A New Competitive Algorithm for Group Testing

Amotz Bar-Noy * Frank K. Hwang † Ilan Kessler * Shay Kutten *

Abstract

Algorithms for the group testing problem when there is no a priori information on the number of defective items are considered. The efficiency criterion used is the competitive ratio, which is the ratio of the number of tests required by an algorithm when there is no a priori information on the number of defective items, to the number of tests required by an optimal algorithm when the number of defective items is known in advance. A new algorithm is presented, and it is shown that the competitive ratio of this algorithm is 2. This result is an improvement over a previous algorithm due to Du and Hwang [DH2] the competitive ratio of which is 2.75. It also proves a conjecture made in [DH2]. A new application of group testing techniques for high speed network is discussed.

1 Introduction

The *group testing* problem may be described as follows. Consider a set of n items each of which being either *defective* or *good*. The objective is to identify all the defective items by a minimal number of *tests*. Each test is performed on a subset of the n items and can have two possible outcomes – either *positive* or *negative*. A positive outcome indicates that the subset under test is *contaminated*, i.e. it contains at least one defective item. A negative outcome indicates that the subset under test is *pure*, i.e. it contains only good items.

Group testing was first introduced by Dorfman [Do] in 1943. In subsequent years a large number of papers were published on this subject. There are two basic models in the literature for this problem – a stochastic model and a deterministic model. In the stochastic model it is assumed that each item is defective with some probability, and the objective is to minimize the *expected* number of tests required to identify all the defective items. In the deterministic model it is assumed that the number of defective items (or an upper bound on this number) is known in advance, and the objective is to minimize the *maximum* number of tests required to identify all the defective items.

In many practical applications, the assumption of the deterministic model that there is an a priori information on the number of defective items is not realistic. Recently, Du and Hwang [DH2] presented algorithms that do not require an a priori information on the number of defective items. The efficiency criterion used in [DH2] is the *competitive ratio*. This criterion, which originated from studies of on-line algorithms (see [MMS]), may be described as follows. Let $M(n,d)$ be the number of tests required by an optimal group testing algorithm to identify among n items all d defective items, when d is known in advance. Let $T_{\mathcal{A}}(n,d)$ be the number of tests required by algorithm $\mathcal{A}$ to identify among n items all d defective items, when d is not known in advance. Then $\mathcal{A}$ is α-competitive if for all $0 \leq d \leq n$, $T(n,d) \leq \alpha M(n,d) + \beta$ for some constant β. The parameter α is called the com-

*IBM T. J. Watson Research Center, P. O. Box 704, Yorktown Heights, NY 10598.

†AT&T Bell Laboratories, Murray Hill, New Jersey 07974.

CH3133-6/92/0000-0786 $3.00 © 1992 IEEE

petitive ratio. Du and Hwang [DH2] presented an algorithm that achieves a competitive ratio of 2.75, and conjectured that a 2-competitive algorithm exists.

In this paper we present a group testing algorithm that achieves a competitive ratio of 2, thus improving the result obtained in [DH2] and also proving the above conjecture.

The organization of the paper is as follows. In Section 2 we present our algorithm. In Section 3 we analyze the performance of this algorithm, and in Section 4 we show that its competitive ratio is 2. Finally, in Section 5 we discuss a new application for group testing techniques in high-speed networks.

2 The algorithm

The basic idea behind the algorithm is as follows. Since the number of defective items d is unknown, the algorithm tries to estimate the value of d. If d is large the algorithm would like to find small contaminated sets whereas if d is small the algorithm would like to find large pure sets.

The algorithm uses the following strategy. It tests disjoint sets of items of sizes $1, 2, \ldots, 2^i$ until the first time a contaminated set is found. Namely, the answer for the first i tests was negative and the last answer was positive. At this stage, the algorithm detected $1+2+\cdots+2^{i-1} = 2^i - 1$ good items and found a contaminated set of size 2^i. Using a binary search this item can be detected by performing i additional tests. Since prior to the binary search the algorithm performed $i+1$ tests, it follows that for the price of $2i+1$ tests the algorithm learned about 2^i items. In other words, the status of a new items is known by performing $2\log a + 1$ tests.

