A Fast Binary Splitting Approach to Non-Adaptive Group Testing

Eric Price
Department of Computer Science, University of Texas at Austin, TX, USA
`https://www.cs.utexas.edu/~ecprice/`
ecprice@cs.utexas.edu

Jonathan Scarlett
Department of Computer Science & Department of Mathematics,
National University of Singapore, Singapore
`https://www.comp.nus.edu.sg/~scarlett/`
scarlett@comp.nus.edu.sg

Abstract

In this paper, we consider the problem of noiseless non-adaptive group testing under the for-each recovery guarantee, also known as probabilistic group testing. In the case of n items and k defectives, we provide an algorithm attaining high-probability recovery with $O(k \log n)$ scaling in both the number of tests and runtime, improving on the best known $O(k^2 \log k \cdot \log n)$ runtime previously available for any algorithm that only uses $O(k \log n)$ tests. Our algorithm bears resemblance to Hwang's adaptive generalized binary splitting algorithm (Hwang, 1972); we recursively work with groups of items of geometrically vanishing sizes, while maintaining a list of "possibly defective" groups and circumventing the need for adaptivity. While the most basic form of our algorithm requires $\Omega(n)$ storage, we also provide a low-storage variant based on hashing, with similar recovery guarantees.

2012 ACM Subject Classification Theory of computation → Design and analysis of algorithms

Keywords and phrases Group testing, sparsity, sublinear-time decoding, binary splitting

Digital Object Identifier 10.4230/LIPIcs.APPROX/RANDOM.2020.13

Category RANDOM

Funding *Eric Price*: Supported in part by NSF Award CCF-1751040 (CAREER).
Jonathan Scarlett: Supported by an NUS Early Career Research Award.

1 Introduction

Group testing is a classical statistical problem with a combinatorial flavor [4, 19, 20], and has recently regained significant attention following new applications [14, 18, 25], connections with compressive sensing [26, 27], and most recently, utility in testing for COVID-19 [28, 42, 45]. The goal is to identify a defective (or infected) subset of items (or individuals) based on a number of suitably-designed tests, with the binary test outcomes indicating whether or not the test includes at least one defective item.

In both classical studies and recent works, considerable effort has been put into the development of group testing algorithms achieving a given recovery criterion with a near-optimal number of tests [1, 3, 5, 11, 15, 16, 20, 33, 35, 36, 39], and many solutions are known with decoding time linear or slower in the number of items. In contrast, works seeking to further reduce the decoding time have only arisen more recently [9, 10, 12, 30, 31, 34, 37]; see Section 1.2 for an overview.

In this paper, we present a non-adaptive group testing algorithm that guarantees high-probability recovery of the defective set with $O(k \log n)$ scaling in both the number of tests and the decoding time in the case of n items and k defectives. By comparison, the best previous decoding time alongside $O(k \log n)$ tests was $O(k^2 \log k \cdot \log n)$ [9], with faster algorithms incurring suboptimal $O(k \log k \cdot \log n)$ scaling in the number of tests [10,34]. By a standard information-theoretic lower bound [4, Sec. 1.4], the $O(k \log n)$ scaling in the number of tests is optimal whenever $k \le n^{1-\Omega(1)}$.

Before outlining the related work and contributions in more detail, we formally introduce the problem.

1.1 Problem Setup

We consider a set of n items indexed by $\{1, \dots, n\}$, and let $S \subset \{1, \dots, n\}$ be the set of defective items. The number of defectives is denoted by $k = |S|$; this is typically much smaller than n, so we assume that $k \le \frac{n}{2}$. For clarity of exposition, we will present our algorithm and analysis for the case that k is known, but these will trivially extend to the case that only an upper bound $\overline{k} \ge k$ is known, and $\overline{k}$ replaces k in the number of tests and decoding time.

A sequence of t tests $X^{(1)}, \dots, X^{(t)}$ is performed, with $X^{(i)} \in \{0,1\}^n$ indicating which items are in the i-th test, and the resulting outcomes are $Y^{(i)} = \bigvee_{j \in S} X_j^{(i)}$ (i.e., 1 if there is any defective item in the test, 0 otherwise). We focus on the non-adaptive setting, in which all tests $X^{(1)}, \dots, X^{(t)}$ must be designed prior to observing any outcomes.

We consider a for-each style recovery guarantee, in which the goal is to develop a randomized algorithm that, for any fixed defective set S of cardinality k, produces an estimate $\widehat{S}$ satisfying $\mathbb{P}[\widehat{S} \ne S] \le \delta$ for some small $\delta > 0$ (typically $\delta \to 0$ as $n \to \infty$). In our algorithm, only the tests $\{X^{(i)}\}_{i=1}^t$ will be randomized, and the decoding algorithm producing $\widehat{S}$ from the test outcomes will be deterministic.

1.2 Related Work

The existing works on group testing vary according to the following defining features [4,20]:

- **For-each vs. for-all guarantees.** In *combinatorial (for-all) group testing*, one seeks to construct a test design that guarantees the recovery of *all* defective sets up to a certain size. In contrast, in *probabilistic (for-each) group testing*, the test design may be randomized, and the algorithm is allowed some non-zero probability of error.
- **Adaptive vs. non-adaptive.** In the *adaptive* setting, each test may be designed based on all previous outcomes, whereas in the *non-adaptive setting*, all tests must be chosen prior to observing any outcomes. The non-adaptive setting is often preferable in practice, as it permits the tests to be performed in parallel.
- **Noiseless vs. noisy.** In the *noiseless* setting, the test outcomes are perfectly reliable, whereas in *noisy settings*, some tests may be flipped according to some probabilistic or adversarial noise model.

As outlined in the previous subsection, our focus is on the for-each guarantee, non-adaptive testing, and noiseless tests, but for comparison, we also discuss other variants in this section.

