Efficient Parallel Algorithms for Dead Sensor Diagnosis and Multiple Access Channels

Michael T. Goodrich and Daniel S. Hirschberg
Dept. of Computer Science
University of California, Irvine
Irvine, CA 92697-3435
goodrich(at)ics.uci.edu, dan(at)ics.uci.edu

ABSTRACT

We study parallel algorithms for identifying the dead sensors in a mobile ad hoc wireless network and for resolving broadcast conflicts on a multiple access channel (MAC). Our approach involves the development and application of new group-testing algorithms, where we are asked to identify all the defective items in a set of items when we can test arbitrary subsets of items. In the standard group-testing problem, the result of a test is binary—the tested subset either contains defective items or not. In the versions we study in this paper, the result of each test is non-binary. For example, it may indicate whether the number of defective items contained in the tested subset is zero, one, or at least two (i.e., the results are 0, 1, or 2+). We give adaptive algorithms that are provably more efficient than previous group testing algorithms (even for generalized response models). We also show how our algorithms can be implemented in parallel, because they possess a property we call conciseness, which allows them to be used to solve dead sensor diagnosis and conflict resolution on a MAC. Dead sensor diagnosis poses an interesting challenge compared to MAC resolution, because dead sensors are not locally detectable, nor are they themselves active participants. Even so, we present algorithms that can be applied in both contexts that are more efficient than previous methods. We also give lower bounds for generalized group testing.

Categories and Subject Descriptors

D.4.4 [**Operating Systems**]: Communications Management – Ethernet; F.2.2 [**Analysis of Algorithms and Problem Complexity**]: Nonnumerical Algorithms and Problems; G.2.1 [**Discrete Mathematics**]: Combinatorics – Combinatorial algorithms.

General Terms

Algorithms, Performance, Design

Keywords

group testing, multiple access channels, dead sensor diagnosis

SPAA'06, July 30–August 2, 2006, Cambridge, Massachusetts, USA.

1. INTRODUCTION

Wireless communication has renewed interest in parallel algorithms for dealing with conflicts and failures among collections of communicating devices. For example, when a collection of wireless devices compete to communicate with a particular access point, the access point becomes a multiple access channel (MAC), which requires a conflict-resolution method to allow all devices to send their packets in a timely manner. In large deployments, the need for conflict resolution among devices may be further complicated by their physical distribution, as the devices may form an ad hoc wireless network. The traditional way a base station communicates with devices in an ad hoc network is via broadcast-and-respond protocols [17], which have a simple structure: Messages are broadcast from a base station to the n sensors in such a network using a simple flooding algorithm (e.g., see [19]) and responses to this message are aggregated back along the spanning tree that is formed by this broadcast. Because the flooding algorithm is topology-discovering, the spanning tree defined by the flooding algorithm can be different with each broadcast. This mutability property is particularly useful for mobile sensors, since their network adjacencies can change over time, although we assume they are not moving so fast that the topology of the spanning tree defined by a broadcast changes before the aggregate response from the broadcast is received back at the base station. A new challenge arises in this context, however, when devices fail (e.g., by running down their batteries) and we wish to efficiently determine the identities of the dead sensors.

1.1 Group Testing

In this paper, we present and analyze new algorithms for group testing, showing how they can be implemented in parallel to solve conflict resolution in MACs and dead sensor diagnosis. In the group testing problem, we are given a set of n items, d of which are defective (bounds on the value of d may or may not be known, depending on the context). A *configuration* specifies which of the items are defective. Thus, there are $\binom{n}{d}$ configurations of d defectives among the n items. To determine which of the n items are defective, we are allowed to sample from the items so as to define arbitrary subsets that can be tested for contamination. In the *standard* group testing problem, each test returns one of two values—either the subset contains no defectives or it contains at least one defective. Therefore, there is an information theoretic lower bound of

$$\lg \binom{n}{d} \approx d \lg n$$

tests, in the worst case, for any binary-result group testing algorithm.

Motivated by the application mentioned above, we consider generalizations of the standard group testing problem, where there can be three or more possible results of a contamination test. In *ternary-result* group testing a result indicates whether the subset contains no defectives, one defective, or at least two defectives (i.e., the results are 0, 1, or 2+). Generalizing further, we may allow for *counting* tests that return the exact number of defective items present in the test. In either case, a one-defective result may either be *identifying*, returning a unique identifier of the defective item, or *anonymous*, indicating, but not identifying, that there is one defective item in the test. We are interested in efficient algorithms and lower bounds for generalized group testing, as well as useful applications for such results.

1.2 Multiple Access Channels

In the *multiple access channel* problem [6, 12, 13, 14] we have a set S of n devices that share a communication channel such that a subset D of d of the devices wish to use the channel to transmit a data packet. In any time slice, some subset T of the devices S may attempt a transmission on the channel. If there is only one remaining device x from D in T, then it succeeds (and all parties learn the identity of x). Alternatively, if no device attempts to transmit, then all parties learn this as well. But if two or more devices attempt to transmit, then all parties learn only that a conflict has occurred (and no transmission is successful during this time slice).

1.3 Dead Sensor Diagnosis

In the *dead sensor diagnosis* problem, we are given an ad hoc network of n sensors, which can communicate with a base station using a broadcast-and-respond protocol along a broadcast tree that may be different with each broadcast. Furthermore, we have that d of the sensors have failed (e.g., d batteries may have died, but we do not necessarily know the value of d), and we wish to identify which sensors are dead. The challenge posed by this problem is complicated by the dynamic nature of the mobile sensors, since there is no local way to detect dead sensors—they simply become invisible to the sensors around them and there is no local way to distinguish this bad event from the common event of a live sensor moving out of range of a set of its former neighbors.

Of course, the group controller could send out n broadcasts, each of which asks an individual sensor to send a "heartbeat" acknowledgment message back as a response. Assuming a reasonable time-out condition for non-responding sensors, this naive solution to the dead sensor diagnosis problem could identify the dead sensors using a total of $O(n^2)$ messages spread across n communication rounds, which is, of course, inefficient. (It would violate the broadcast-and-respond model to have the sensors respond individually to a single "who's alive" broadcast, since the responses would not be aggregated and, in any case, this would require an expected number of $O(n^{1.5})$ messages for a planar sensor network, would require sensors close to the base station do proportionally more work (hence, running down their batteries faster), and it would still have a delay of $O(n)$ communication rounds at the base station.) Thus, we are interested in this paper in efficient solutions to the dead sensor diagnosis problem that fit the broadcast-and-respond model.