The above described strategy, called the *doubling process*, is the heart of Algorithm DOUBLE that is depicted in Figure 1. However, this strategy by itself is not enough to guarantee a competitive ratio of 2. The reason for this is that when d is small, the algorithm may perform many unnecessary tests on a large pure set. In the extreme case where $d = 0$, the algorithm performs $\log n$ such unnecessary tests. To overcome this difficulty, the algorithm can test all the unknown items before it starts the doubling process. However, this solution achieves competitive ratio of only 2.16.

The final trick added to the algorithm is as follows. First the algorithm tests a set of size three. If the set is pure then the algorithm can test the rest of the items and if they are contaminated then the algorithm begins the doubling process. By testing three items instead of two sets one of two items and one of one item, the algorithm saved a test to spend on testing the rest of the items. If the set containing the three items is contaminated, then by two additional tests either two defective items are detected or a good item and a defective item are detected.

In Figure 1 a high level description of Algorithm DOUBLE is presented. During the run of the algorithm three sets of items are maintained:

1. U – the set of unknown items,
2. D – the set of defective items, and
3. G – the set of good items.

Initially, $U := \{1, \ldots, n\}$ is the set of all items where D and G are empty sets. These sets are updated accordingly when either a defective item is detected or good items are detected.

Testing a set of items is done by Procedure TEST that returns a positive answer for a contaminated set and a negative answer for a pure set. In addition, the algorithm calls three other procedures.

1. Procedure 3-TEST (Figure 2) – The input for this procedure is a contaminated set of three items. The procedure tests two items one at a time and finds either two defective items or one good item and one defective item.
2. Procedure BINARY-TEST (Figure 3) – The input for this procedure is a contaminated set of $0 < k \leq 2^i$ items, for $i \geq 2$. The procedure performs at most i tests and finds one defective item.
3. Procedure FINAL-TESTS (Figure 4) – This procedure tests the remaining items in the

set U. The maximum size of U at this stage is 2.

3 The performance of the algorithm

Consider the **while** loop in Figure 1. There are four possible flows of this loop:

Flow 1: A subset of three items is tested and found to be pure, and then the rest of the yet untested items are tested and found to be contaminated. Thereafter, $i-2$ ($i \geq 2$) disjoint sets of sizes $4, 8, \ldots, 2^{i-1}$ are tested and found to be pure, and then a subset of size at most 2^i is tested and found to be contaminated. Finally, Procedure BINARY-TEST is invoked and detects one defective item.

Flow 2: A subset of three items is tested and found to be contaminated. Then Procedure 3-TEST detects two defective items by performing two additional tests.

Flow 3: A subset of three items is tested and found to be contaminated. Then Procedure 3-TEST detects one defective item and one good item by performing two additional tests.

Flow 4: A subset of three items is tested and found to be pure, and then the rest of the yet untested items are tested and found to be pure.

Clearly, defective items are detected only in Flows 1-3 and by Procedure FINAL-TEST. During the run of the algorithm, each of these flows may occur several times. Let d_i be the total number of defective items that are detected in all occurrences of Flow i ($i = 1, 2, 3$) throughout the run of the algorithm. Let d be the total number of defective items. Since Procedure FINAL-TEST can detect at most two defective items, it follows that $d - 2 \leq d_1 + d_2 + d_3 \leq d$.

The following lemma is implied by the fact that exactly three tests are performed in both Flow 2 and Flow 3.

Lemma 3.1 *The total number of tests performed in all occurrences of Flows 2 and 3 during the run of the algorithm is* $1.5d_2 + 3d_3$, *and the total number of items that are identified by these tests is* $d_2 + 2d_3$.

The next lemma determines the relation between the number of good items found by Algorithm DOUBLE in Flow 1 and the total number of tests performed in this flow.

