 2^k items for each item. Thus, the size of all test groups and the overlap with existing contaminated groups is multiplied by 2^k. Whenever the original subalgorithm identifies a defective, the scaled up version uses B_{2^k} to identify a defective in the corresponding group of 2^k items using k tests. If we scale up a subalgorithm on contaminated groups of size l, say X_l, we get a subalgorithm on contaminated groups of size $2^k l$, which we denote by $X_l \times B_{2^k}$. For example, the binary tree representation of $O_6^6 = O_3^3 \times B_2$ is depicted in Figure 3.1. If P is a path in X_l, we denote by $P \times B_{2^k}$ the path in $X_l \times B_{2^k}$ represented by the sequence of test results from P with k positive results inserted for each occurrence of B_{2^k} in $X_l \times B_{2^k}$. Note that $(X_l \times B_{2^k}) \times B_{2^j} = X_l \times (B_{2^k} \times B_{2^j}) = X_l \times B_{2^{k+j}}$.

The following lemma describes the worst case behaviour of $X_l \times B_{2^k}$.

Lemma 3.4 *Let W be a worst case set of paths in the subalgorithm X_l. Then $W \times B_{2^k} = \{P \times B_{2^k} | P \in W\}$ is a worst case set of paths in $X_l \times B_{2^k}$. For each path P in X_l the plane of the path $P' = P \times B_{2^k}$ is determined by $\bar{s}_{P'} = \bar{s}_P$, $\bar{t}_{P'} = \bar{t}_P + k$, and $\bar{g}_{P'} = 2^k \bar{g}_P$.*

Proof. Along the path $P' = P \times B_{2^k}$ one defective is identified by the last test of each B_{2^k} that occurs in $X_l \times B_{2^k}$ exactly where X_l identifies a defective. Hence $d_{P'} = d_P$. The transition from P to P' does not change the number of contaminated groups of size l and $2^k l$ respectively that are present at the start and remain at the end of the path. Thus $\bar{s}_{P'} = \frac{s_{P'}}{d_{P'}} = \frac{s_P}{d_P} = \bar{s}_P$. The path P' contains k additional tests per identified defective compared to P. Therefore $\bar{t}_{P'} = \frac{t_{P'}}{d_{P'}} = \frac{t_P + k d_P}{d_P} = \bar{t}_P + k$. Good items are identified by tests with negative results only, whose test group sizes are all multiplied by 2^k in the transition from P to P'. The additional tests in P' all yield positive test results. Hence $\bar{g}_{P'} = \frac{g_{P'}}{d_{P'}} = \frac{2^k g_P}{d_P} = 2^k \bar{g}_P$.

Together, this leads to

$$\begin{aligned} f_{P'}(x,y) &= \bar{t}_{P'} + \bar{s}_{P'} x - \bar{g}_{P'} y \\ &= \bar{t}_P + k + \bar{s}_P x - 2^k \bar{g}_P y \\ &= f_P(x, 2^k y) + k. \end{aligned}$$

In this, $(x,y) \in D_{2^k l}$ if and only if $(x, 2^k y) \in D_l$. Denote by T and T' the set of all paths in X_l and $X_l \times B_{2^k}$, respectively. Then for all $(x,y) \in D_{2^k l}$

$$\begin{aligned} \max_{P' \in W \times B_{2^k}} f_{P'}(x,y) &= \max_{P \in W} f_P(x, 2^k y) + k \\ &= \max_{P \in T} f_P(x, 2^k y) + k \\ &= \max_{P' \in T \times B_{2^k}} f_{P'}(x,y) \\ &= \max_{P' \in T'} f_{P'}(x,y). \end{aligned}$$

The last equality is due to the fact that negative instead of positive test results in a B_{2^k} yield additional good items without changing anything else. Therefore, the cost of a path that contains such negative test results is always lower than the cost of the path from $T \times B_{2^k}$ whose test results in the B_{2^k} are always positive. Hence $W \times B_{2^k}$ is a worst case set. □

This method of scaling up X_l is superior to repeatedly halving groups of size $2^k l$ and applying X_l to the resulting groups of size l, because it requires the same number of tests but additionally scales up the number of good items identified.

It is possible to scale up a subalgorithm by multiples other than powers of two by using some other subalgorithm instead of B_{2^k} in the construction above. However, this seems to result in comparatively less efficient algorithms.

We indicate the scaling up of one of the subalgorithms introduced in the last section by multiplying all its indices by the scaling factor, for instance, $O_{10,10,10} = O_{5,5,5} \times B_2$ and $O_{12}^{12} = O_3^3 \times B_4$.

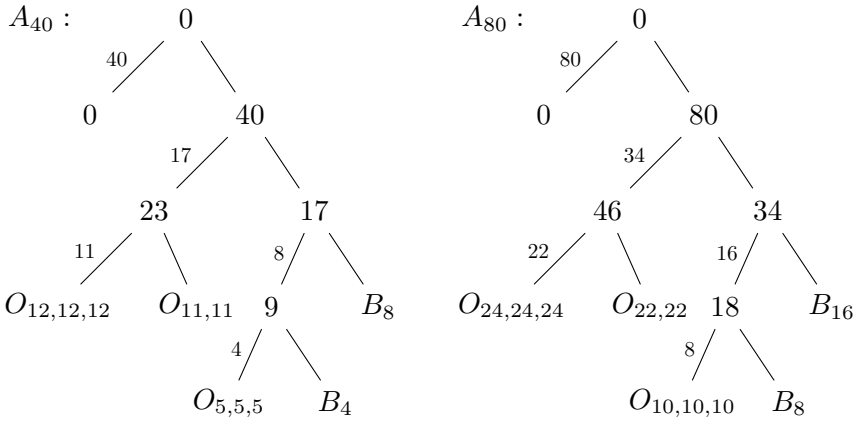


Figure 3.6: The procedures of A_{40} and A_{80}

3.6 Fixed Size Algorithms

We can combine subalgorithms in the following way. Assume that a contaminated group M of size m is known. Then we test a subset $L \subset M$ of size $l < m$. If the test result is positive, we continue with subalgorithm X_l on L. If the test result is negative, we have identified l good items and proceed with subalgorithm Y_{m-l} on $M \setminus L$. We denote this splitting of a contaminated group of size m by $S_m(X_l, Y_{m-l})$. By nesting these splittings we can build procedures for any start size m. If a subalgorithm, for instance, $O_{3,3,3}$, requires more than one group of size l, then the contaminated groups of size l are put aside until enough groups have been collected to execute the subalgorithm.