1.4 Previous Related Work

For group testing, there is a tremendous amount of previous work on the standard (binary) version of the problem (e.g., see [1, 8, 9, 10, 15, 16, 18, 20, 21, 22]), but we are not familiar with any previous efforts directed at the (anonymous) generalized group testing problems we consider in this paper. The standard group testing problem has been applied to several other problems, including testing DNA clone libraries [11], testing blood samples for diseases, data forensics [2], and cryptography [7].

In terms of getting as close to the binary-result information-theoretic lower bound as possible, the best previous algorithms for the standard group testing problem are all adaptive (as are our algorithms). That is, tests are performed one at a time, with the processing of a single step usually requiring a parallel invocation across test elements, such that the results from previous tests allowed be used to guide future tests. When the exact number, d, of defective items is known, Hwang's generalized binary splitting algorithm [15] for the standard group testing problem exceeds the information theoretic lower bound by at most $d-1$. This algorithm is basically a set of d parallel binary searches, which start out together and eventually are split off. When d is not known to be an upper bound on the number of defective items, at most one additional test is required. Alleman [1] gives a split-and-overlap algorithm for the standard group testing problem that exceeds the information theoretic lower bound on the number of tests by less than $0.255d + \frac{1}{2}\lg d + 5.5$ for $d \leq n/2$. The 0.255 is replaced with 0.187 when $d \leq n/38$. When no constraint on the number, d, of defectives is known in advance, Schlaghoff and Triesch [22] give algorithms that require 1.5 times as many tests as the information theoretic lower bound for d defective out of n items.

Work on multiple access channels (MACs) dates back to before the invention of the Ethernet protocol, and there has been a fair amount of theoretical work on this problem as well (e.g., see [6, 12, 13, 14]). (Using our terminology, a MAC algorithm is equivalent to a ternary-result group testing algorithm with identifying results in the 1-result case.) There is a simple halfway-split binary tree algorithm that achieves an expected $2.885d$ number of steps (e.g., see [6]), which correspond to group tests in our terminology, to send d packets. This algorithm was improved by Hofri [14], using a biased splitting strategy (which we review below) to achieve an expected $2.623d$ steps. The best MAC algorithm we are familiar with is due to Greenberg and Ladner [12], who claim that their algorithm uses $2.32d$ expected number of steps, assuming d is known in advance. Interestingly, in the lower-bound paper of Greenberg and Winograd [13], the Greenberg-Ladner paper [12] is referenced as achieving $2.14d$ expected tests and, indeed, our analysis confirms this better bound for their algorithm, if d is known. Greenberg and Ladner [12] also present an algorithm for estimating d if it is not known in advance and, by our analysis, using this approximation algorithm with their MAC algorithm achieves $2.25d + O(\log d)$ expected number of steps (which is also better than the bound claimed in [12]).

Normally, such concern over small improvements in the constant factor for a leading term of a complexity bound would be of little interest. In this case, however, the reciprocal of the constant factor for this leading term corresponds to the throughput of the MAC; hence, even small improvements can yield dramatic improvements in achievable bandwidth. Admittedly, such improvements are not as applicable to the ways ethernets are used today, since the ethernet protocol is most commonly used for small subnets, with larger networks usually connected via routers and switches into a restricted topology network. Still, with the expanding deployment of wireless access points, there is a renewed motivation for MAC algorithms, particularly for environments where there are many wireless devices per access point. In this context, we are not familiar with any MAC algorithms that achieve our degree of efficiency without making additional probabilistic assumptions about the nature of packet traffic (e.g., see [6, 12, 13, 14]).

We believe the dead sensor diagnosis problem is new, but there is considerable previous work on device fault diagnosis for the case

in which devices can test each other and label the other device as "good" or "faulty," if the group controller can dictate the network's topology. For example, Yuan *et al.* [23] describe an aggregation protocol that assumes that sensors can detect when neighbors are faulty. Likewise, Beigel *et al.* [3, 4, 5] have designed a number of efficient diagnosis algorithms based on the paradigm of having devices test each other according to a schedule dictated by a group controller.

1.5 Our Results

In this paper, we present algorithms for generalized group testing when the result of each test may be non-binary. The worst-case performance of our algorithms beats the binary-result information-theoretic lower bound for standard group testing. (This is not an actual lower-bound violation, since ternary-result tests provide more information than do binary-result tests.) We also provide novel lower bounds for ternary-result group testing, which show that our algorithmic performance is within a small constant factor of the lower bound for ternary-result group testing.

Ternary-result group testing can be applied to multiple access channels. In this context, we provide new MAC conflict-resolution algorithms that achieve an expected $2.054d$ steps if d is known and $2.08d + O(\log d)$ tests if d is not known. Both of these bounds improve the previous constant factors for MAC algorithms and are based on the use of a new deferral technique that demonstrates the power of procrastination in the context of MAC algorithms. We also show that our MAC algorithm uses $O(d)$ steps with high probability, even if we reduce the randomness used, and we provide an improved algorithm for estimating the value of d if it is not known in advance.

Our group testing algorithms can be applied to dead sensor diagnosis, where the items are sensors and the defective items are the dead sensors. Our algorithms also are *concise*, which implies that they can be implemented as a parallel algorithm formulated as a constant-size broadcast query from the base station such that the aggregated response to such a query can provide the possible results needed for ternary-result and counting group testing. This immediately implies efficient parallel algorithms for the dead sensor diagnosis problem based on our ternary-result group testing algorithms. We also provide a novel counting-based group testing algorithm that uses an expected $1.89d$ tests to identify the d defective items. In addition, we give new deterministic ternary-result group-testing algorithms using $O(d \lg n)$ broadcast rounds (which would use a total of $O(dn \log n)$ messages for dead sensor diagnosis), with constant factors below the lower bound for binary-result group testing.

2. MOTIVATION AND DEFINITIONS

We have already discussed how collision resolution for a multiple access channel corresponds to ternary-result $(0/1/2+)$ group testing, with identifying tests in the 1-result case. In this section, we discuss further motivation for our other generalizations of group testing and we give some needed definitions as well.