By a simple counting argument, even in the least stringent scenario of noiseless adaptive testing and the for-each guarantee, $\Omega\big(k \log \frac{n}{k}\big)$ tests are required for reliable recovery [6,36]. In the noiseless adaptive setting, the classical generalized binary splitting algorithm of Hwang [29] matches this bound with sharp constant factors, and attains the stronger for-all guarantee. More recently, $O(k \log n)$ adaptive tests and decoding time were shown to suffice under the for-each guarantee in the presence of random noise [10].

Table 1 Overview of existing noiseless non-adaptive group testing results with $\text{poly}(k \log n)$ decoding time under the for-each guarantee, with n items and k defectives.

Reference	Number of tests	Decoding time	Construction
SAFFRON [34]	$O(k \cdot \log k \cdot \log n)$	$O(k \log k)^1$	Randomized
GROTESQUE [10]	$O(k \cdot \log k \cdot \log n)$	$O(k \cdot \log k \cdot \log n)$	Randomized
Inan *et al.* [30]	$O\big(k \cdot \log n \cdot \log \frac{\log n}{\log k}\big)$	$O\big(k^3 \cdot \log n \cdot \log \frac{\log n}{\log k}\big)$	Explicit
BMC [9]	$O(k \log n)$	$O(k^2 \cdot \log k \cdot \log n)$	Randomized
This Paper	$O(k \log n)$	$O(k \log n)$	Randomized

In the non-adaptive setting with the for-each guarantee, a recent of result of [2] shows that n tests are required to attain asymptotically vanishing error probability as $n \to \infty$ with $k = \lfloor \beta n \rfloor$, for arbitrarily small $\beta > 0$. Thus, $O(k \log \frac{n}{k})$ scaling (with a universal implied constant) is impossible; see also [16]. On the other hand, $O(k \log n)$ tests do suffice, and this scaling is equivalent to $O(k \log \frac{n}{k})$ in the regime $k \le n^{1-\Omega(1)}$. Early works achieved this for the scaling regime $k = O(1)$ [5, 35, 36], and a variety of more recent works considered more general k [1, 5, 15, 16, 33, 35, 36, 39], particularly under "unstructured" random test designs such as i.i.d. Bernoulli [3, 5, 39] and near-constant tests-per-item [15, 33]. In fact, following the recent work of Coja-Oghlan et al. [16], the precise constants for non-adaptive group testing have been nailed down in the regime $k \le n^{1-\Omega(1)}$, with the algorithm for the upper bound requiring $\Omega(n)$ decoding time.

Since non-adaptive algorithms with $\text{poly}(k \log n)$ decoding time under the for-each guarantee are particularly relevant to the present paper, we provide a summary of the existing works in Table 1. We separate our discussion according to whether or not each algorithm attains $O(k \log n)$ scaling in the number of tests, as we believe this to be a particularly important desiderata in practice when tests are expensive:

- SAFFRON and GROTESQUE both use $O(k \log k \cdot \log n)$ tests, which is suboptimal by a $\log k$ factor. In particular, SAFFRON's decoding time is as low as $O(k \log k)$ in the word-RAM model.[1] While storage requirements were not a significant point of focus in the existing works, we also note (for later comparison) that SAFFRON only requires $O(k \log k \cdot \log n)$ bits of storage; see Appendix A for details.
- In the regime $k = \Theta(n^\alpha)$ with fixed $\alpha \in (0, 1)$, Inan et al. [30] attain $O(k^3 \log n)$ decoding time with $O(k \log n)$ tests, whereas the number of tests becomes suboptimal in sparser regimes such as $k = \text{poly}(\log n)$. The best previous decoding time for an algorithm that only uses $O(k \log n)$ tests is $O(k^2 \log k \cdot \log n)$, via bit-mixing coding (BMC) [9]. To our knowledge, all other existing algorithms succeeding with $O(k \log n)$ tests incur $\Omega(n)$ decoding time [1, 5, 15, 16, 33, 35, 36, 39].

Our main theoretical contribution is to provide an algorithm that attains $O(k \log n)$ scaling in both the number of tests and decoding time, as well as using distinct algorithmic ideas from the existing works (see Section 2.1 for discussion). In particular, among the algorithms using $O(k \log n)$ tests, ours reduces the dependence on k in the decoding time from quadratic to linear.

[1] The decoding time of SAFFRON is typically stated as $O(k \log k \cdot \log n)$, but can be $O(k \log k)$ in the word-RAM model of computation, where the test outcomes are stored as $O(k \log k)$ words of length $\log_2 n$ each; see Appendix A for details.

It is worth noting that the works [9, 10, 30, 34] also extend their guarantees to noisy settings, and the approach in [30] has the additional advantage of using an *explicit* test design, i.e., the test vectors $X^{(1)}, \dots, X^{(t)}$ can be constructed deterministically in time polynomial in n and t. Although noisy and/or de-randomized variants of our algorithm may be possible, this is deferred to future work.[2]

In addition, [34] demonstrated that $O\big(k \log n \cdot \log \frac{1}{\epsilon}\big)$ tests suffice for SAFFRON in the case that one is only required to identify a fraction $1-\epsilon$ of the defectives, as opposed to the entire defective set. Thus, $O(k \log n)$ scaling is maintained for approximate recovery with constant $\epsilon > 0$, but not for the more common requirement of exact recovery.

While we have focused our discussion on the for-each setting, the first poly$(k \log n)$-time non-adaptive group testing algorithms were for the more stringent for-all setting [12, 31, 37]. The strength of the for-all guarantee comes at the expense of requiring significantly more tests, e.g., see [23] for an $\Omega\big(\min\big\{k^2 \frac{\log n}{\log k}, n\big\}\big)$ lower bound. An $O(k^2 \log n)$ upper bound on the number of tests was originally attained with $\Omega(n)$ decoding time, [22, 38], and more recently with poly$(k \log n)$ decoding time [31, 37]. The earlier work of [12] attained poly$(k \log n)$ decoding time under a list decoding recovery criterion, and also allowed for adversarial noise in the test outcomes.