We denote by A_m an algorithm based on the best procedure with initial test group size m constructed in this way. Table 3.2 lists the procedures of A_m for selected initial test group sizes m with $m \leq 80$. The choice of these particular values for m is explained in Section 3.8. Procedures for $m \geq 80$ can be obtained by scaling up the procedures from Table 3.2 with $40 \leq m < 80$ in the same way as the subalgorithms in the last section. Then the procedure of $A_{2^k m} = A_m \times B_{2^k}$ with $k \geq 1$ has the same structure as the procedure of A_m with all test sizes multiplied by 2^k and each subalgorithm X_l substituted by $X_l \times B_{2^k}$.

A procedure can be represented as a binary tree. The root of the tree corresponds to the start, when no contaminated groups are known. At each node the size of the only contaminated group is given. Each leaf, except the one after a negative result in the first test, corresponds to the situation in which one contaminated group of some size l is known and is marked by the name of a subalgorithm that works on contaminated groups of size l. The binary tree representations of the procedures of A_{40}, A_{77}, and A_{80} are shown in Figures 3.6 and 3.7.

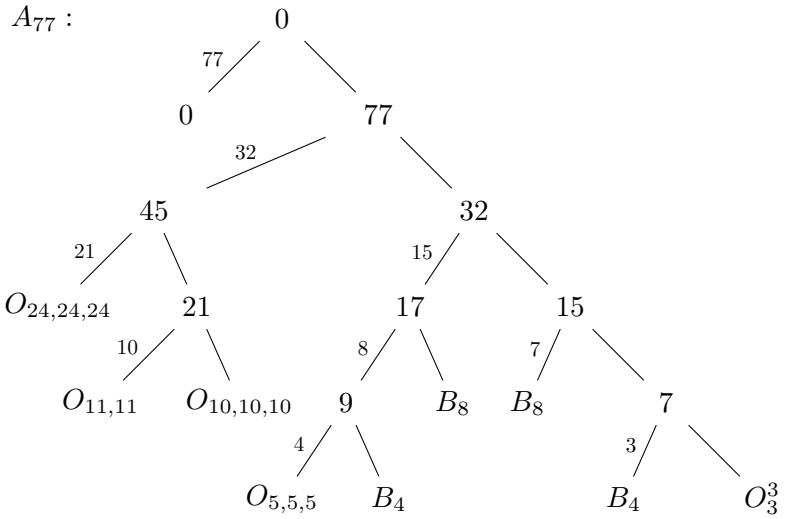


Figure 3.7: The procedure of A_{77}

We denote the set of all paths from the root to a leaf by T, excluding the path that consists of one negative test result. For the path $Q \in T$ we denote by t_Q the length of the path, that is, the number of tests performed, and by g_Q the number of good items identified in the process. Furthermore we use the notation defined in Section 3.4. Let $W(Q)$ denote the worst case set of the subalgorithm at the leaf of path Q that is based on Table 3.1 and Lemmas 3.3 and 3.4. We define the worst case cost c_m to identify a defective by

$$\begin{aligned} c_m &= \max_{Q \in T} \max_{P \in W(Q)} f_P\left(t_Q - \tfrac{g_Q}{m}, \tfrac{1}{m}\right) \\ &= \max_{Q \in T} \max_{P \in W(Q)} \bar{s}_P\left(t_Q - \tfrac{g_Q}{m}\right) + \bar{t}_P - \tfrac{\bar{g}_P}{m}. \end{aligned}$$

The values of c_m are listed in Table 3.2 for $m \leq 80$. The values of c_m for $m \geq 80$ can be obtained by the following Lemma.

Lemma 3.5 *For $m \geq 40$ and $k \geq 1$,*

$$c_{2^k m} = c_m + k.$$

Proof. Denote by Q and Q' paths in the binary tree representations of the procedures of A_m and $A_{2^k m} = A_m \times B_{2^k}$ respectively that describe the same sequence of test results. Then $t_{Q'} = t_Q$ and $g_{Q'} = 2^k g_Q$. For $P \in W(Q)$ and $P' = P \times B_{2^k}$ Lemma 3.4 provides $\bar{s}_{P'} = \bar{s}_P$, $\bar{t}_{P'} = \bar{t}_P + k$, and $\bar{g}_{P'} = 2^k \bar{g}_P$. Together, this results in $f_{P'}\left(t_{Q'} - \frac{g_{Q'}}{2^k m}, \frac{1}{2^k m}\right) = f_P\left(t_Q - \frac{g_Q}{m}, \frac{1}{m}\right) + k$. Substitution in the definition of $c_{2^k m}$ proves the lemma. □

m	c_m	n_m	l_m	procedure of A_m
1	1	1	1	B_1
2	2	2	2	B_2
3	$2\frac{1}{2}$	4	2	O_3^2
4	$2\frac{5}{6}$	14	3	O_4^3
7	$3\frac{4}{7}$	7	5	$S_7(O_3^3, B_4)$
12	$4\frac{1}{3}$	12	8	$S_{12}(O_{5,5,5}, S_7(O_3^3, B_4))$
15	$4\frac{28}{45}$	15	10	$S_{15}(O_6^6, S_9(B_4, O_{5,5,5}))$
19	5	19	16	$S_{19}(B_8, S_{11}(O_{5,5,5}, O_{6,6,6}))$
24	$5\frac{1}{3}$	24	16	$S_{24}(O_{10,10,10}, S_{14}(O_6^6, B_8))$
30	$5\frac{28}{45}$	30	20	$S_{30}(O_{12}^{12}, S_{18}(B_8, O_{10,10,10}))$
40	$6\frac{1}{40}$	40	32	$S_{40}(S_{17}(B_8, S_9(B_4, O_{5,5,5})), S_{23}(O_{11,11}, O_{12,12,12}))$
47	$6\frac{25}{94}$	47	32	$S_{47}(S_{20}(S_9(B_4, O_{5,5,5}), O_{11,11}),$ $S_{27}(O_{12,12,12}, S_{15}(S_7(O_3^3, B_4), B_8)))$
49	$6\frac{16}{49}$	49	32	$S_{49}(S_{21}(O_{10,10,10}, O_{11,11}), S_{28}(O_{12}^{12}, B_{16}))$
54	$6\frac{25}{54}$	54	35	$S_{54}(S_{23}(O_{11,11}, O_{12,12,12}),$ $S_{31}(S_{14}(O_6^6, B_8), S_{17}(B_8, S_9(B_4, O_{5,5,5}))))$
62	$6\frac{41}{62}$	62	43	$S_{62}(S_{25}(O_{12,12,12}, S_{13}(O_{6,6,6}, S_7(O_3^3, B_4))),$ $S_{37}(B_{16}, S_{21}(O_{10,10,10}, O_{11,11})))$
77	$6\frac{75}{77}$	77	62	$S_{77}(S_{32}(S_{15}(S_7(O_3^3, B_4), B_8), S_{17}(B_8, S_9(B_4, O_{5,5,5}))),$ $S_{45}(S_{21}(O_{10,10,10}, O_{11,11}), O_{24,24,24}))$
80	$7\frac{1}{40}$	80	64	$S_{80}(S_{34}(B_{16}, S_{18}(B_8, O_{10,10,10})), S_{46}(O_{22,22}, O_{24,24,24}))$

Table 3.2: The fixed size algorithms A_m for $m \leq 80$

We denote the maximum number of free items required by the procedure of A_m by n_m. In most procedures in Table 3.2, the free items required by a subalgorithm like O_3^3 can be drawn from items that belong to the m items of the initial test group but have become free again. This is not affected by scaling up a procedure. Therefore, in general $n_m = m$. The only exceptions are A_3 and A_4, for which $n_3 = 4$ and $n_4 = 14$ according to the description of O_3^2 and O_4^3 in Section 3.3.