2.1 Some Definitions for Group Testing

Recall that in the group testing problem we are given a set S of n items, d of which are defective. We are allowed to form an arbitrary subset, $T \subseteq S$, and perform a group test on T which, in the case of ternary-result group testing, has a ternary outcome. We say that T is *pure* if T contains no defective items, *tainted* if it contains exactly one defective item, and *impure* if it contains at least two defective items.

Furthermore, as mentioned above, in the case when T is tainted, we distinguish two possible variations in the way the test result is conveyed to us. We say that the result is *identifying* if it reveals the specific item, $x \in T$, that is defective. Otherwise, we say that the result is *anonymous* if it states that T is tainted but does not identify the specific item x in T that is defective.

Finally, we say that a testing scheme is *concise* if each test subset $T \subseteq S$ that might be formed by this scheme can be defined with an $O(1)$-sized expression E that allows us to determine, for any item $x \in S$, whether x is in T in $O(1)$ time using information only contained in E and x (that is, we allow for a limited amount of memory to be associated with x itself). For example, a test T might be defined by a simple regular expression, $101*10**011$, for the binary representation of the name of each x in T (we assume that item names are unique). The applications of MAC conflict resolution and dead sensor diagnosis both require that the corresponding testing scheme be concise, so the algorithms can be efficiently implemented in parallel. Incidentally, most MAC algorithms (e.g., see [6, 12, 13, 14]) also require that all devices have access to independent random bits, but we show that this requirement is not strictly necessary.

2.2 Group Testing for Packet Resolution in Multiple Access Channels

In this subsection, we present a simple reduction of probabilistic MAC conflict resolution solutions to generalized group testing.

In MAC algorithms, each device decides whether to attempt to send a message based on what has been observed and, if an attempt is to be made, the decision to send is made by flipping a biased coin with probability p.

Consider the case in which there are n devices in the system, and d devices each attempt to transmit with independent probability p. It is seen that, for large n, this scenario is approximated by using identifying ternary tests on a size-pn random subset of a set of n items, d of which are defective. In the MAC situation, the probability that exactly i devices will transmit is

$$P_{MAC}(i) = \binom{d}{i} p^i (1-p)^{d-i}.$$

A conflict arises when two or more devices transmit.

In the testing situation, the probability that exactly i of the d defective items are within the randomly selected subset of size pn is

$$P_{test}(i) = \binom{d}{i} \prod_{j=0}^{i-1} \frac{pn-j}{n-j} \prod_{j=i}^{d-1} \frac{n-pn-j+i}{n-j}.$$

The subset is impure when two or more defective items are in that subset.

2.3 Group Testing for Dead Sensor Diagnosis

In this subsection, we present some simple reductions of the dead sensor diagnosis problem to generalized group testing. All of our reductions fit the broadcast-and-respond paradigm of sensor communication, where the base station issues a broadcast and receives back an aggregated response, which is the result of an associative function applied to the sensor responses, and which is computed by the sensors routing the combined response back to the base station. Because of the assumed simple nature of sensors, we desire aggregate responses based on the use of simple functions.

Given a concise ternary-result group testing algorithm, A, we can use A to perform dead sensor diagnosis by simulating each step of A with a broadcast and response. Because A is concise, each test in A can be defined by a constant-sized expression E that is then broadcast to each live sensor. Moreover, each live sensor x

can determine in $O(1)$ time whether it belongs to the set T defined by E and can participate in an aggregate response back to the base station. Thus, the remaining detail is to define possible aggregate responses that support useful responses, with either identifying or anonymous results in the tainted cases:

- *Count*. The first function we consider is a simple count of the live sensors in T. Each live sensor x can determine if it belongs to T in $O(1)$ time, since we are restricting our attention to concise testing algorithms. Likewise, each sensor y routing an answer back to the base station need only sum the counts it receives from downstream routers (plus 1 if y is in T). This aggregation function supports ternary responses, since the base station knows $|T|$ and can compare this value with the count performed by the live sensors. The count function is associative, and easily fits in the broadcast-and-respond model, but it does not allow for identifying the dead sensor in the tainted case.

- *Large-ID summation*. Suppose that the n sensors are assigned ID numbers that are guaranteed to all be greater than $2n$ such that no ID number can be formed as the sum of two or more other ID numbers. Then a summation of the ID numbers of the live sensors in T can be used to perform a ternary-result test, which will be an identifying test in the case of a result indicating that T is tainted. Specifically, the difference between $\sum_{x\in T} x$ and the returned value will either be 0, the ID of a single sensor, or a value that is the sum of two or more sensor IDs. Of course, this function requires that sensors can add integers as large as $\sum_{x\in S} x$.

Thus, we can use dead sensor diagnosis to motivate identifying ternary-result $(0/1/2+)$ group testing as well as anonymous counting group testing. Of course, if we combine these two functions to operated on paired responses, we can implement an identifying counting group testing algorithm. These aggregation functions are not meant to be exhaustive.

3. THE BINARY TREE ALGORITHM FOR TERNARY-RESULT GROUP TESTING

Since it provides a starting point for our more sophisticated algorithms, we review in this section the binary tree algorithm for ternary-result group testing with identifying results for tainted tests, which was originally presented in the context of MACs [6]. That is, we consider the problem of identifying the defective items in a set of items when we can adaptively test arbitrary subsets and each test result indicates whether the number of defective items contained in the tested subset is zero, one, or at least two. We also provide a simplified analysis of its expected performance.

The main idea of the binary tree algorithm (parameterized by p) is to partition a set that is known to be impure into two unequal-sized subsets, of sizes p and $q = 1-p$ of the set's size, and to recursively test each of these subsets in an obvious binary tree fashion. However, the algorithm takes advantage of one simple optimization—if the first subset in a recursive call turns out to be pure (that is, the result is 0 defectives), we can avoid the top level testing of the second subset and go immediately to splitting it in two and testing the two parts.

The original algorithm used $p = 0.5$ and it has been shown [14] that $p \approx .4175$ minimizes the expected number of tests. We make use of the smaller root of the equation $p_2 = (1-p_2)^2$, which is solved by $p_2 = \frac{3-\sqrt{5}}{2} \approx 0.38197$, and of $q_2 = (1-p_2) \approx 0.61803$.