Lemma 3.2 *In each of the occurrences of Flow 1, the algorithm performs* $2\log a + 1$ *tests and detects one defective item and* $a - 1$ *good items, where* $a = 2^i$ *for some* $i \geq 2$.

Proof: The claim regarding the number of detected good items follows as $3+4+8+\cdots+2^{i-1} = 2^i - 1$. The claim regarding the number of tests follows since the number of tests prior to the call for Procedure BINARY-TEST is $i + 1$ and Procedure BINARY-TEST performs at most i additional tests to detect one defective item. □

Let $A_1, A_2, \ldots, A_{d_1}$ be the d_1 subsets each of which consists of $|A_j| - 1$ good items and one defective item, that are detected in the corresponding d_1 occurrences of Flow 1 during the run of the algorithm. Denote $|A_j|$ by a_j. Let $T(n, d)$ be the number of tests required by algorithm DOUBLE to identify among n items all d defective items.

The two claims of the next lemma follow directly from Lemma 3.1 and lemma 3.2 and the fact that Procedure FINAL-TEST performs at most two tests.

Lemma 3.3

a. $\sum_{j=1}^{d_1} a_j \leq n - d_2 - 2d_3$.

b. $T(n, d) \leq$

$$\sum_{j=1}^{d_1} (2\log a_j + 1) + 1.5d_2 + 3d_3 + 2.$$

The following is the key lemma of the performance analysis.

Lemma 3.4 *For* $0 < d \leq \frac{n}{2}$,

$$T(n, d) \leq 2d\log\frac{n}{d} + d + 2.$$

Proof: If $d_1 = 0$ then we have

$$\begin{aligned} T(n,d) &\leq 1.5d_2 + 3d_3 + 2 \leq 3d + 2 \\ &\leq 2d \log \frac{n}{d} + d + 2 \end{aligned}$$

where the first inequality follows from Lemma 3.3(b) and the last inequality follows from the fact that $d \leq n/2$.

Suppose now that $d_1 > 0$. By Lemma 3.3(b),

$$T(n,d) \leq \sum_{j=1}^{d_1+d_3} (2 \log c_j + 1) + 1.5d_2 + 2$$

where $c_j = a_j$ for $1 \leq j \leq d_1$ and $c_j = 2$ for $d_1 < j \leq d_1 + d_3$. From the convexity of $\log x$ it follows that

$$\begin{aligned} \sum_{j=1}^{d_1+d_3} \log c_j &\leq (d_1 + d_3) \log \frac{\sum_{j=1}^{d_1+d_3} c_j}{d_1 + d_3} \\ &= (d_1 + d_3) \log \frac{\sum_{j=1}^{d_1} a_j + 2d_3}{d_1 + d_3} \\ &\leq (d_1 + d_3) \log \frac{n - d_2}{d_1 + d_3} \end{aligned}$$

where the last inequality follows from Lemma 3.3(a). Using this we obtain that

$$\begin{aligned} T(n,d) &\leq 2(d_1 + d_3) \log \frac{n - d_2}{(d_1 + d_3)} + (d_1 + d_3) \\ &\quad + 1.5d_2 + 2 \\ &= 2(d_1 + d_3) \log \frac{(n - d') + (d_1 + d_3)}{(d_1 + d_3)} \\ &\quad - \frac{d_1 + d_3}{2} + \frac{3}{2}d' + 2 \\ &\triangleq G(d_1 + d_3) \end{aligned}$$

where $d' = d_1 + d_2 + d_3$. Since $G(x)$ is an increasing function of x in the interval $0 < x \leq n - d'$, it follows that

$$T(n,d) \leq G(d_1 + d_3) \leq G(d') = 2d' \log \frac{n}{d'} + d' + 2$$

Since $d - 2 \leq d' \leq d$, this bound implies that

$$\begin{aligned} &T(n,d) \leq \\ &\quad \max_{0 \leq i \leq 2} \left\{ 2(d - i) \log \frac{n}{d - i} + (d - i) + 2 \right\} \\ &\quad = 2d \log \frac{n}{d} + d + 2 \end{aligned}$$

where the equality follows from the fact that $d/n \leq 1/2 < \sqrt{2}/e$. □

The above lemma was restricted to the case $0 < d \leq \frac{n}{2}$. The next two lemmas give bounds for the remaining values of d.