When less than n_m free items are left, the procedure of A_m cannot be executed any more. Therefore, we continue by testing groups of size l_m and using B_{l_m} to identify a defective in the case of a positive test result where l_m is the largest integer such that the cost of identifying a defective is at most c_m, that is, not greater than for the procedure of A_m. This leads to

the definition

$$l_m = \max(2^{\lfloor c_m \rfloor - 1}, 2^{\lceil c_m \rceil - 1} - \lceil (\lceil c_m \rceil - c_m)m \rceil).$$

The values of l_m for $m \leq 80$ are listed in Table 3.2. As $l_m < m$ for $m > 2$, the cost of identifying a good item is greater with this method than by using the procedure of A_m.

In the following, we give a description of the *fixed size split and overlap algorithm* A_m where m can be any value for which a procedure is given in Table 3.2 or one of the values greater than or equal 40 from Table 3.2 multiplied by a power of two.

Algorithm A_m

Denote by I the set of n items. Always remove items from I when they are identified as good or defective. Denote by $\mathcal{C}$ a collection of contaminated groups that is initially empty.

Part 1

While I contains at least n_m free items,
  test a group M of m free items.
  If the result is positive,
    continue the procedure on M.
    If the final subalgorithm requires $k \geq 2$ contaminated groups of size l,
      add the contaminated group of size l to $\mathcal{C}$.
      If $\mathcal{C}$ contains k groups of size l,
        apply the subalgorithm and remove the k groups from $\mathcal{C}$.

Part 2

For each group L from $\mathcal{C}$,
  apply $B_{|L|}$ to L and remove L from $\mathcal{C}$.

Part 3

While $|I| \geq 1$,
  test a group of size $l = \min(l_m, |I|)$.
  If the result is positive,
    apply B_l to the contaminated group.

Part 1 is the main part of the algorithm, in which the procedure is applied repeatedly as long as enough free items are left. In Part 2, the contaminated groups left over from Part 1 are used up by identifying a defective in each group. Finally, Part 3 is just a simple nested algorithm to identify the small number of remaining items as good or defective.

The fixed size algorithm A_m does not use any knowledge about the number of defectives d.

3.7 Cost Estimate of Fixed Size Algorithms

The following theorem gives a bound for the number of tests needed by the fixed size algorithm A_m to solve the complete group testing problem.

Theorem 3.6 $M_{A_m}(d|n) < c_m d + \frac{1}{m}(n-d) + 5$.

This means that A_m needs a constant number of tests for the identification of each defective item, $\frac{1}{m}$ tests for each good item, and at most 5 tests due to restrictions near the end, when the number of unidentified items becomes small.

We derive an algorithm A'_m for the (d, n) group testing problem from A_m by simply omitting all tests whose result can be deduced in advance from the knowledge that there are exactly d defectives. Lemma 2.4 states that A'_m always needs at least one test less than A_m, leading to the following corollary of Theorem 3.6.

Corollary 3.7 $M_{A'_m}(d, n) < c_m d + \frac{1}{m}(n-d) + 4$.

For the proof of Theorem 3.6, we fix an arbitrary path S from the root to a leaf in the binary tree representation of the fixed size algorithm A_m. Denote by S_1, S_2, and S_3 the subpaths of this path that fall into Part 1, 2, and 3 of A_m, respectively. Let t_{S_i} be the length of S_i, and d_{S_i} and g_{S_i} the number of defective and good items identified along S_i. Denote by $\mathcal{C}_1$ the content of $\mathcal{C}$ at the end of Part 1 of A_m, that is, the collection of contaminated groups left over from Part 1.

An inspection of the procedures listed in Table 3.2 reveals that each subalgorithm used in the procedure of A_m that requires at least two groups of size l occurs only once in the procedure. Therefore there is exactly one path Q leading to this subalgorithm in the procedure. We denote by $c_{m,l} = t_Q - \frac{g_Q}{m}$ the cost to obtain a contaminated group of size l for this subalgorithm.

Lemmas 3.8, 3.9, and 3.10 estimate the number of tests needed in S_1, S_2, and S_3. Their proofs are all based on the partition of the path S at the points at which no contaminated groups except those in $\mathcal{C}$ are known. In Part 1 of A_m, each execution of the procedure of A_m forms a subpath in this partition except that each subalgorithm requiring at least two contaminated groups forms a separate subpath. In Part 2, each execution of a B_l is a subpath of the partition. In Part 3, each subpath is either a single test with negative result or a test with positive result together with the following execution of B_l. On each subpath in this partition the number of tests is estimated using the number of identified defective and good items and the cost of contaminated groups added to or removed from $\mathcal{C}$.

Lemma 3.8 $t_{S_1} \le c_m d_{S_1} + \frac{1}{m} g_{S_1} + \sum_{L \in \mathcal{C}_1} c_{m,|L|}$.

Proof. It suffices to show for all subpaths R in the partition of S_1 that

$$c_m d_R + \tfrac{1}{m} g_R + k_R c_{m,l} \geq t_R$$

where k_R denotes the number of contaminated groups of size l that are added to $\mathcal{C}$ minus the number that are removed from $\mathcal{C}$ along the subpath R. The estimations of the subpaths follow the description of the fixed size algorithm A_m.

If the first test result in the procedure is negative, m good items and no defectives are identified in one test. Since $0 + \frac{m}{m} = 1$, the above inequality is satisfied.

The rest of the proof treats the case that the first test result in the procedure is positive. Denote by Q the path in the binary tree representation of the procedure that corresponds to the test results.