1.3 Summary of Results

Before summarizing our main results, we briefly highlight that our algorithmic techniques are distinct from the existing works summarized above (see Section 2.1 for further details), and are more closely related to the adaptive binary splitting approach of Hwang [29]. We first test large groups of items together, placing each group into a single randomly-chosen test among a sequence of $O(k)$ tests. We then "split" these groups into smaller sub-groups, while using the $O(k)$ test outcomes to eliminate those known to be non-defective. This process is repeated (with the elimination step ensuring that the number of groups under consideration does not grow too large) until a superset of S is found. This superset is shown to be of size $O(k)$ with high probability, and S is deduced from this superset via a final sequence of $O(k \log k)$ tests (similar ideas to this final step have appeared in works such as [12, 37]). Despite the sequential nature of this procedure, the tests can be performed non-adaptively.

Our first main result is informally stated as follows; see Theorem 1 for a formal statement.

▶ **Main Result 1** (Algorithmic Guarantees – Informal Version). *There exists a non-adaptive group testing algorithm that, for any constant $c > 0$, succeeds with probability $1 - O(k^{-c})$ using $O(k \log n)$ tests and $O(k \log n)$ decoding time, and requires $O(n \log^2 k)$ bits of storage.*

Main Result 1 uses $\Omega(n)$ storage to record the random choices of tests the items are included in. For our second main result, we consider a low-storage variant in which the tests are instead allocated using hash functions. In this case, the precise decoding time and storage guarantees depend on the properties of the hash family used. The following theorem informally states the guarantee resulting from $O(\log k)$-wise independent hashing using the classical hash family of Wegman and Carter [44]; see Theorem 11 for a formal and more general statement.

▶ **Main Result 2** (Algorithmic Guarantees with Reduced Storage – Informal Version). *There exists a non-adaptive group testing algorithm that, for any constant $c > 0$, succeeds with probability $1 - O\big(\frac{\log n}{k^c}\big)$ using $O(k \log n)$ tests and $O(k \log k \cdot \log n)$ decoding time, and requires $O(k \log n + \log k \cdot \log^2 n)$ bits of storage.*

[2] Our analysis can easily be adapted to handle false positive tests, but handling false negative tests appears to require more significant changes and a slightly increased decoding time.

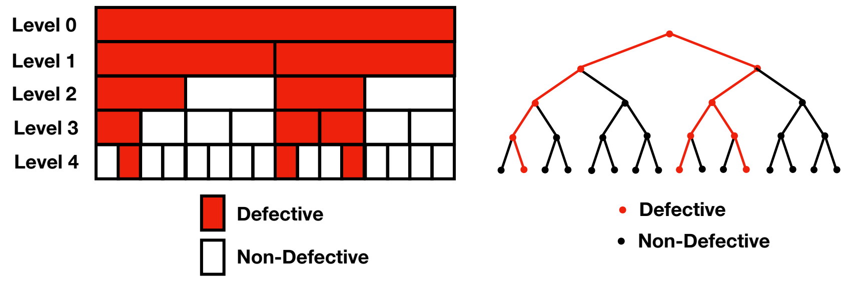

Figure 1 Example structure of groups of items that are tested together when $n = 16$ and $k = 3$ (Left), and the corresponding tree representation (Right). Leaves correspond to single items, and nodes at the higher levels correspond to groups of items.

2 Non-Adaptive Binary Splitting Algorithm

In this section, we introduce our algorithm, and state and prove our main result giving guarantees on its correctness, number of tests, decoding time, and storage. For simplicity of notation, we assume throughout the analysis that k and n are powers of two. Our algorithm only requires an upper bound on the number of defectives, and hence, any other value of k can simply be rounded up to a power of two. In addition, the total number of items n can be increased to a power of two by adding "dummy" non-defective items.

2.1 Description of the Algorithm

We propose an algorithm that works with a tree structure, as illustrated in Figure 1. On the left of the figure, we have $1 + \log_2 n$ levels, with the top level containing all items, the second level containing two groups with half the items each, and so on, with each level "splitting" the previous levels' groups in half until the final level containing individual items. The ordering of items is inconsequential for our analysis and results provided that the two children of a given node can be identified in constant time, so for convenience we assume the natural order.[3] For $\ell = 0, \dots, \log_2 n$, the j-th group at the ℓ-th level is denoted by $G_j^{(\ell)} \subseteq \{1, \dots, n\}$, and we observe that for fixed ℓ, there are 2^ℓ groups $G_1^{(\ell)}, \dots, G_{2^\ell}^{(\ell)}$ of cardinality $\frac{n}{2^\ell}$ each.

We consider the binary tree representation in Figure 1 (Right), in which each node corresponds to a group of items (e.g., the root corresponds to the set of all items, and the leaves correspond to individual items). The levels of the tree are indexed by $\ell = 0, \dots, \log_2 n$. Our algorithm works down the tree one level at a time, keeping a list of *possibly defective* (PD) nodes, and performing tests to obtain such a list at the next level, while ideally keeping the size of the list small (e.g., $O(k)$). When we perform tests at a given level, we treat each node as a "super-item"; including a node in a test amounts to including all of the items in the corresponding group $G_j^{(\ell)}$. While this may appear to naturally lead to an adaptive algorithm, we can in fact perform all of the tests non-adaptively.

If we were to test nodes at the root or the early levels, they would almost all return positive, and hence not convey significant information. We therefore let the algorithm start at the level $\ell_{\min} = \log_2 k$, and we consider the following test design:

[3] Alternatively, one may use a dyadic splitting approach: Assign each item a unique $(\log_2 n)$-bit string, and first split according to the first bit, then the second bit, and so on.

- For each $\ell = \log_2 k, \dots, \log_2 n - 1$, form a sequence of Ck tests (for some $C > 0$ to be selected later), and for each $j = 1, \dots, 2^\ell$, choose a single such test uniformly at random, and place all items from $G_j^{(\ell)}$ into that test.
- For the final level $\ell = \log_2 n$, each $G_j^{(\ell)} = \{j\}$ is a singleton. In this case, we form $C' \log k$ sequences of $2k$ tests (for some $C' > 0$). For each item, and each of the $C' \log k$ sequences of tests, we place the item in one of the $2k$ corresponding tests, chosen uniformly at random.