The binary tree algorithm begins by testing the set, S, of items. If the test indicates that S is pure or tainted, in which case the one defective item will have been identified, then the algorithm is done. Otherwise, initialize the set L of identified defective items to empty and proceed with subroutine *Identify(S)* as follows:

1. Partition S into two subsets, A and B, where $|A| = p|S|$.
2. Test subset A.
   (a) If A is impure, then recursively invoke *Identify*(A).
   (b) If A is tainted with item z, then add z to list L.
3. If A is pure then we know that subset B is impure, and so there is no need to test B. Otherwise, test subset B.
   (a) If B is impure, then recursively invoke *Identify*(B).
   (b) If B is tainted with item z, then add z to list L.

When partitioning S into A and B, we can select A as consisting of those items whose ID values are ranked contiguously, 1 through $p|S|$. The items in A, or B, can be specified by giving lower and upper limits on ID values. Thus, the binary tree algorithm is concise.

THEOREM 1. *$w_2 d \lg n + o(\lg n)$ ternary tests under the identifying model suffice, in the worst case, to identify all defectives in a set containing n items of which d are defective, where $w_2 = -(1/\lg p_2) \approx 0.720210$.*

PROOF. We analyze the performance of the binary tree algorithm with $p = p_2$. Let $X_d(n)$ be the worst case number of tests required by algorithm *Identify*(S) when S is a set of n items of which d turn out to be defective.

For $d = 2$ and $d = 3$, we have the following recurrence. (Note that sets with 0 or 1 defective items require no further testing, thus $X_d(1) = 1$, and that it is assumed that $X_3(n) \geq X_2(n)$.)

$$X_d(n) = \max \left\{ \begin{array}{l} 2 + X_d(p_2 n) \\ 1 + X_d(q_2 n) \end{array} \right. \tag{1}$$

If the first term of the recurrence were to be the maximum term, then $X_d(n) = 2 + X_d(p_2 n) = -(2/\lg p_2)\lg n$. If the second term of the recurrence were to be the maximum term, then $X_d(n) = 1 + X_d(q_2 n) = -(1/\lg q_2)\lg n$.

We see that $X_2(n) = X_3(n) = -(1/\lg q_2)\lg n = 2 w_2 d \lg n \approx 1.4404 \lg n$.

For $d \geq 4$, we have the following recurrence. (It is assumed that $X_d(n) \geq X_{d-1}(n)$.)

$$X_d(n) = \max \left\{ \begin{array}{ll} 2 + X_i(p_2 n) + X_{d-i}(q_2 n), & \text{for } 1 \leq i \leq d-2 \\ 2 + X_d(p_2 n) & \\ 1 + X_d(q_2 n) & \end{array} \right. \tag{2}$$

Consider $X_d(n) = x \lg n + o(\lg n)$, and we shall solve for x.

Assume that, for even $1 < i < d$, $X_i(n) = w_2 i \lg n + o(\lg n)$, and that, for odd $1 < i < d$, $X_i(n) = w_2 (i-1) \lg n + o(\lg n)$.

Consider any $d \geq 4$. If the first term of the recurrence were to be the maximum term, then $x \geq (d-1) w_2 > 2.16$, since $d \geq 4$. If the second term were to be the maximum term, then $x = -2/\lg p_2 \approx 1.44$. If the third term were to be the maximum term, then $x = -1/\lg q_2 \approx 1.44$.

Thus, the first term is the maximum term and

$$X_d(n) \approx d X_2(n)/2 = w_2 d \lg n, \text{ for even } d,$$

$$X_d(n) \approx (d-1) X_2(n)/2 = w_2 (d-1) \lg n, \text{ for odd } d.$$

□

Thus, the binary tree algorithm has good worst-case performance. It also has good average-case performance, as the following theorem shows[1].

THEOREM 2. *On average, when $p = p_2$, Identify requires fewer than $2.631d$ ternary tests to identify all defectives in a set containing n items of which d are defective, for $n >> d$. Thus, the binary tree algorithm requires fewer than $1 + 2.631d$ ternary tests.*

PROOF. Provided in the full version. □

Using different values of p yields different results. To minimize E_2, a value of $p = \sqrt{2} - 1 \approx 0.4142$ is best [14], requiring 3.414 tests. To minimize E_3, $p \approx 0.4226$ is best and requires 5.884 tests. To minimize E_4, $p \approx 0.4197$ is best and requires 8.482 tests. $p = p^* \approx 0.41750778$ is asymptotically optimal for large d. The curves are fairly flat, so, although one could tune p depending on the expected distribution of the values of d, choosing $p = p^*$ is a good choice for most distributions and, as noted by Hofri [14], is optimal for the naturally arising distribution, when the defective items are i.i.d., requiring $\approx 2.6229d$ tests.

4. THE DEFERRAL ALGORITHM

In this section, we describe how to substantially improve on the average case of the binary tree algorithm under the assumption that we have a good approximation on the number, d, of defective items. This algorithm is especially useful for the Multiple Access Channel problem.

The main idea of our algorithm, which we call *Deferral*, begins by using an approach used by Greenberg and Ladner [12] where we use knowledge of the approximate number of defective items to randomly partition the set of items into a set, L, of buckets such that the expected number of defective items in each bucket is a constant. This process is called a *spreading* action and, for $|L| = sd$, the parameter s is called the *spread factor*. Greenberg and Ladner's algorithm performs a spreading action using an appropriate spread factor (they recommend $s = 0.8$), performs a test on each bucket, and then applies the binary tree algorithm to each bucket that has a 2+ result.

Our approach does something similar, but augments it with a new deferral technique that may at first seem counter-intuitive. We also perform a spreading action, perform a test for each bucket, and apply the binary tree algorithm recursively to any bucket with a 2+ result, except that we cut recursive calls short in certain cases and defer to the future all items whose status remains unclear from all such calls. We then recursively apply the entire algorithm on these deferred items. As we show in our analysis, this is a case when procrastination provides asymptotic improvements, for this deferral algorithm has a better average-case performance than does the direct do-it-now approach of Greenberg and Ladner.