If the subalgorithm at the end of path Q requires only one contaminated group, it is executed immediately. Denote by P the path followed in the subalgorithm, along which the contaminated group is always used up. Then $s_P = 1$, and d_P defectives and $g_Q + g_P$ good items are identified using $t_Q + t_P$ tests. This leads to the estimation

$$\begin{aligned} c_m d_P + \frac{g_Q + g_P}{m} &\geq \left(\bar{s}_P \left(t_Q - \frac{g_Q}{m}\right) + \bar{t}_P - \frac{\bar{g}_P}{m}\right) d_P + \frac{g_Q + g_P}{m} \\ &= s_P \left(t_Q - \frac{g_Q}{m}\right) + t_P - \frac{g_P}{m} + \frac{g_Q + g_P}{m} \\ &= t_Q + t_P. \end{aligned}$$

In this, the inequality $c_m \geq \bar{s}_P \left(t_Q - \frac{g_Q}{m}\right) + \bar{t}_P - \frac{\bar{g}_P}{m}$ follows from the definition of c_m in the last section.

On the other hand, if the subalgorithm at the end of path Q requires $k \geq 2$ contaminated groups of size l, the contaminated group of size l obtained along Q is added to the collection $\mathcal{C}$ of contaminated groups. Then g_Q good items and no defectives are identified using t_Q tests. Inserting the definition $c_{m,l} = t_Q - \frac{g_Q}{m}$ yields $0 + \frac{g_Q}{m} + c_{m,l} = t_Q$.

If k contaminated groups of size l are present after this, the subalgorithm is executed. Denote by P the path followed in the subalgorithm and by Q the path by which all k contaminated groups have been obtained. Then d_P defectives and g_P good items are identified using t_P tests, while s_P contaminated groups of size l are used up. This leads to the estimation

$$\begin{aligned} c_m d_P + \frac{g_P}{m} - s_P c_{m,l} &\geq \left(\bar{s}_P \left(t_Q - \frac{g_Q}{m}\right) + \bar{t}_P - \frac{\bar{g}_P}{m}\right) d_P \\ &\quad + \frac{g_P}{m} - s_P \left(t_Q - \frac{g_Q}{m}\right) \\ &= t_P. \end{aligned}$$

This process is repeated as long as enough free items are available. The addition of the inequalities for all subpaths in the partition of S_1 yields the statement of the lemma. □

Lemma 3.9 $t_{S_2} < c_m d_{S_2} + \frac{1}{m} g_{S_2} - \sum_{L \in \mathcal{C}_1} c_{m,|L|} + 3.25$.

Proof. For $L \in \mathcal{C}_1$ with $l = |L|$ denote by R the subpath of S_2 that contains exactly the tests in B_l on L. By Lemma 3.1 B_l needs at most $\lceil \log l \rceil$ tests, in which case at least $2^{\lceil \log l \rceil} - l$ good items are identified as well. All other cases lead to a lower overall cost, hence $t_R - \frac{g_R}{m} \leq \lceil \log l \rceil - \frac{2^{\lceil \log l \rceil} - l}{m}$. In all cases, exactly one defective is identified and the contaminated group L of size l is used up.

Let

$$a_{m,l} = \min \left(0, \lceil \log l \rceil - \frac{2^{\lceil \log l \rceil} - l}{m} + c_{m,l} - c_m \right)$$

denote the maximum additional cost incurred by identifying one defective using B_l on a contaminated group of size l instead of following the procedure of A_m. Summation over all $L \in \mathcal{C}_1$ yields

$$t_{S_2} - c_m d_{S_2} - \tfrac{1}{m} g_{S_2} + \sum_{L \in \mathcal{C}_1} c_{m,|L|} \leq \sum_{L \in \mathcal{C}_1} a_{m,|L|}.$$

It remains to show $\sum_{L \in \mathcal{C}_1} a_{m,|L|} < 3.25$.

Contaminated groups of size l can appear in $\mathcal{C}_1$ if and only if the procedure of A_m contains a path leading to a subalgorithm that processes $k \geq 2$ groups of size l. Then $\mathcal{C}_1$ can contain up to $k-1$ groups of size l, as k groups would have been eliminated in Part 1 of A_m by the application of the subalgorithm. For instance, the procedure of A_{77} depicted in Figure 3.7 uses subalgorithms $O_{5,5,5}$, $O_{10,10,10}$, $O_{11,11}$, and $O_{24,24,24}$, resulting in the estimation

$$\begin{aligned} \sum_{L \in \mathcal{C}_1} a_{77,|L|} &\leq 2a_{77,5} + 2a_{77,10} + a_{77,11} + 2a_{77,24} \\ &= 2 \cdot \tfrac{7}{11} + 2 \cdot \tfrac{41}{77} + \tfrac{32}{77} + 2 \cdot \tfrac{18}{77} \\ &= 3\tfrac{17}{77} \\ &< 3.25. \end{aligned}$$

A similar estimation shows $\sum_{L \in \mathcal{C}_1} a_{m,|L|} < 3.25$ for each m listed in Table 3.2. These estimations extend to all scaled up algorithms: For $m \geq 40$ and $k \geq 1$, Lemma 3.5 states $c_{2^k m} = c_m + k$, whereas the beginning of its proof shows $c_{2^k m, 2^k l} = c_{m,l}$. Inserting in the definition of $a_{m,l}$ leads to $a_{2^k m, 2^k l} = a_{m,l}$. □

Lemma 3.10 $t_{S_3} \leq c_m d_{S_3} + \frac{1}{m} g_{S_3} + 1.75$.

Proof. If the test on a group of size $l = \min(l_m, |I|)$ yields a positive result, denote by R the subpath of S_3 that contains this test and the tests belonging to the following execution of B_l. The estimation of the overall cost of R is the same as at the beginning of the proof of Lemma 3.9 plus one for the first test. Together with $l \leq l_m$ this results in

$$\begin{aligned}
t_R - \frac{g_R}{m} &\leq \lceil \log l \rceil + 1 - \frac{2^{\lceil \log l \rceil} - l}{m} \\
&\leq \lceil \log l_m \rceil + 1 - \frac{2^{\lceil \log l_m \rceil} - l_m}{m} \\
&= \max \left(\lfloor c_m \rfloor - 1 + 1 - \frac{2^{\lfloor c_m \rfloor - 1} - 2^{\lfloor c_m \rfloor - 1}}{m}, \right. \\
&\qquad \left. \lceil c_m \rceil - 1 + 1 - \frac{2^{\lceil c_m \rceil - 1} - 2^{\lceil c_m \rceil - 1} + \lceil (\lceil c_m \rceil - c_m)\, m \rceil}{m} \right) \\
&\leq \max \left(\lfloor c_m \rfloor, \lceil c_m \rceil - \frac{(\lceil c_m \rceil - c_m)\, m}{m} \right) \\
&\leq c_m.
\end{aligned}$$