This creates a total of $t = Ck \log_2 \frac{n}{k} + 2C'k \log k = O(k \log n)$ tests.

Upon observing the t non-adaptive test outcomes, the decoder forms an estimate $\widehat{S}$ of the defective set via the following procedure:

- Initialize $\mathcal{G}^{(\ell_{\min})} = \{G_j^{(\ell_{\min})}\}_{j=1}^k$, where $\ell_{\min} = \log_2 k$;
- Iterate the following for $\ell = \log_2 k, \dots, \log_2 n - 1$:
  - For each group $G \in \mathcal{G}^{(\ell)}$, check whether the single test corresponding to that group is positive or negative. If positive, then add both children of G (see Figure 1) to $\mathcal{G}^{(\ell+1)}$.
- Let the estimate $\widehat{S}$ of the defective set be the (singleton) elements of $\mathcal{G}^{(\log_2 n)}$ that are not included in any negative test among the $2C'k \log k$ tests at the final level.

Comparisons with existing methods

Our algorithm is distinct from existing sublinear-time non-adaptive group testing algorithms, which are predominantly based on code concatenation [31,37] or related methods that encode item indices' binary representations directly into the test matrix [8,10,34]. In fact, in the context of group testing, our approach is perhaps most reminiscent of the *adaptive* algorithm of Hwang [29] (see also [4,20]), which identifies one defective at a time using binary splitting, removing each defective from consideration after it is identified. By keeping track of the multiple defective nodes simultaneously and allowing a small number of false positives up to the final level, we are able to exploit similar ideas without requiring adaptivity.

Beyond group testing and binary splitting, this recursive tree-structured approach is also reminiscent of ideas used in sketching algorithms, such as dyadic tree recovery for count-min sketch [17]. Our particular approach – maintaining a superset at each level, and relying on a lack of false negatives – is most similar to the pyramidal reconstruction method for compressive sensing under the earth mover's distance [32].

A nearly identical algorithm to ours was given independently in the concurrent work of [13], with similar theoretical guarantees. In addition, [13] observes that since the total number of non-discarded nodes in the algorithm is $O\big(k \log \frac{n}{k}\big)$ with overwhelming probability (see Lemma 6 below), one can take a union bound over the $\binom{n}{k}$ possible defective sets to attain *list recovery* in the combinatorial (for-all) setting. This idea allows the authors of [13] to attain a variety of other results, including in the related problems of heavy hitters and compressive sensing. In the context of probabilistic (for-each) group testing, some slight advantages of our analysis include (i) proving that the first batch of tests leads to $O(k)$-size list recovery rather than $O\big(k \log \frac{n}{k}\big)$-size, thus circumventing the need for a third batch of tests, and (ii) only requiring $O(\log k)$-wise independence in our low-storage variant based on hashing (see Section 3), rather than $O\big(k \log \frac{n}{k}\big)$-wise independence (albeit at the expense of a higher error probability).

2.2 Algorithmic Guarantees

In the following, we provide the formal version of Main Result 1. Here and subsequently, we assume a word-RAM model of computation; for instance, with n items and t tests, it takes constant time to read a single integer in $\{1, \dots, n\}$ from memory, perform arithmetic operations on such integers, fetch a single test outcome indexed by $\{1, \dots, t\}$, and so on.

▶ **Theorem 1** (Algorithmic Guarantees). *Let S be a fixed (defective) subset of $\{1,\dots,n\}$ of cardinality k. For any constant $c>0$, there exist choices of C and C' such that the group testing algorithm described in Section 2.1 satisfies the following with probability $1-O(k^{-c})$:*

- *The returned estimate $\widehat{S}$ equals S;*
- *The algorithm runs in time $O(k\log n)$;*
- *The algorithm uses $O(n(\log k)^2)$ bits of storage.*

In addition, the number of tests scales as $O(k\log n)$.

We note that the success probability approaches one as $n\to\infty$ in any asymptotic regime satisfying $k=\omega(1)$, but not in the regime $k=O(1)$; the same is true in the existing works listed in Table 1, with the exception of [9]. In addition, it is straightforward to improve the success probability to $1-e^{-\Omega(k)}-O(n^{-c})$ by using $O(\log n)$ (rather than $O(\log k)$) sequences of $2k$ tests at the final level.

Theorem 1 is proved in the remainder of the section. We emphasize that our focus is on scaling laws, and we make no significant effort to optimize the underlying constant factors.

2.3 Analysis

Throughout the analysis, the defective set S will be fixed but otherwise arbitrary, and we will condition on fixed placements of the defective items into tests (and hence, fixed test outcomes). The test placements of the non-defective items are independent of those of the defectives, and our analysis will hold regardless of which particular tests the defectives were placed in. We use $\mathbb{P}[\cdot\,|\,\mathcal{T}_S]$ to represent this conditioning, and we refer to $\mathcal{T}_S$ as the defective test placements. In addition, for the tree illustrated in Figure 1, we refer to nodes containing defective items as defective nodes, to all other nodes as non-defective nodes, and to the set of defective nodes as the defective (sub-)tree.[4]

We begin with the following simple lemma.

▶ **Lemma 2** (Probabilities of Non-Defectives Being in Positive Tests). *Under the test design described in Section 2.1, the following holds at any given level $\ell=\log_2 k,\dots,\log_2 n-1$: Conditioned on any defective test placements $\mathcal{T}_S$, any given non-defective node at level ℓ has probability at most $\frac{1}{C}$ of being placed in a positive test.*

Proof. Since there are k defective items, at most k nodes at a given level can be defective, and since each node is placed in a single test, at most k tests out of the Ck tests at the given level can be positive. Since the test placements are independent and uniform, it follows that for any non-defective node, the probability of being in a positive test is at most $\frac{1}{C}$. ◀

In view of this lemma, when starting at any non-defective child of any given defective node (or alternatively, starting at a non-defective node at level $\ell_{\min}$), we can view any further branches down the non-defective sub-tree as "continuing" (i.e., the two children remain "possibly defective") with probability at most $\frac{1}{C}$. Hence, we have the following.