Deferral proceeds as follows.

1. Initialize a deferral bucket to empty.
2. For each bucket K in set L, identify some of the defective items in K (and defer others) as follows.

   Test K. If the test shows that K is pure or tainted, all defective items of K will have been identified. Otherwise, use algorithm *BucketSearch* on bucket K.
3. Finally, if the deferral bucket is non-empty then recursively apply *Deferral* to the set of items in the deferral bucket.

Algorithm *BucketSearch* proceeds as follows.

1. Partition K into a first portion A having fraction p of the items in K, and a second portion B having the remainder fraction $q = 1 - p$ of K's items.
2. Test A. One of three results will occur:
   (a) If A is pure, then recursively invoke *BucketSearch*(B).
   (b) If A is tainted, then the lone defective item in A will have been identified. In this case, test B and, only when B is impure, recursively invoke *BucketSearch*(B).
   (c) If A is impure, recursively invoke *BucketSearch*(A). Finally, merge B with the deferral bucket.

It might not be immediately obvious, but this algorithm can be made concise, using $O(1)$ words of memory per test element (one of which, for example, can keep the state of whether an item is being deferred or not).

4.1 Analysis of the Deferral Algorithm

Let $P_s(k)$ be the probability of a bucket containing exactly k defective items, given that we are using $|L| = sd$ buckets, *i.e.*, we have a spread factor of s. Then

$$P_s(k) = \binom{d}{k}\left(\frac{1}{sd}\right)^k\left(1-\frac{1}{sd}\right)^{d-k} \approx \frac{1}{k!s^k e^{1/s}}$$

For example, if we use a spread factor of $s = .75$, then $P_s(0) < 0.2636$, $P_s(1) < 0.3515$, $P_s(2) < 0.2344$, ..., $P_s(6) < 0.0021$, and $P_s(7) < 0.0004$. We observe that we expect that 99.9% of all buckets contain fewer than seven defective items in this case (and this is true for all spread factors greater than 0.5). Furthermore, $P_s(i)$ is monotonic decreasing for $i > 2$. Therefore, in analyzing the expected behavior of algorithms that use a spreading step with a reasonable spread factor, the expected number of tests is dominated by the expected number of tests performed on buckets with fewer than seven defective items.

4.1.1 Analyzing the Expected Number of Tests per Bucket

We begin by estimating the expected number, E_d, of tests performed in a bucket containing d defectives (not counting the global test for the bucket or future deferred tests for items currently in the bucket). Certainly, $E_d \le E'_d$, where E'_d is the expected number of tests in the standard binary tree algorithm, since every test performed by the deferral binary tree algorithm would also be made by the standard algorithm. Moreover, for large values of d, this estimate will be sufficient for our purposes.

Therefore, we concentrate on bounding E_d for small values of d. By construction, $E_0 = E_1 = 0$. For $d > 1$, we consider the cases x-y that arise when partitioning a set containing d defective items into two subsets that turn out to contain, respectively, x and y defective items. If $d = 2$, then the 2-0 case entails 1 test and a recursive call (and a deferral of a pure set), the 1-1 case entails 2 tests, and the 0-2 case entails 1 test and a recursive call. Thus, letting $q = 1 - p$,

$$\begin{aligned} E_2 &= p^2(E_2+1) + 2pq(2) + q^2(E_2+1) \\ &= p^2E_2 + (p^2+2pq+q^2) + 2pq + q^2E_2 \\ &\le \frac{1+2pq}{1-p^2-q^2} = \frac{1+2pq}{2pq}. \end{aligned}$$

[1]This theorem simplifies a result of [14] and it implies a randomized algorithm with the same performance if we preface the binary tree algorithm with an initial random permutation of the items.

Likewise, if $d = 3$, the 3-0 case entails 1 test and a recursive call (and a deferral of a pure set), the 2-1 case entails 1 test and a recursive call on a 2-defective set (and a deferral of a 1-defective set), the 1-2 case entails 2 tests and a recursive call on a 2-defective set, and the 0-3 case entails 1 test and a recursive call. Thus,

$$\begin{aligned} E_3 &= p^3(E_3+1)+3p^2q(E_2+1)+3pq^2(E_2+2)+q^3(E_3+1) \\ &\leq \frac{1+3pq^2+(3p^2q+3pq^2)E_2}{1-p^3-q^3}. \end{aligned}$$

Similarly,

$$E_4 \leq \frac{1+4pq^3+(4p^3q+4pq^3)E_3+6p^2q^2E_2}{1-p^4-q^4}.$$

Likewise,

$$E_5 \leq \frac{1+5pq^4+(5p^4q+5pq^4)E_4+10p^3q^2E_3+10p^2q^3E_2}{1-p^5-q^5}.$$

Moreover,

$$E_6 \leq \frac{1+6pq^5+(6p^5q+6pq^5)E_5+15p^4q^2E_4+20p^3q^3E_3+15p^2q^4E_2}{1-p^6-q^6}.$$

Finally (which will be sufficient for our analysis),

$$E_7 \leq \frac{\begin{array}{c}1+7pq^6+(7p^6q+7pq^6)E_6+21p^5q^2E_5+35p^4q^3E_4 \\ +35p^3q^4E_3+21p^2q^5E_2\end{array}}{1-p^7-q^7}.$$

But this is only for the first round. We still need to account for the expected number of defective items deferred from this round to future rounds.

4.1.2 Analyzing the Expected Number of Deferred Defective Items

Let D_d denote the expected number of defective items deferred in a bucket with d defective items. Certainly, since we are guaranteed to find at least 2 defective items for any bucket with $d \geq 2$, we can bound $D_d \leq d-2$ for $d \geq 2$. Moreover, we trivially have that $D_0 = D_1 = 0$. We derive a more accurate bound on D_d for some small values of d, beginning with D_3.