Lemma 3.5 *For any d, $T(n,d) \leq 2n$.*

Proof: At the end of each flow the following claim holds: The number of tests performed so far is at most twice the number of items that were identified so far (good or defective). The claim is proved by induction. Initially the claim is true since both numbers equal 0. For Flow 1 the claim is true since $2i + 1 \leq 2 \cdot 2^i$ for any positive integer i. In both Flow 2 and Flow 3, three tests are performed and two items are identified. In Flow 4 two tests are performed and at least three items are identified. Finally, Procedure FINAL-TESTS performs exactly one test per item.

The lemma follows from the above claim, because at the end of the algorithm all the n items are detected either as good items or as defective items. □

We conclude with a trivial fact regarding the case $d = 0$.

Fact 3.1 $T(n,0) = 2$.

4 The competitive ratio of the algorithm

Let $M(n,d)$ be the number of tests performed by an optimal algorithm on n items where d is known in advance. The following three lemmas are lower bounds for $M(n,d)$ for all possible values for d. The first lemma is by definition, the second lemma is due to Hwang [Hw], and the last lemma is a variation of a lemma of Du and Hwang [DH2].

Fact 4.1 $M(n,0) = 0$.

Lemma 4.1 ([Hw]) *For $d > \frac{n}{2}$, $M(n,d) = n - 1$.*

(In [DH1], the above claim is proved to be true for $d \geq \frac{8n}{21}$).

Lemma 4.2 *For* $1 \leq d \leq \frac{n}{2}$,

$$M(n,d) \geq d \log \frac{n}{d} + 0.678d - 2.$$

Proof: When $d = 1$, clearly $M(n,d) \geq \log n$ and the result follows immediately. Lemma 1 in [DH2] states that for $0 < \rho < 1$ and $2 \leq d \leq \rho n$,

$$M(n,d) \geq d \log \frac{n}{d} + d \log(e\sqrt{1-\rho}) - \frac{\log d}{2} - 2.$$

For $\rho = \frac{1}{2}$,

$$\begin{aligned} M(n,d) &\geq d \log \frac{n}{d} + d\left(\log e + \frac{\log \frac{1}{2}}{2}\right) \\ &\quad - d\left(\frac{\log d}{2d}\right) - 2 \\ &\geq d \log \frac{n}{d} + 0.678d - 2 \end{aligned}$$

where the second inequality follows from the fact that the maximum of $\frac{\log d}{2d}$ for $d \geq 2$ is obtained when $d = 3$ and therefore $\frac{\log d}{2d} \leq \frac{\log 3}{6} \leq 0.264$. □

We are now ready to state the main result of the paper.

Theorem 4.1 *Algorithm* DOUBLE *is a 2-competitive algorithm for solving the group testing problem on n items when the number of defective items is not known in advance.*

Proof: We distinguish between three cases:

(*i*) $d = 0$: By Fact 3.1 and Lemma 4.1 it follows that $T(n,0) = 2M(n,0) + 2$.

(*ii*) $0 < d \leq \frac{n}{2}$: By Lemmas 3.4 and 4.2 it follows that $T(n,d) \leq 2M(n,d) + 6$.

(*iii*) $\frac{n}{2} < d \leq n$: By Lemmas 3.5 and 4.1 it follows that $T(n,d) = 2M(n,d) + 2$.

Putting all together, we get for $0 \leq d \leq n$,

$$T(n,d) \leq 2M(n,d) + 6.$$

□

5 Summary and Discussion

In this paper a group testing algorithm that does not require any a priori information on the number of defective items was presented, and was shown to have a competitive ratio of 2. This result is an improvement over a previous algorithm the competitive ratio of which is 2.75.