▶ **Lemma 3** (Probability of Reaching a Non-Defective Node). *Under the setup of Lemma 2, any given non-defective node having Δ non-defective ancestor nodes is reached (i.e., all of its ancestor nodes are placed in positive tests, so the node is considered possibly defective) with probability at most $\left(\frac{1}{C}\right)^{\Delta}$.*

We will use the preceding lemmas to control the following two quantities:

[4] Since we start at level $\ell_{\min}=\log_2 k$, this could more precisely be considered as a forest.

- N_{total}, the total number of non-defective nodes that are reached in the sense of Lemma 3;
- N_{leaf}, the number of non-defective leaf nodes that are reached, and thus are still considered possibly defective by the final level.

It will be useful to upper bound N_{total} for the purpose of controlling the overall decoding time, and to upper bound N_{leaf} for the purpose of controlling the number of items considered at the final level.

2.3.1 Bounding N_{total} and N_{leaf} on average

We first present two lemmas bounding the averages of N_{total} and N_{leaf}, and then establish high-probability bounds.

▶ **Lemma 4** (Bounding N_{total} on Average). *For any $C \geq 4$ and any defective test placements $\mathcal{T}_S$, we have*

$$\mathbb{E}[N_{\text{total}} \,|\, \mathcal{T}_S] \leq 6k \log_2 \frac{n}{k}. \tag{1}$$

Proof. Since the tree is binary, each defective node can have at most 2^i non-defective descendants appearing exactly i levels further down the tree; such descendants correspond to $\Delta = i-1$ in Lemma 3. Since there are at most $k \log_2 \frac{n}{k}$ defective nodes, it follows that there are at most $\big(k \log_2 \frac{n}{k}\big) 2^{\Delta+1}$ non-defective nodes with Δ non-defective ancestors and at least one defective ancestor. Similarly, there are at most k non-defective nodes at level $\ell_{\min} = \log_2 k$, each of which has at most 2^{Δ} non-defective nodes appearing Δ levels later.

Since Lemma 3 demonstrates a probability $\big(\frac{1}{C}\big)^{\Delta}$ of being reached for a given number Δ of non-defective ancestors, it follows that

$$\mathbb{E}[N_{\text{total}} \,|\, \mathcal{T}_S] \leq \Big(k \log_2 \frac{n}{k}\Big) \sum_{\Delta=0}^{\log_2 \frac{n}{k} - 1} 2^{\Delta+1} \Big(\frac{1}{C}\Big)^{\Delta} + k \sum_{\Delta=0}^{\log_2 \frac{n}{k}} 2^{\Delta} \Big(\frac{1}{C}\Big)^{\Delta}. \tag{2}$$

Hence, the assumption $C \geq 4$ gives

$$\mathbb{E}[N_{\text{total}} \,|\, \mathcal{T}_S] \leq \Big(2k \log_2 \frac{n}{k}\Big) \sum_{\Delta=0}^{\log_2 \frac{n}{k} - 1} 2^{-\Delta} + k \sum_{\Delta=0}^{\log_2 \frac{n}{k}} 2^{-\Delta} \leq 6k \log_2 \frac{n}{k}, \tag{3}$$

where we used $\sum_{\Delta=0}^{\infty} 2^{-\Delta} = 2$ and $\log_2 \frac{n}{k} \geq 1$ (for $k \leq \frac{n}{2}$). ◀

▶ **Lemma 5** (Bounding N_{leaf} on Average). *For any $C \geq 4$ and any defective test placements $\mathcal{T}_S$, we have*

$$\mathbb{E}[N_{\text{leaf}} \,|\, \mathcal{T}_S] \leq 6k. \tag{4}$$

Proof. Again using the fact that each defective node can have at most 2^i descendants appearing i levels further down the tree, we find that there are at most $k2^{\Delta+1}$ leaf nodes with Δ non-defective ancestors and at least one defective ancestor. In addition, a leaf having only non-defective ancestors corresponds to $\Delta = \log_2 \frac{n}{k}$ in Lemma 3 (since $\ell_{\min} = \log_2 k$), and to handle this case, we trivially upper bound he number of leaves by n. Hence, for $C \geq 4$, we have similarly to (3) that

$$\mathbb{E}[N_{\text{leaf}} \,|\, \mathcal{T}_S] \leq 2k \sum_{\Delta=0}^{\log_2 \frac{n}{k} - 1} 2^{\Delta} \Big(\frac{1}{C}\Big)^{\Delta} + n \Big(\frac{1}{C}\Big)^{\log_2 \frac{n}{k}} \leq 6k. \tag{5}$$

◀

2.3.2 Bounding N_{total} with high probability

In this subsection, we prove the following.

▶ **Lemma 6** (High-Probability Bound on N_{total}). *For any $C \geq 16$, conditioned on any defective test placements $\mathcal{T}_S$, we have $N_{\text{total}} = O\big(k \log \frac{n}{k}\big)$ with probability $1 - e^{-\Omega(k \log \frac{n}{k})}$.*

To prove this result, we make use of Lemmas 2 and 4, along with the following auxiliary result written in generic notation.