When $d = 3$, the 3-0, 1-2, and 0-3 cases all entail recursive calls, but only the 2-1 case causes a defective item to be deferred. Thus,

$$D_3 = p^3D_3+3p^2q+q^3D_3 \leq \frac{3p^2q}{1-p^3-q^3} = p.$$

For $d = 4$, the 4-0 and 0-4 cases both entail recursive calls, the 3-1 case entails a 3-defective recursive call and 1 deferral, the 2-2 case entails 2 deferrals, and the 1-3 case entails a 3-defective recursive call. Thus,

$$\begin{aligned} D_4 &= p^4D_4+4p^3q+(4p^3q+4pq^3)D_3+12p^2q^2+q^4D_4 \\ &\leq \frac{4p^3q+12p^2q^2+(4p^3q+4pq^3)D_3}{1-p^4-q^4}. \end{aligned}$$

Likewise,

$$D_5 \leq \frac{5p^4q+20p^3q^2+30p^2q^3+(5p^4q+5pq^4)D_4+10p^3q^2D_3}{1-p^5-q^5}.$$

Finally (which will be sufficient for our analysis),

$$D_6 \leq \frac{\begin{array}{c}6p^5q+30p^4q^2+60p^3q^3+60p^2q^3+(6p^5q+6pq^5)D_5 \\ +15p^4q^2D_4+20p^3q^3D_3\end{array}}{1-p^6-q^6}.$$

It does not result in elegant equations, but we can nevertheless combine this analysis with the previous bounds on E_d and $P_s(k)$ to derive the expected number of tests performed by our algorithm. For example, with a spread factor of $s = 0.8$ and a split parameter of $p = 0.479$, we obtain that the expected number of tests is less than $2.054d$.

4.2 Estimating the Number of Defectives

Greenberg and Ladner [12] give a simple repeated doubling algorithm for estimating the number of defectives, d, in a set. Their algorithm repeatedly selects a random set of size $n/2^i$, for $i = 1,2,\ldots$, until a test results in a non-collision (that is, a 0- or 1-result), and then it sets its estimate of the number of defectives as $\hat{d} = 2^i$. Unfortunately, this simple approximation is not sufficiently accurate for our purposes, so we provide in this section a simple improvement of the doubling algorithm, which increases the accuracy of the estimate while only increasing the number of tests by a small additive factor.

We begin by applying the simple doubling algorithm. This algorithm is 99.9% likely to use $O(\log d)$ tests and produce an estimate, $\hat{d}$, such that $d/32 \leq \hat{d} \leq 32d$. However, the estimate is within a factor of 2 of d only about 75% of the time. (It varies, approximately 65% to 90%, depending on how close d is to a power of 2.) While this is insufficient to produce a useful estimate of d for the sake of computing a spread factor, it is sufficient as a first step for coming up with a better approximation for d.

Let us, therefore, assume we have computed the estimate $\hat{d}$. We next perform a sequence of experiments, for $i = j, j+1,\ldots$, where experiment i involves choosing a constant number, c, of random subsets of size $n/2^{i/a}$ and performing a test for each one, where $j = \max\{1, a(\log\hat{d}-5)\}$ with a a small integer such as 2 or 4. We stop the sequence of experiments as soon as one of the c tests returns a non-collision result (i.e., a result of 0 or 1). We then use the value of i to produce a refined estimate, $\hat{d}'$, for d. We use $\hat{d}' = f(a,c)\cdot 2^{i/a}$, where f is a normalizing function so that $\mathrm{E}[\hat{d}'] = d$.

The probability that all c subsets for experiment i contain collisions quickly approaches $1-\left(1-(t+1)e^{-t}\right)^c$, where $t = d/2^{i/a}$. This fact can then be used to find a good estimate of d, based on the values of a and c. For example, when $a = 2$ and $c = 16$, then the best estimate of $\hat{d}'$ is $2^{i/2+2}$. When $a = 4$ and $c = 8$, using $f(a,c) = 4.3$ results in the estimate being within a factor of 2 of d about 99.3% of the time (varying about 98% to 100%, depending on d). Moreover, combining this estimate algorithm with our deferred binary tree algorithm results in a testing algorithm that uses an expected $2.08d + O(\log d)$ tests, and which does not need to know the value of d in advance.

4.3 Reducing the Randomness of the Deferral Algorithm

In this subsection, we show how to reduce the randomness needed for the deferral algorithm, while keeping it concise. In particular, we do not need $O(\log n)$ random bits associated with each defective item; we can use an expected $O(\log n)$ random bits associated with a group controller instead. Moreover, even with this reduced randomness, we show that we will make only $O(d)$ tests, with high probability, $1-O(1/d)$.

The main idea of our modified algorithm is to apply the *Deferral* algorithm, as described above, but use a random hash function to define the top-level partitioning to be performed. Indeed, the top-level distribution of our algorithm is closely related to the hashing of d out of n elements into a table of size $O(d)$, in that mapping items to cells without collisions is quite helpful (corresponding to identifying tests in our case). The main difference between our

```
Algorithm AN ( S )
        // Given: set S of items
        // Return: identity of all defective items
    if test(S) ≤ 1 then identify the defective via binary search and exit
    list L ← ∅
    Reduce(S)
    if list L has only 2 sets, A and B then Final2(A,B)
    else Final3(L)

Subroutine Reduce ( S )
        // Given: set S of items that includes at least 2 defective items
        // Return: list L of disjoint subsets of S that each contain one defective item
    p2 ← 0.38196601
    Partition S into two subsets, A and B, where |A| = p2|S|
    t1 ← test(A)
    if t1 ≥ 2  then     Reduce(A)
    if t1 = 1  then     add A to list L
    if t1 = 0  then     t2 ← 2
               else     t2 ← test(B)
    if t2 ≥ 2  then     Reduce(B)
    if t2 = 1  then     add B to list L

Subroutine Final2 ( A,B )
        // Given: two disjoint tainted sets
        // Return: identity of the 2 defective items
    p3 ← 0.3176722              // q3 = (1 − p3)
    while |A| > 1 and |B| > 1
                                // Start with sets (A,B) having sizes (x,y)
        Partition A into A1 and A2, where |A1| = p3|A|
        Partition B into B1 and B2, where |B1| = p3|B|
        t1 ← test(A1 ∪ B1)
        if t1 = 0 then      ⟨A,B⟩ ← ⟨A2,B2⟩  // R0: sizes (q3x,q3y), 1 test
        else if t1 = 1 then
            t2 ← test(A1)
            if t2 = 0 then  ⟨A,B⟩ ← ⟨A2,B1⟩  // R1: sizes (q3x,p3y), 2 tests
            else            ⟨A,B⟩ ← ⟨A1,B2⟩  // R1: sizes (p3x,q3y), 2 tests
        else /* (t1 = 2) */ ⟨A,B⟩ ← ⟨A1,B1⟩  // R2: sizes (p3x,p3y), 1 test
    use binary search to identify defectives in the (at most 1) set of A and B whose size > 1
```