On the other hand, denote by $\mathcal{R}$ the set of all subpaths of S_3 that consist of a single test of a group of size $l = \min(l_m, |I|)$ with negative result. All but the last of the tests in $\mathcal{R}$ are on groups of size l_m, as for $|I| < l_m$ all remaining items are tested and identified as good. Denote by I_1 and I_2 the items remaining in I at the end of Part 1 and 2, respectively. Then $|\mathcal{R}| \leq \frac{|I_2|}{l_m} \leq \frac{|I_1|}{l_m}$. The set I_1 contains less than n_m free items and the items in the groups in $\mathcal{C}_1$. Therefore

$$|I_1| < n_m + \sum_{L \in \mathcal{C}_1} |L| \leq \begin{cases} 3l_m & \text{for } m \neq 4 \\ 5l_m & \text{for } m = 4. \end{cases}$$

It can easily be checked that the last inequality holds for all m in Table 3.2, which extends to $m \geq 80$ because for $k \geq 1$

$$\begin{aligned}
l_{2^k m} &= \max \left(2^{\lfloor c_m + k \rfloor - 1}, 2^{\lceil c_m + k \rceil - 1} - \left\lceil (\lceil c_m + k \rceil - (c_m + k))\, 2^k m \right\rceil \right) \\
&= \max \left(2^k 2^{\lfloor c_m \rfloor - 1}, 2^k 2^{\lceil c_m \rceil - 1} - \left\lceil 2^k (\lceil c_m \rceil - c_m)\, m \right\rceil \right) \\
&\geq 2^k l_m.
\end{aligned}$$

The inequality $l_m > \frac{5}{8}m$ can be checked in the same way.

Together this yields $|\mathcal{R}| \leq 3$ for $m \neq 4$ and $|\mathcal{R}| \leq 5$ for $m = 4$, leading to

$$\sum_{R \in \mathcal{R}} t_R - \frac{g_R}{m} \leq \begin{cases} 3 - \frac{2l_m + 1}{m} < 1.75 & \text{for } m \neq 4 \\ 5 - \frac{4 \cdot 3 + 1}{4} = 1.75 & \text{for } m = 4. \end{cases}$$

Adding the estimation from above for all subpaths of S_3 beginning with a positive test result yields the statement of the lemma. □

Proof of Theorem 3.6. For each path S of A_m that identifies d defectives in n items the addition of the inequalities of Lemmas 3.8, 3.9, and 3.10 yields

$$t_S < c_m d + \tfrac{1}{m}(n-d) + 5.$$

Maximizing over all paths yields the theorem. □

An analysis of the arguments above for a path that achieves equality in Lemma 3.8 shows that $M_{A_m}(d|n) > c_m d + \frac{1}{m}(n-d) - 1$ for $\frac{n}{d} \geq m$. For sufficiently small $\frac{n}{d}$, some algorithms A_m need fewer tests than this in the worst case, as there are not enough good items to follow a path achieving equality in Lemma 3.8. However, in the main algorithm A in the next section A_m is used only for $\frac{n}{d} \geq m$.

A possible modification of Part 2 of the fixed size algorithm A_m is to use subalgorithms that require multiple contaminated groups not necessarily of the same size to identify several defectives together. This leads to a lower constant in Lemma 3.9 and therefore in Theorem 3.6.

3.8 Main Algorithm

We suppose $d > 0$ and denote by $r = \frac{n}{d}$ the initial ratio of all items to defective items. To find the best fixed size algorithm for a given ratio r, we compare the number of tests needed by different algorithms in the worst case. The difference between the upper bounds from Theorem 3.6 for $M_{A_{m'}}(d|n)$ and $M_{A_m}(d|n)$ with $m \neq m'$ is

$$(c_{m'} - c_m)\, d + \left(\tfrac{1}{m'} - \tfrac{1}{m}\right)(n-d) = d\left(c_{m'} - c_m - \frac{m'-m}{mm'}(r-1)\right).$$

Let

$$r_{m,m'} = \frac{mm'}{m'-m}\,(c_{m'} - c_m) + 1$$

denote the value of r for which the above expression is zero, that is, the upper bounds for $M_{A_{m'}}(d|n)$ and $M_{A_m}(d|n)$ are equal.

For $r \geq m$ the difference between $M_{A_m}(d|n)$ and the upper bound from Theorem 3.6 is less than 6. Thus, if we suppose $m' > m$, then for each $r \geq m$ there exists $d_0(r)$ such that for all $d \geq d_0(r)$ and $n = rd$

$$M_{A_m}(d|n) < M_{A_{m'}}(d|n) \text{ for } r < r_{m,m'} \text{ and}$$
$$M_{A_m}(d|n) > M_{A_{m'}}(d|n) \text{ for } r > r_{m,m'}.$$

Therefore we can describe the range of ratios r for which A_m is best by

$$r_m^{\min} = \max_{m'<m} r_{m',m} \text{ and}$$
$$r_m^{\max} = \min_{m'>m} r_{m,m'},$$

m	c_m	$r_m^{\min}$ $r_m^{\max}$	$\alpha(r)$	$\bar{\alpha}(r)$	m	c_m	$r_m^{\min}$ $r_m^{\max}$	$\alpha(r)$	$\bar{\alpha}(r)$
1	1				24	$5\frac{1}{3}$			
		3	0.246	0.558			$35\frac{2}{3}$	0.199	0.220
2	2				30	$5\frac{28}{45}$			
		4	0.255	0.473			$49\frac{1}{3}$	0.181	0.196
3	$2\frac{1}{2}$				40	$6\frac{1}{40}$			
		5	0.224	0.391			$65\frac{5}{7}$	0.174	0.185
4	$2\frac{5}{6}$				47	$6\frac{25}{94}$			
		$7\frac{8}{9}$	0.229	0.329			$70\frac{3}{4}$	0.173	0.184
7	$3\frac{4}{7}$				49	$6\frac{16}{49}$			
		$13\frac{4}{5}$	0.225	0.280			$73\frac{1}{5}$	0.174	0.184
12	$4\frac{1}{3}$				54	$6\frac{25}{54}$			
		$18\frac{1}{3}$	0.179	0.220			84	0.174	0.183
15	$4\frac{28}{45}$				62	$6\frac{41}{62}$			
		$27\frac{11}{12}$	0.198	0.224			$100\frac{8}{15}$	0.180	0.187
19	5				77	$6\frac{75}{77}$			
		$31\frac{2}{5}$	0.208	0.232			$105\frac{2}{3}$	0.175	0.181
24	$5\frac{1}{3}$				80	$7\frac{1}{40}$			

Table 3.3: Ranges of A_m in the main algorithm A for $m \leq 80$

except $r_1^{\min} = 0$. There is always some $m' > m$ with $r_m^{\max} = r_{m'}^{\min}$. For $m \leq 80$, the values $r_m^{\min}$ and $r_m^{\max}$ can be found in Table 3.3 together with α and $\bar{\alpha}$, which are defined in the next section. For $m \geq 40$ and $k \geq 1$, substituting $c_{2^k m} = c_m + k$ from Lemma 3.5 leads to $r_{2^k m, 2^k m'} - 1 = 2^k \left(r_{m,m'} - 1\right)$. Thus

$$r_{2^k m}^{\min} - 1 = 2^k \left(r_m^{\min} - 1\right) \text{ for } m \geq 47, \text{ and}$$
$$r_{2^k m}^{\max} - 1 = 2^k \left(r_m^{\max} - 1\right) \text{ for } m \geq 40.$$

This leads to the following *split and overlap algorithm* A for the complete group testing problem that always uses the best fixed size algorithm A_m for the actual ratio r. We refer to A in the following as the *main algorithm* to highlight the difference to the fixed size algorithm A_m.