It is possible to relax the requirement for competitive algorithms: The competitive ratio is taken for fixed d and $n \to \infty$. With such a requirement, our algorithm remains a 2-competitive algorithm, whereas algorithm C in [DH2] achieves a competitive ratio that tends to 1. However, the competitive ratio of algorithm C without the above relaxation is 2.89.

The applications of group testing techniques in communication networks were so far mainly in the design of multiaccess algorithms. We now discuss a new application of group testing techniques in high speed networks.

Consider a communication network in which processors may fail and then recover. A basic task for a processor that has just recovered is to find which among the other processors are failed. The only way processor p can detect a failed processor q is by communicating with q. If q does not acknowledge p, then p can deduce that q is a failed processor. Moreover, p cannot obtain information on the status of processor q from other processors. The goal is to accomplish this task in minimum time. We call the above task the *fault detection* problem.

In high speed networks, processor p can send a message to itself along some path, and it takes one unit of time for the message to return to p, independent of the length of the path. If after one unit of time p does not receive the message, it deduces that at least one of the processors on the path is failed. To demonstrate the advantage of this technique, consider a fully connected network with $n+1$ processors when it is known that there exists exactly one failed processor. In conventional networks, in the worst-case a processor may poll all the other n processors before it finds the failed one, and therefore, the time complexity is n. In high speed networks, using the above idea, the failed processor may be found

by a binary search that takes $\log n + 1$ units of time.

Assuming that the above technique is used, the fault detection problem in a fully connected high speed network with $n + 1$ processors is clearly equivalent to the group testing problem on n items. When the network is not fully connected, any solution to the group testing problem is a solution to the fault detection problem. However, by utilizing information on the topology of the network, more efficient algorithms for the fault detection problem can be designed. The design of such algorithms is a subject for further research.

Acknowledgments

We would like to thank Moshe Sidi for valuable discussions.

References

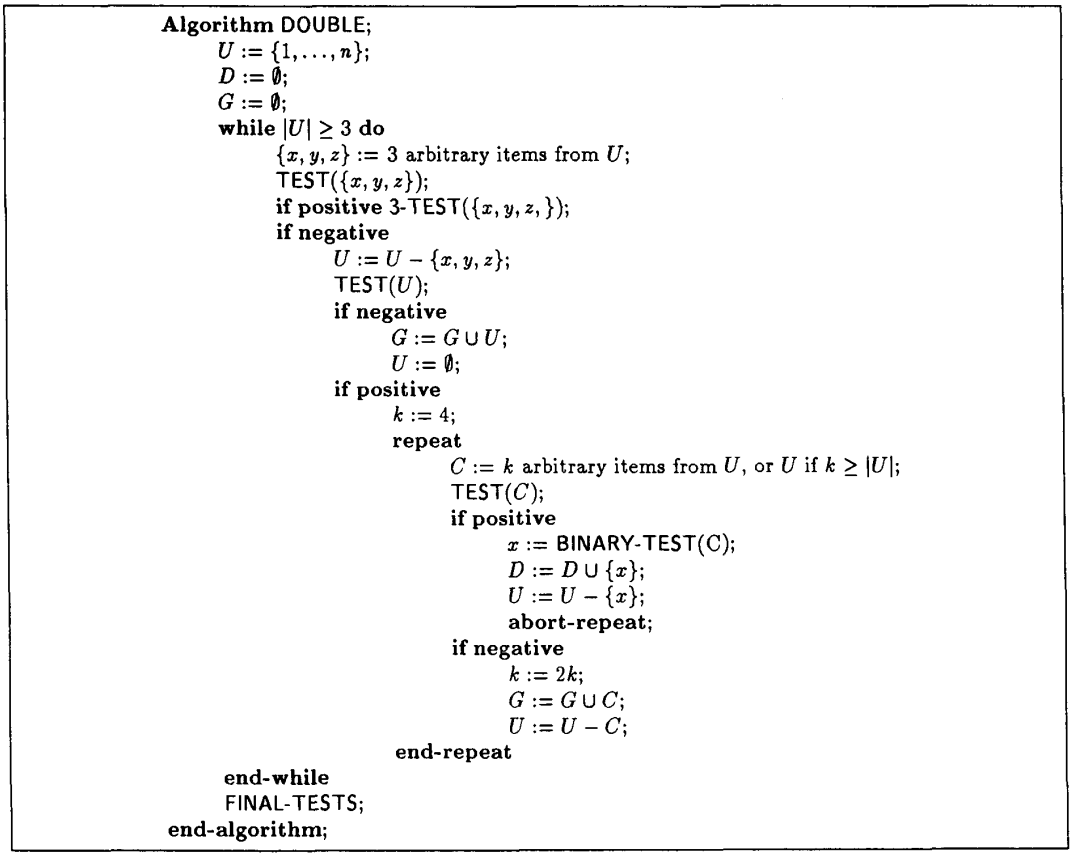
[Do] B. Dorfman, "The Detection of Defective Members of Large Population", *Annals of Math. Stat.*, Vol. 14, pp. 436 - 440, 1943.