Figure 1: Analysis algorithm using anonymous ternary tests

problem and the hashing problem is that, in the case of a collision (corresponding to an impure test set in our case), we do not know which items or even how many items have collided. We provide the details of this algorithm and its analysis in the full version, proving the following:

THEOREM 3. *Given a set of n items with d defectives, the number of tests performed by the reduced-randomness ternary-result group testing algorithm is $O(d)$ with probability $1-O(1/d)$.*

5. OUR ANONYMOUS ALGORITHM

In this section, we discuss an efficient concise deterministic ternary-result group testing algorithm for the case in which a test of a tainted set does not identify the defective item.

Consider algorithm *AN*(*S*), shown in Figures 1 and 2.

Subroutine *Reduce* reduces the original problem to one of identifying the d defective items in a collection L of d tainted subsets. Note that *Reduce* is essentially our earlier *Identify* algorithm, in which testing a tainted set immediately identified the defective item. Here, we require additional testing to identify the defective item. When $d = 2$, subroutine *Final2* iterates reducing the size of the two sets in L until they are singletons. When $d \geq 3$, subroutine *Final3* iterates reducing the size of three of the sets in L until at most two of the d sets are non-singleton, and then utilizes either *Final2* or binary search to reduce the remaining set(s) to become singleton(s).

All subsets can be selected so that the items of each subset have ID value ranks that are contiguous. All tests involve the union of at most three subsets, each of which can be specified as consisting of items whose ID values are in a specified range. Thus, algorithm *AN* is concise.

5.1 Analysis of Algorithm *AN*

Let $W_d(n)$, for $d > 1$, be the worst-case numbers of tests made by *AN*(*S*) when $|S| = n$ and there turns out to be d defectives. We provide the analysis and prove the correctness of the *AN* algorithm in the full version, proving the following theorem:

THEOREM 4. $W_2(n) \leq 1.8756 \lg n + o(\lg n)$
and, for $d \geq 3$,

$$W_d(n) \leq (0.3307 + 0.7202d) \lg n + o(\lg n).$$

```
Subroutine Final3 ( L )
        // Given: list L of d ≥ 3 disjoint tainted sets
        // Return: identity of the d defective items
    p4 ← 0.27550804                 // q4 = (1 − p4)
    while ∃ at least three non-singleton sets in L
        ⟨a,b,c⟩ ← indices of the largest three non-singleton sets in L
                                    // Start with sets (La,Lb,Lc) having sizes (x,y,z)
        Partition La into A1 and A2, where |A1| = p4|La|
        Partition Lb into B1 and B2, where |B1| = p4|Lb|
        Partition Lc into C1 and C2, where |C1| = p4|Lc|
        t1 ← test(A1 ∪ B1 ∪ C1)
        if t1 = 0 then              ⟨La,Lb,Lc⟩ ← ⟨A2,B2,C2⟩  // R0: sizes (q4x,q4y,q4z), 1 test
        else if t1 = 1 then
            t2 ← test(A1 ∪ B2)
            if t2 = 0 then          ⟨La,Lb,Lc⟩ ← ⟨A2,B1,C2⟩  // R1: sizes (q4x,p4y,q4z), 2 tests
            else if t2 = 1 then     ⟨La,Lb,Lc⟩ ← ⟨A2,B2,C1⟩  // R1: sizes (q4x,q4y,p4z), 2 tests
            else /* (t2 = 2) */     ⟨La,Lb,Lc⟩ ← ⟨A1,B2,C2⟩  // R1: sizes (p4x,q4y,q4z), 2 tests
        else // (t1 = 2)
            t2 ← test(A1 ∪ B2)
            if t2 = 0 then          ⟨La,Lb,Lc⟩ ← ⟨A2,B1,C1⟩  // R2: sizes (q4x,p4y,p4z), 2 tests
            else if t2 = 1 then
                t3 ← test(C1)
                if t3 = 0 then      ⟨La,Lb,Lc⟩ ← ⟨A1,B1,C2⟩  // R2: sizes (p4x,p4y,q4z), 3 tests
                else                ⟨La,Lb,Lc⟩ ← ⟨A1,B1,C1⟩  // R3: sizes (p4x,p4y,p4z), 3 tests
            else /* (t2 = 2) */     ⟨La,Lb,Lc⟩ ← ⟨A1,B2,C1⟩  // R2: sizes (p4x,q4y,p4z), 2 tests
    if ∃ two non-singleton sets (A and B) in L then Final2(A,B)
    else if ∃ one non-singleton set, A, in L then identify A's defective by using binary search
```

Figure 2: Final subroutine when $d \geq 3$

6. LOWER BOUNDS

Let $T_d(n)$ be the worst-case minimum number of tests required by any algorithm that uses ternary tests under the anonymous model to identify all defective items in a set containing n items of which d are defective. The following is an immediate corollary of Theorem 4.

COROLLARY 1.

$$T_2(n) \leq 1.8756 \lg n + o(\lg n)$$

and, for $d \geq 3$*,*

$$T_d(n) \leq (0.3307 + 0.7202d) \lg n + o(\lg n).$$

We prove the following theorem in the full version.

THEOREM 5.

$$T_2(n) \geq 1.8133 \lg n,$$

$$T_3(n) \geq 2.1507 \lg n,$$

and, for $d \geq 4$*,*

$$T_d(n) \geq 0.6309 d \lg n.$$

7. USING COUNTING QUERIES

In this section, we discuss a variant of our testing algorithm for the case when the queries provide an exact count of the number of defectives in a test set, and the result in the case of a 1-result identifies the defective item in the test set. As we show, the expected performance of this algorithm is significantly better than that of the ternary-result group testing algorithm.

We apply an initial spreading action to distribute items across a set of buckets and we then perform a test for each bucket. The main difference is in the binary tree algorithm we then apply to each bucket B whose test indicates it has $t \geq 2$ defective items:

1. We set a partition factor, p, according to the analysis, and we split B into subsets B_1 and B_2 so that B_1 has $p|B|$ items from B and B_2 has the remaining items.