Algorithm A

If $d = 0$,
  test the group of all items.
  If the result is negative, then stop, else set $d := 1$.
Let $r = \frac{n}{d}$.
Choose m satisfying $r_m^{\min} < r \leq r_m^{\max}$.
Execute A_m.

In the main algorithm A the initial test group size m, which is used if no contaminated group is known, remains constant even if the ratio of good to defective unidentified items changes during the execution. In contrast, the generalized binary splitting algorithm G from Hwang presented in Section 3.1 adapts the test group size to the ratio of remaining good to defective

items. Surprisingly, this does not seem to be necessary to obtain a good algorithm for the worst case scenario.

The list of fixed size algorithms A_m in Tables 3.2 and 3.3 is the result of a computation that searches for the best algorithms for each $m \geq 1$ using all subalgorithms and techniques introduced in this chapter, that is, those with the lowest cost c_m for identifying a defective. Then the above definitions can be extended to all $m \geq 1$, and the algorithms listed in Tables 3.2 and 3.3 are just those with $r_m^{\min} < r_m^{\max}$. For example, A_5 with the procedure $S_5(B_2, O_3^3)$ and A_6 with the procedure O_6^4 are made redundant by A_4 and A_7. It is possible to continue this process for $m > 80$ instead of scaling up algorithms for smaller m, but this yields only minor further improvements.

3.9 Cost Estimate of Main Algorithm

The minimum number of tests needed for the (d, n) group testing problem is known only for the trivial cases $M(0, n) = M(n, n) = 0$ and for a large proportion of defectives, that is, a low ratio $r = \frac{n}{d}$ of all items to defectives. Presume $0 < d < n$. For $r \leq \frac{21}{8} = 2.625$, Du and Hwang [8] showed $M(d, n) = n - 1$, which can be reached by testing all items individually. Leu, Lin, and Weng [14] extended this to $r \leq \frac{43}{16} = 2.687\ldots$ for $d \geq 193$, Riccio and Colbourn [16] to $r < \log_{\frac{3}{2}} 3 = 2.709\ldots$ for sufficiently large d depending on r. All these results are based on a Lemma by Hu, Hwang, and Wang [10], who also conjectured $M(d, n) = n - 1$ for $r \leq 3$ and proved $M(d, n) < n - 1$ for $r > 3$. The latter is achieved by a variant of the fixed size algorithm A_2 omitting all predictable tests, as shown by Du and Hwang [6, Section 3.5]. However, the proof of the conjecture remains elusive and calls for a different approach.

These results transfer to the complete group testing problem by Theorem 2.1, which states $M(d|n) = M(d, n) + 1$ for $0 \leq d < n$. For $d = n$, all items have to be tested individually, thus $M(n|n) = n$.

The main algorithm A is optimal in all these cases: Applied to the complete group testing problem, it performs only one test on the group of all defectives for $d = 0$ and tests all items individually for $r \leq 3$.

The only general lower bound that is known for the (d, n) group testing problem is the information lower bound $\lceil \log \binom{n}{d} \rceil$. Therefore we compare the number of tests needed by A with $\log \binom{n}{d}$. First we prove some lemmas.

Lemma 3.11 *For $x > 0$,*

$$\left(\frac{x+1}{x}\right)^x \leq e \leq \left(\frac{x+1}{x}\right)^{x+1}.$$

Proof. The statement of the lemma is equivalent to

$$x \ln\left(1 + \frac{1}{x}\right) \leq 1 \leq (x+1) \ln\left(1 + \frac{1}{x}\right).$$

Application of Taylor's formula yields for some ξ with $1 \leq \xi \leq 1 + \frac{1}{x}$

$$x \ln\left(1 + \frac{1}{x}\right) = x\frac{1}{\xi x} = \frac{1}{\xi} \leq 1$$

and

$$(x+1)\ln\left(1 + \frac{1}{x}\right) = (x+1)\frac{1}{\xi x} = \frac{1}{\xi}\left(1 + \frac{1}{x}\right) \geq 1.$$

□

Lemma 3.12 *For* $0 < d \leq \frac{n}{2}$,

$$\log\binom{n}{d} > d\log\left(r\left(\frac{r}{r-1}\right)^{r-1}\right) - \frac{1}{2}\log d - 1.5.$$

Proof. For $d = 1$ the application of Lemma 3.11 with $x = r - 1$ yields

$$\begin{aligned}\log\left(r\left(\frac{r}{r-1}\right)^{r-1}\right) - 1.5 &\leq \log r + \log e - 1.5 \\ &< \log\binom{n}{1}.\end{aligned}$$

For $d \geq 2$ the application of Stirling's Formula $k! = \sqrt{2\pi k}\left(\frac{k}{e}\right)^k \theta_k$ with $1 < \theta_k < e^{\frac{1}{12k}}$, see Bollobás [1, Chapter VII §1], results in

$$\begin{aligned}\log\binom{n}{d} &= \log\frac{(rd)!}{((r-1)d)!d!} \\ &= \log\left(\sqrt{\frac{r}{2\pi(r-1)d}}\left(\frac{r^r}{(r-1)^{r-1}}\right)^d \frac{\theta_{rd}}{\theta_{(r-1)d}\theta_d}\right) \\ &= d\log\left(r\left(\frac{r}{r-1}\right)^{r-1}\right) - \frac{1}{2}\log d - \frac{1}{2}\log 2\pi \\ &\quad + \frac{1}{2}\log\frac{r}{r-1} + \log\theta_{rd} - \log\theta_{(r-1)d} - \log\theta_d.\end{aligned}$$

Leaving out the terms $\frac{1}{2}\log\frac{r}{r-1}$ and $\log\theta_{rd}$, which are always positive, and considering $d \geq 2$ and $r \geq 2$ in the next step leads to

$$\begin{aligned}\log\binom{n}{d} &> d\log\left(r\left(\frac{r}{r-1}\right)^{r-1}\right) - \frac{1}{2}\log d - \frac{1}{2}\log 2\pi \\ &\quad - \left(\frac{1}{12(r-1)d} + \frac{1}{12d}\right)\log e \\ &\geq d\log\left(r\left(\frac{r}{r-1}\right)^{r-1}\right) - \frac{1}{2}\log d - \frac{1}{2}\log 2\pi - \frac{1}{12}\log e \\ &> d\log\left(r\left(\frac{r}{r-1}\right)^{r-1}\right) - \frac{1}{2}\log d - 1.5.\end{aligned}$$

□

It can be shown by similar estimates that the difference between the two sides of the inequality in Lemma 3.12 is less than one.