▶ **Lemma 7** (Sub-Exponential Behavior in a Branching Process). *Consider an infinite-depth binary tree in which each node is independently assigned a value $\{0,1\}$ with probability q of being 1, and let N be the number of nodes whose ancestors are all marked as 1. Then, when $q \leq \frac{1}{16}$, we have all $n > 0$ that $\mathbb{P}[N = n] \leq 2^{-(n-1)}$.*

Proof. We make use of branching process theory [24, Ch. XII], in particular noting that N equals the *total progeny* [21] when the initial population size is 1 (i.e., the root) and the distribution of the number of children per node is given by

$$P_X(0) = 1 - q, \quad P_X(2) = q, \quad P_X(x) = 0, \forall x \notin \{0,2\}. \tag{6}$$

The results we use from branching process theory are expressed in terms of the generative function $M_X(s) = \mathbb{E}[s^X]$, which is given by

$$M_X(s) = (1-q) + qs^2. \tag{7}$$

It is evident in our case that $N < \infty$ with probability one provided that $q < \frac{1}{2}$, and one way to formally prove this is to utilize the general result that the *extinction probability* $\mathbb{P}[N < \infty]$ equals one provided that the smallest root of $s = M_X(s)$ is $s = 1$ [24, Sec. XII.4].

We utilize an exact expression for the distribution of N for general branching processes [21] (specialized to the case that the initial population size is one):

$$\mathbb{P}[N = n] = \frac{1}{n} m_{n-1}^{(n)}, \tag{8}$$

where $m_{n-1}^{(n)}$ is computed according to the expansion

$$\big(M_X(s)\big)^n = \sum_{j=0}^{\infty} m_j^{(n)} s^j. \tag{9}$$

Substituting our expression (7) for $M_X(s)$ on the left-hand side, we find that $(M_X(s))^n = \big((1-q) + qs^2\big)^n$, which equals $\sum_{i=0}^n \binom{n}{i}(1-q)^{n-i}(qs^2)^i$ by a binomial expansion. Hence, we obtain $m_{n-1}^{(n)} = 0$ for even-valued n, and for odd-valued n we substitute $i = \frac{n-1}{2}$ to obtain

$$m_{n-1}^{(n)} = \binom{n}{\frac{n-1}{2}}(1-q)^{\frac{n+1}{2}} q^{\frac{n-1}{2}} \tag{10}$$

$$\leq 2(2\sqrt{q})^{n-1}, \tag{11}$$

where we applied $\binom{n}{\frac{n-1}{2}} \leq 2^n = 2 \cdot 2^{n-1}$ and $(1-q)^{\frac{n+1}{2}} \leq 1$.

Substituting (11) into (8), we obtain

$$\mathbb{P}[N = n] \leq \frac{2}{n}\big(\sqrt{4q}\big)^{n-1} \mathbb{1}\{n \text{ is odd}\}. \tag{12}$$

This implies $\mathbb{P}[N = n] \leq 2^{-(n-1)}$ when $q \leq \frac{1}{16}$ and $n \geq 2$, and the same trivially holds when $n = 1$. ◀

The condition $\mathbb{P}[N = n] \leq 2^{-(n-1)}$ in Lemma 7 implies that N is a sub-exponential random variable, and the same trivially follows for a branching process that only runs up to a certain depth (number of generations). In the group testing setup of Lemma 6, we are adding together $O\big(k \log \frac{n}{k}\big)$ independent copies of such random variables (each corresponding to a different non-defective sub-tree) to get N_{total}. As a result, we can apply a standard concentration bound for sums of independent sub-exponential random variables [43, Prop. 5.16] to obtain

$$\mathbb{P}[N_{\text{total}} \geq \mathbb{E}[N_{\text{total}} \,|\, \mathcal{T}_S] + t \,|\, \mathcal{T}_S] \leq e^{-\Omega(\min\{t^2/(k \log_2 \frac{n}{k}), t\})}, \tag{13}$$

from which Lemma 6 follows by setting $t = \Theta\big(k \log_2 \frac{n}{k}\big)$ and using $\mathbb{E}[N_{\text{total}} \,|\, \mathcal{T}_S] = O\big(k \log_2 \frac{n}{k}\big)$ (see Lemma 4). The condition $C \geq 16$ coincides with $q \leq \frac{1}{16}$ in Lemma 7.

2.3.3 Bounding N_{leaf} with high probability

In this subsection, we prove the following.

▶ **Lemma 8** (High-Probability Bound on N_{leaf}). *For any $C \geq 12$, conditioned on any defective test placements $\mathcal{T}_S$, we have $N_{\text{leaf}} = O(k)$ with probability $1 - e^{-\Omega(k)}$.*

We again use Lemma 2, along with the following auxiliary results written in generic notation.

▶ **Lemma 9** (Tail Bound on Binary Tree Paths). *Consider a binary tree of height h in which each node is independently assigned a value $\{0,1\}$ with probability q of being 1, and let N_h be the number of leaf nodes that have a path of 1's back to the root (including both endpoints). Then $\mathbb{P}[N_h \geq t] \leq 4^{-(h+t)}$ for any integer $t \geq 1$.*

Proof. We use a proof by induction. The base case is $h = 1$, in which case we have $\mathbb{P}[N_h > 2] = 0$, $\mathbb{P}[N_h = 2] = \frac{1}{q^3}$, and $\mathbb{P}[N_h \geq 1] \leq \frac{2}{q^2}$ by the union bound. These bounds satisfy the claim of the lemma due to the assumption $q \leq \frac{1}{12}$.