[DH1] D. Z. Du and F. K. Hwang, "Minimizing a Combinatorial Function", *SIAM J. Alg. Disc. Meth.*, Vol. 3, pp. 523 - 528, 1982.

[DH2] D. Z. Du and F. K. Hwang, "Competitive Group Testing", To appear in *Discrete Applied Math.*; also AT&T Bell Laboratories Technical Memorandum, Dec. 1990; also presented in *DIMACS Workshop on On-Line Algorithms*, 1991.

[Hw] F. K. Hwang, "A minimax Procedure on Group Testing Problems", *Tamkang J. Math.*, Vol. 2, pp. 39 - 44, 1971.

[MMS] M. S. Manasse, L.A. McGeoch and D. D. Sleator, "Competitive Algorithms for On-Line Problems", *Proceedings of the 20th ACM Symposium on Theory of Computing*, pp. 322-333, 1988.



```
Algorithm DOUBLE;
    U := {1, ..., n};
    D := ∅;
    G := ∅;
    while |U| ≥ 3 do
        {x, y, z} := 3 arbitrary items from U;
        TEST({x, y, z});
        if positive 3-TEST({x, y, z, });
        if negative
            U := U − {x, y, z};
            TEST(U);
            if negative
                G := G ∪ U;
                U := ∅;
            if positive
                k := 4;
                repeat
                    C := k arbitrary items from U, or U if k ≥ |U|;
                    TEST(C);
                    if positive
                        x := BINARY-TEST(C);
                        D := D ∪ {x};
                        U := U − {x};
                        abort-repeat;
                    if negative
                        k := 2k;
                        G := G ∪ C;
                        U := U − C;
                end-repeat
    end-while
    FINAL-TESTS;
end-algorithm;
```

Figure 1: Algorithm DOUBLE for group testing on n items

```
Procedure 3-TEST({x, y, x});
    TEST({x});
    if positive
        D := D ∪ {x};
        TEST({y});
        if positive D := D ∪ {y};
        if negative G := G ∪ {y};
        U := U − {x, y};
    if negative
        G := G ∪ {x};
        TEST({y});
        if positive
            D := D ∪ {y};
            U := U − {x, y};
        if negative
            D := D ∪ {z};
            U := U − {x, z};
```

Figure 2: Procedure 3-TEST

```
Procedure BINARY-TEST(C);
    k := |C|;
    repeat
        C' := ⌊k/2⌋ arbitrary items from C;
        TEST(C');
        if positive C := C';
        if negative C := C − C';
        k := |C|;
    until k = 1;
    x := the only item in C;
    return (x);
end-procedure;
```

Figure 3: Procedure BINARY-TEST

```
Procedure FINAL-TESTS;
    while U ≠ ∅ do
        x := an arbitrary item from U;
        TEST({x});
        if positive D := D ∪ {x};
        if negative G := G ∪ {x};
        U := U − {x};
    end-while;
end-procedure;
```

Figure 4: Procedure FINAL-TESTS