We denote by

$$\alpha_m(r) = c_m + \frac{r-1}{m} - \log\left(r\left(\frac{r}{r-1}\right)^{r-1}\right)$$

the average number of tests per defective by which the fixed size algorithm A_m exceeds the information lower bound for large d, which we call the *loss per defective*. Similarly,

$$\alpha(r) = \min_m \alpha_m(r)$$

denotes the loss per defective of the main algorithm A. Figure 3.8 shows a graph of $\alpha(r)$ and $\alpha_m(r)$.

In addition, let

$$\bar{\alpha}(r) = \min_m \left(c_m + \frac{r-1}{m}\right) - \log\left((r-1)e\right)$$

denote the upper bound of the loss per defective of all scaled up versions of the fixed size algorithm used for the ratio r, as shown by the following Lemma.

Lemma 3.13 *For $r \geq r_{47}^{\min}$ and $k \geq 1$,*

$$\alpha(2^k(r-1)+1) \leq \bar{\alpha}(r).$$

Proof. Choose $m \geq 47$ satisfying $r_m^{\min} < r \leq r_m^{\max}$. From the last section follows $r_{2^k m}^{\min} < 2^k(r-1)+1 \leq r_{2^k m}^{\max}$. Inserting in the definition of $\alpha_m(r)$

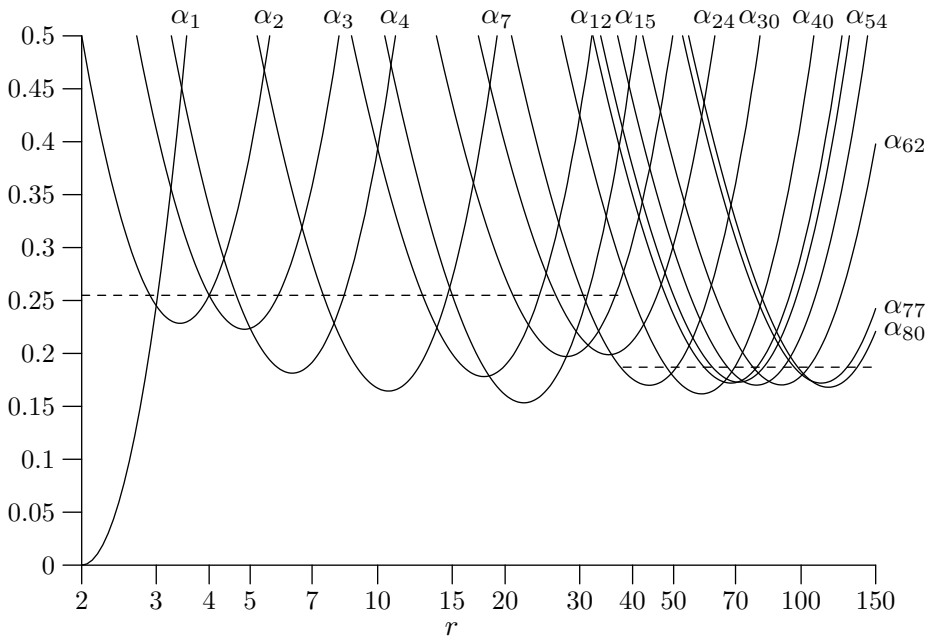


Figure 3.8: The loss per defective $\alpha(r)$

using $r\left(\frac{r}{r-1}\right)^{r-1} = (r-1)\left(\frac{r}{r-1}\right)^{r}$ then yields

$$\begin{aligned}\alpha(2^k(r-1)+1) &= \alpha_{2^k m}(2^k(r-1)+1)\\ &= c_{2^k m} + \frac{2^k(r-1)}{2^k m}\\ &\quad - \log\left(2^k(r-1)\left(\frac{2^k(r-1)+1}{2^k(r-1)}\right)^{2^k(r-1)+1}\right).\end{aligned}$$

The application of Lemma 3.5, which states $c_{2^k m} = c_m + k$, and Lemma 3.11 with $x = 2^k(r-1)$ results in

$$\begin{aligned}\alpha(2^k(r-1)+1) &\le c_m + k + \frac{r-1}{m} - \log\left(2^k(r-1)e\right)\\ &= \bar{\alpha}(r).\end{aligned}$$

□

The following theorem gives an upper bound for the loss per defective $\alpha(r)$.

Theorem 3.14 $\alpha(r) \le \frac{1}{2} - \log\frac{32}{27} < 0.255$ *for* $r \ge 2$ *and*
$\alpha(r) < 0.187$ *for* $r \ge 38$.

Proof. $\alpha_m(r)$ is convex, as $\alpha_m''(r) = \frac{\log e}{r(r-1)} > 0$ for $r > 1$. Therefore $\alpha(r)$ is convex between $r_m^{\min}$ and $r_m^{\max}$ for all m, implying

$$\alpha(r) \le \max\left(\alpha(r_m^{\min}), \alpha(r_m^{\max})\right) \text{ for } r_m^{\min} \le r \le r_m^{\max}.$$

For $m \le 80$ Table 3.3 lists $\alpha(r_m^{\min})$, whereas for $47 \le m \le 80$ and $k \ge 1$ by Lemma 3.13 $\alpha(r_{2^k m}^{\min}) \le \bar{\alpha}(r_m^{\min})$, which is also listed in Table 3.3. Since $r_m^{\max} = r_{m'}^{\min}$ for some $m' > m$, this extends to $\alpha(r_m^{\max})$.

This leads to $\alpha(r) \le \alpha(4) = \frac{1}{2} - \log\frac{32}{27}$. Together with $\alpha(38) < 0.185$ follows $\alpha(r) < 0.187$ for $r \ge 38$. □

The bounds for $\alpha(r)$ given in Theorem 3.14 are shown in Figure 3.8 as a dashed line.

A simple calculation shows that $\alpha_m(r)$ assumes its minimum at $\frac{1}{2^{\frac{1}{m}}-1}+1$. The lowest minimum value of $\alpha_m(r)$ with $m \ge 2$ is the minimum of $\alpha(r)$ for $r \ge 2$:

$$\min_{r \ge 2} \alpha(r) = \alpha_{15}(22.1) < 0.154.$$

Now we can estimate the difference between the number of tests required by the split and overlap algorithm A for the complete group testing problem and $\log\binom{n}{d}$.