Now fix $h \geq 2$ and suppose that the claim is true for height $h-1$. Let L and R be the number of leaf nodes reached in the left and right sub-trees of the root, and observe that

$$\mathbb{P}[N_h \geq t] \leq \sum_{j=0}^{t} \mathbb{P}[L \geq j \cap R \geq t-j] \tag{14}$$

$$= \mathbb{P}[L \geq t] + \mathbb{P}[R \geq t] + \sum_{j=1}^{t-1} \mathbb{P}[L \geq j \cap R \geq t-j]. \tag{15}$$

Applying the induction hypothesis, we find that the first two terms are at most $q4^{-(h-1+t)} = 4q \cdot 4^{-(h+t)}$, and for $1 \leq j \leq t-1$, we have $\mathbb{P}[L \geq j \cap R \geq t-j] \leq q \cdot 4^{-(h-1+j)} \cdot 4^{-(h-1+t-j)} = 16q \cdot 4^{-(2h+t)}$, where the multiplications by q correspond to the root node being marked as 1. Substituting back into (15) gives

$$\mathbb{P}[N_h \geq t] \leq 8q4^{-(h+t)} + 16qt4^{-(2h+t)}. \tag{16}$$

We may assume that $t \leq 2^h$, since for $t > 2^h$ we trivially have $\mathbb{P}[N_h \geq t] = 0$. Hence, the above bound simplifies to

$$\mathbb{P}[N_h \geq t] \leq 8q4^{-(h+t)} + 16q4^{-(h+t)}2^{-h} \leq 4^{-(h+t)}, \tag{17}$$

since $h \geq 2$ implies $2^{-h} \leq \frac{1}{4}$, and we have assumed $q \leq \frac{1}{12}$. ◀

▶ **Lemma 10** (Sub-Exponential Behavior for Binary Tree Paths). *Under the setup of Lemma 9, N_h satisfies $\mathbb{E}[e^{\lambda N_h}] \le 1 + 4^{-h}$ for $\lambda \le \log 2$. In addition, for any integer $h_{\max} \ge 1$, if $N_1, \dots, N_{h_{\max}}$ are independent random variables with the same distribution as N_h for the specified height, then $N = \sum_{h=1}^{h_{\max}} N_h$ satisfies $\mathbb{E}[e^{\lambda N}] \le 2$ for $\lambda \le \log 2$.*

Proof. We seek to upper bound $\mathbb{E}[e^{\lambda N_h}] = \sum_{t=0}^{\infty} \mathbb{P}[N_h = t] e^{\lambda t}$. Separating out the $t = 0$ term and applying Lemma 9, we obtain $\mathbb{E}[e^{\lambda N_h}] \le 1 + \sum_{t=1}^{\infty} 4^{-(h+t)} e^{\lambda t} = 1 + 4^{-h} \sum_{t=1}^{\infty} e^{(\lambda - \log 4)t} = 1 + 4^{-h} \cdot \frac{e^{\lambda}}{4 - e^{\lambda}}$ for $\lambda \in [0, \log 4)$. In particular, if $\lambda \le \log 2$, then $\mathbb{E}[e^{\lambda N_h}] \le 1 + 4^{-h}$.

For the second part, we again consider $\lambda \le \log 2$, and use the independence assumption to write $\mathbb{E}[e^{\lambda N}] = \prod_{h=1}^{h_{\max}} \mathbb{E}[e^{\lambda N_h}] \le \prod_{h=1}^{h_{\max}} \left(1 + 4^{-h}\right) \le e^{\sum_{h=1}^{h_{\max}} 4^{-h}} \le e^{1/3} \le 2$. ◀

We now consider the following decomposition of N_{leaf}:

$$N_{\text{leaf}} = N'_{\text{leaf}} + N''_{\text{leaf}}, \tag{18}$$

where N'_{leaf} counts the reached non-defective leaf nodes having at least one defective ancestor, and N''_{leaf} counts the reached non-defective leaf nodes having all non-defective ancestors. In the case of a single defective item (i.e., $k = 1$), we claim that N'_{leaf} has the same distribution as $2N$, with N given in Lemma 10 for a suitably-chosen value of $h_{\max}$ and $q = \frac{1}{C}$. To see this, we identify the leaf nodes in Lemma 10 with nodes at the *second-last* level of the tree illustrated in Figure 1, and observe that the factor of 2 arises since every such node produces two children at the final level (hence contributing to N'_{leaf}) when placed in a positive test.

In the case of k defective items, some care is needed, as the relevant sets of leaves associated with the k defective paths may overlap, potentially creating complicated dependencies between the associated random variables. However, if we take the definition of N in Lemma 10 and remove some leaves from the count, the quantity $\mathbb{E}[e^{\lambda N}]$ can only decrease further, so the conclusion $\mathbb{E}[e^{\lambda N}] \le 2$ remains valid. Hence, by counting any overlapping leaves only once (in an otherwise arbitrary manner), we can form k independent random variables $N^{(1)}, \dots, N^{(k)}$ satisfying $2\sum_{i=1}^{k} N^{(i)} \overset{\text{d}}{=} N'_{\text{leaf}}$ and $\mathbb{E}[e^{\lambda N^{(i)}}] \le 2$. In addition, since there are k nodes at level $\ell_{\min} = \log_2 k$, we can similarly form $k' \le k$ independent random variables $N^{(k+1)}, \dots, N^{(k+k')}$ satisfying $\sum_{i=k+1}^{k+k'} N^{(i)} \overset{\text{d}}{=} N''_{\text{leaf}}$ and $\mathbb{E}[e^{\lambda N^{(i)}}] \le 2$.

Thus, by (18), N_{leaf} is the sum of at most $2k$ independent sub-exponential random variables. Again applying a standard concentration bound [43, Prop. 5.16], it follows that

$$\mathbb{P}[N_{\text{leaf}} \ge \mathbb{E}[N_{\text{leaf}} \,|\, \mathcal{T}_S] + t \,|\, \mathcal{T}_S] \le e^{-\Omega(\min\{t^2/k, t\})}, \tag{19}$$

from which Lemma 8 follows upon setting $t = \Theta(k)$ and using the fact that $\mathbb{E}[N_{\text{leaf}} \,|\, \mathcal{T}_S] = O(k)$ (see Lemma 5).

2.3.4 Analysis of the final level

Recall that at the final level, we perform $C' \log k$ independent sequences of $2k$ tests, with each item being randomly placed in one test in each sequence. We study the error probability conditioned on the high-probability event that $N_{\text{leaf}} = O(k)$ (see Lemma 8)

For a given non-defective item and a given sequence of $2k$ tests, the probability of colliding with any defective item is at most $\frac{1}{2}$, similarly to Lemma 2. Due to the $C' \log k$ repetitions, for any fixed $c' > 0$, there exists a choice of C' yielding $O(k^{-c'})$ probability of a given non-defective appearing only in positive tests. By a union bound over the $N_{\text{leaf}} = O(k)$ non-defectives at the final level, we find that the estimate $\widehat{S}$ equals S with (conditional) probability $1 - O(k^{1-c'})$.