2. We perform a test for B_1 and, if the number, t_1, of defective items in B_1 is at least two, then we recursively search in B_1.

3. If the (possibly recursive) testing of B_1 has revealed all t defective items from B, then we skip the testing of B_2, for it contains no defective items in this case.

4. Otherwise, if the test for B_1 revealed $t_1 = t - 1$ defectives, then we immediately test B_2 to identify its one defective item.

5. If, on the other hand, the test for B_1 revealed t_1 defectives, with

   $$0 \leq t_1 < t - 1,$$

   then we recursively search in B_2 (without performing a global test for B_2, since we know it must have at least 2 defectives).

Note that no deferral is needed in this algorithm.

7.1 Analysis of the Counting Algorithm

In the full version, we provide a set of bounds, similar to those given above for the ternary-result algorithm, on the expected number of tests performed for small-sized buckets. These bounds can then be combined with an analysis (as given above) for bounding the number of buckets of various sizes to derive an expected bound on the number of tests performed by our algorithm. For example, if we choose a spread factor of

$$s = 0.58$$

and a split parameter

$$p = .4715,$$

then we find that

$$E_d \leq 1.896d,$$

which is significantly better than that obtained by the ternary-result group testing algorithm.

Acknowledgment

We would like to thank David Eppstein for helpful discussions regarding the topics of this paper.

8. REFERENCES

[1] A. Allemann. *Improved Upper Bounds for Several Variants of Group Testing*. PhD thesis, Rheinisch-Westfalischen Technischen Hochschule Aachen, November 2003.

[2] M. J. Atallah, M. T. Goodrich, and R. Tamassia. Indexing information for data forensics. In *3rd Applied Cryptography and Network Security Conference (ACNS)*, volume 3531 of *Lecture Notes in Computer Science*, pages 206–221. Springer, 2005.

[3] R. Beigel, S. R. Kosaraju, and G. Sullivan. Locating faults in a constant number of parallel testing rounds. In *Proc. of the ACM Symposium on Parallel Algorithms and Architectures (SPAA)*, pages 189–198, 1989.

[4] R. Beigel, G. Margulis, and D. Spielman. Fault diagnosis in 32 parallel testing rounds. In *Proc. of the ACM Symposium on Parallel Algorithms and Architectures (SPAA)*, pages 21–29, 1993.

[5] R. Beigel, G. Margulis, and D. Spielman. Fault diagnosis in a flash. In *Proc. of the 36th IEEE Symposium on Foundations of Computer Science*, pages 571–580, 1995.

[6] J. I. Capetanakis. Tree algorithms for packet broadcast channels. *IEEE Trans. Inf. Theory*, IT-25(5):505–515, 1979.

[7] Colbourn, Dinitz, and Stinson. Applications of combinatorial designs to communications, cryptography, and networking. In *Surveys in Combinatorics, 1993, Walker (Ed.), London Mathematical Society Lecture Note Series 187*. Cambridge University Press, 1999.

[8] A. DeBonis, L. Gasieniec, and U. Vaccaro. Generalized framework for selectors with applications in optimal group testing. In *Proceedings of 30th International Colloquium on Automata, Languages and Programming (ICALP'03)*, pages 81–96. Springer, 2003.

[9] D.-Z. Du and F. K. Hwang. *Combinatorial Group Testing and Its Applications, 2nd ed.* World Scientific, 2000.

[10] D. Eppstein, M. T. Goodrich, and D. S. Hirschberg. Improved combinatorial group testing for real-world problem sizes. In *Workshop on Algorithms and Data Structures (WADS)*, Lecture Notes Comput. Sci. Springer, 2005.

[11] M. Farach, S. Kannan, E. Knill, and S. Muthukrishnan. Group testing problems with sequences in experimental molecular biology. In *SEQUENCES*, page 357. IEEE Press, 1997.

[12] A. G. Greenberg and R. E. Ladner. Estimating the multiplicities of conflicts in multiple access channels. In *Proc. 24th Annual Symp. on Foundations of Computer Science (FOCS'83)*, pages 383–392. IEEE Computer Society, 1983.

[13] A. G. Greenberg and S. Winograd. A lower bound on the time needed in the worst case to resolve conflicts deterministically in multiple access channels. *J.ACM*, 32(3):589–596, 1985.

[14] M. Hofri. Stack algorithms for collision-detecting channels and their analysis: A limited survey. In A. V. Balakrishnan and M. Thoma, editors, *Proc. Inf. Sem. Modelling and Performance Evaluation Methodology*, volume 60 of *Lecture Notes in Control and Info. Sci.*, pages 71–85, 1984.

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

[16] F. K. Hwang and V. T. Sós. Non-adaptive hypergeometric group testing. *Studia Scient. Math. Hungarica*, 22:257–263, 1987.

[17] C. Intanagonwiwat, D. Estrin, R. Govindan, and J. Heidemann. Impact of network density on data aggregation in wireless sensor networks. In *Proc. of Int. Conf. on Distributed Computing Systems*, 2002.

[18] W. H. Kautz and R. C. Singleton. Nonrandom binary superimposed codes. *IEEE Trans. Inf. Th.*, 10:363–377, 1964.

[19] N. A. Lynch. *Distributed Algorithms*. Morgan Kaufmann, San Francisco, 1996.

[20] A. J. Macula and G. R. Reuter. Simplified searching for two defects. *J. Stat. Plan. Inf.*, 66:77–82, 1998.

[21] M. Ruszinkó. On the upper bound of the size of the r-cover-free families. *J. Combin. Th. Ser. A*, 66:302–310, 1994.

[22] J. Schlaghoff and E. Triesch. Improved results for competitive group testing. Report, Forschungsinstitut fur Diskrete Mathematik, Institut fur Okonometrie und Operations Research, Rheinische Friedrich-Wilhelms-Universitat Bonn, 1997. Report 97858.

[23] W. Yuan, S. V. Krishnamurthy, and S. K. Tripathi. Improving the reliability of event reports in wireless sensor networks. In *Proc. of IEEE Int. Symp. on Computers and Communication (ISCC)*, pages 220–225, 2004.