Theorem 3.15 *For* $0 < d \le \frac{n}{2}$,

$$M_A(d|n) - \log\binom{n}{d} < \alpha(r)\, d + \frac{1}{2}\log d + 6.5.$$

Proof. Assume that the main algorithm A chooses the fixed size algorithm A_m with the initial test group size m. Applying Theorem 3.6 and Lemma 3.12 yields

$$\begin{aligned} M_A(d|n) - \log\binom{n}{d} &= M_{A_m}(d|n) - \log\binom{n}{d} \\ &< c_m d + \tfrac{1}{m}(n-d) + 5 \\ &\quad - d\log\left(r\left(\frac{r}{r-1}\right)^{r-1}\right) + \frac{1}{2}\log d + 1.5 \\ &= \alpha_m(r) d + \frac{1}{2}\log d + 6.5 \\ &= \alpha(r)\, d + \frac{1}{2}\log d + 6.5. \end{aligned}$$

□

As with the fixed size algorithm A_m in Section 3.7, we can derive from A an algorithm A' for the (d, n) group testing problem by simply omitting all

tests that become predictable due to the knowledge that there are exactly d defectives. From Lemma 2.4 then follows $M_{A'}(d,n) \leq M_A(d|n) - 1$, leading to the following corollary of Theorem 3.15.

Corollary 3.16 *For* $0 < d \leq \frac{n}{2}$,

$$M_{A'}(d,n) - \log \binom{n}{d} < \alpha\,(r)\,d + \frac{1}{2}\log d + 5.5.$$

For $d \geq 10$, this is considerably better than the $d-1$ additional tests for Hwang's generalized binary splitting algorithm G presented in Section 3.1.

The difference between the two sides of the inequalities in Theorem 3.15 and Corollary 3.16 is less than 7, as the corresponding differences in Theorem 3.6 and Lemma 3.12, which are used in the proof of Theorem 3.15, are less than 6 and 1, respectively. This leads to the following corollary.

Corollary 3.17 *For* $r \geq 2$,

$$\lim_{\substack{n,d\to\infty \\ \frac{n}{d}\to r}} \frac{1}{d}\left(M_{A'}(d,n) - \log \binom{n}{d}\right) = \alpha(r).$$

For small r the necessity of integral test sizes restricts significantly the choice of good algorithms, motivating the following conjecture.

Conjecture 3.18 *For* $2 \leq r \leq 4$,

$$\lim_{\substack{n,d\to\infty \\ \frac{n}{d}\to r}} \frac{1}{d}\left(M(d,n) - \log \binom{n}{d}\right) = \alpha(r).$$

This implies that the number of tests per defective needed by A_2 and A_3 is optimal and that there exists no fixed size algorithm with initial test group size 4 requiring less than $2\frac{3}{4}$ tests per defective. Furthermore, the conjecture implies that the general upper bound $\frac{1}{2} - \log\frac{32}{27}$ for $\alpha(r)$ given in Theorem 3.14 is the best possible for any group testing algorithm.

Bibliography

[1] B. Bollobás, *Graph Theory* (Springer, New York 1979).

[2] X. M. Chang, F. K. Hwang, and J. F. Weng, Group testing with two and three defectives, *Ann. N. Y. Acad. Sci.* 576 (1989) 86-96.

[3] P. Chen, L. Hsu, and M. Sobel, Entropy-based optimal group-testing procedures, *Probab. Engrg. Inform. Sci.* 1 (1987) 497-509.

[4] C. J. Colbourn, Group testing, *The CRC handbook of combinatorial designs* (CRC Press, Boca Raton, CA 1996) 564-565.

[5] R. Dorfman, The detection of defective members of large populations, *Ann. Math. Statist.* 14 (1943) 436-440.

[6] D. Z. Du and F. K. Hwang, *Combinatorial group testing and its applications*, 2nd ed. (World Scientific, Singapore 2000).

[7] D. Z. Du and F. K. Hwang, Competitive group testing, *Disc. Appl. Math.* 45 (1993) 221-232.

[8] D. Z. Du and F. K. Hwang, Minimizing a combinatorial function, *SIAM J. Alg. Disc. Methods* 3 (1982) 523-528.

[9] L. Hsu, New procedures for group-testing based on the Huffman lower bound and Shannon entropy criteria, *Adaptive Designs (South Hadley, MA 1992), IMS Lecture Notes Monogr. Ser.* 25 (Inst. Math. Statist., Hayward, CA 1995) 249-262.

[10] M. C. Hu, F. K. Hwang, and J. K. Wang, A boundary problem for group testing, *SIAM J. Alg. Disc. Methods* 2 (1981) 81-87.

[11] F. K. Hwang, A method for detecting all defective members in a population by group testing, *J. Amer. Statist. Assoc.* 67 (1972) 605-608.

[12] F. K. Hwang, An optimum nested procedure in binomial group testing, *Biometrics* 32 (1976) 939-943.

[13] F. K. Hwang, T. T. Song, and D. Z. Du, Hypergeometric and generalized hypergeometric group testing, *SIAM J. Alg. Disc. Methods* 2 (1981) 426-428.

[14] M. G. Leu, C. Y. Lin, and S. Y. Weng, Note on a conjecture for group testing, *Ars Combin.* 64 (2002) 29-32.

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

[16] L. Riccio and C. J. Colbourn, Sharper bounds in adaptive group testing, *Taiwanese J. Math.* 4 (2000) 669-673.

[17] J. Schlaghoff and E. Triesch, Improved results for competitive group testing, *Combin. Probab. Comput.*, to appear.

[18] M. Sobel, Optimal group testing, *Proc. Colloquium on Information Theory (Debrecen, 1967)* (János Bolyai Math. Soc., Budapest 1968) 411-488.

[19] M. Sobel and P. A. Groll, Group testing to eliminate efficiently all defectives in a binomial sample, *Bell System Tech. J.* 38 (1959) 1179-1252.

[20] E. Triesch, A group testing problem for hypergraphs of bounded rank, *Disc. Appl. Math.* 66 (1996) 185-188.

Lebenslauf

Name:	Andreas Allemann
Geburtsdatum:	9.8.1969
Geburtsort:	Bonn
Familienstand:	verheiratet
Nationalität:	schweizerisch

Schulbildung:

08/1975-06/1979	Grundschule Bonn-Venusberg
08/1979-06/1988	Clara-Schumann-Gymnasium Bonn
06/1988	Abitur

Studium:

10/1988-03/1997	Studium der Mathematik mit Nebenfach Informatik an der Rheinischen Friedrich-Wilhelms-Universität Bonn
03/1997	Diplom

Berufliche Tätigkeiten:

07/1993-09/1994	Werkstudent bei IBM in Köln, Bonn und Böblingen
04/1997-12/2001	Freier Mitarbeiter bei der Unternehmensberatung inform. AG in Köln