2.3.5 Number of tests, error probability, decoding time, and storage

The claims of Theorem 1 are established as follows:

- *Number of tests:* As stated in Section 2.1, the number of tests is $t = Ck\log_2\frac{n}{k} + 2C'k\log k$, which behaves as $O\big(k\log\frac{n}{k} + k\log k\big) = O(k\log n)$.
- *Error probability*: The concentration bounds on $N_{\rm leaf}$ and $N_{\rm total}$ (see Lemmas 6 and 8) hold with probability $1 - e^{-\Omega(k)}$ and $1 - e^{-\Omega(k\log\frac{n}{k})}$ respectively, and we treat their complements as error events. These terms are dominated by the final stage, which incurs $O(k^{1-c'})$ failure probability; setting $c' = c+1$ gives the $O(k^{-c})$ behavior stated in Theorem 1.
- *Decoding time*: We claim that conditioned on the the high-probability events $N_{\rm total} = O\big(k\log\frac{n}{k}\big)$ and $N_{\rm leaf} = O(k)$, the decoding time is $O\big(k\log\frac{n}{k} + k\log k\big) = O(k\log n)$. The first term comes from considering all levels except the last, since it takes $O(1)$ time to check whether each node associated with $N_{\rm total}$ is in a positive or negative test. While $N_{\rm total}$ only counts the non-defective nodes, the number of defective nodes also trivially behaves as $O\big(k\log\frac{n}{k}\big)$. At the last level, for each of the $k + N_{\rm leaf} = O(k)$ relevant leaf nodes, we perform $O(\log k)$ such checks for a total time of $O(k\log k)$.
- *Storage*: At the ℓ-th level, we need to store 2^ℓ integers indicating the test associated with each of the 2^ℓ nodes. Hence, excluding the last level, we need to store $\sum_{\ell=\log_2 k}^{\log_2 n - 1} 2^\ell = O(n)$ integers in $\{1,\dots,2k\}$, or $O(n\log k)$ bits. Similarly, the $O(\log k)$ independent sequences of tests at the final level amount to storing $O(n\log k)$ integers, or $O(n(\log k)^2)$ bits. In addition, under the high-probability event $N_{\rm total} = O\big(k\log\frac{n}{k}\big) = O(k\log n)$, the storage of the possibly defective set requires $O(k\log n)$ integers, or $O(k\log^2 n)$ bits. Since $k \le n$, this is no higher than $O(n\log^2 k)$.

3 Storage Reductions via Hashing

A notable weakness of Theorem 1 is that the storage required at the decoder is higher than linear in the number of items. In this section, we present a variant of our algorithm with considerably lower storage that attains similar guarantees to Theorem 1.

The idea is to interpret the mappings $\{1,\dots,2^\ell\} \to \{1,\dots,Ck\}$ at each level as hash functions. Since the high storage in Theorem 1 comes from explicitly storing the corresponding $2^\ell = O(n)$ values, the key to reducing the overall storage is to use lower-storage hash families. The reduced storage comes at the expense of reduced independence between different hash values (e.g., only r-wise independence for some $r \ll n$), and the proof of Theorem 1 utilizes full independence. The modified algorithm in this section uses $\Theta(\log k)$-wise independent hash families, and we leave open the question of whether similar recovery guarantees can be attained with only $O(1)$-wise independence.

A second modification to the algorithm is that in order to attain the behavior $O(k^{-c})$ in the error probability similarly to Theorem 1, we consider the use of $\widetilde{C} \ge 1$ independent repetitions at each level $\ell = \ell_{\min},\dots,\log_2 n - 1$, in the same way that we already used repetitions at the final level. This means that at each level, there are $\widetilde{C}$ sequences of Ck tests (each with a different and independent hash function), and a node is only considered to be possibly defective at a given level if *all* of its associated $\widetilde{C}$ tests are positive.

▶ Remark. This idea of using low-storage hash functions is reminiscent of the Bloom filter data structure [7]. A direct approach to transferring Bloom filters to group testing is to perform $L = O(\log n)$ hashes of each item into one of $t = O(k\log n)$ tests, and then estimate the defective set to be the set of items only included in positive tests [4, Sec. 1.7]. By checking the L test outcomes associated with each item, the defective set can be identified with low storage (depending on the hash family properties) under the preceding scaling laws. However, a drawback of this direct approach is that checking all items separately takes $\Omega(n)$ time. Our algorithm circumvents this via the binary splitting approach.

3.1 Statement of Result

With the above-described modifications to the algorithm, we have the following counterpart to Theorem 1, which formalizes and generalizes Main Result 2.

▶ **Theorem 11** (Algorithmic Guarantees with Reduced Storage)**.** *Let S be a fixed (defective) subset of $\{1, \dots, n\}$ of cardinality k. For any constant $c > 0$, there exist choices of C, $\widetilde{C}$, and C' such that the above-described group testing algorithm adapted from Section 2.1, with a $\Theta(\log k)$-wise independent hash family and $\widetilde{C}$ independent repetitions per level, yields the following with probability at least $1 - O\big(\frac{\log n}{k^c}\big)$:*

- *The returned estimate $\widehat{S}$ equals S;*
- *The algorithm runs in time $O(\mathsf{T}_{\text{hash}} k \log n)$, where T_{hash} is the evaluation time for one hash value;*
- *The algorithm uses $O(k \log n + \mathsf{S}_{\text{hash}} \log n)$ bits of storage, where S_{hash} is the number of bits of storage required for one hash function.*

In addition, the number of tests scales as $O(k \log n)$.

We briefly discuss some explicit values that can be attained for T_{hash} and S_{hash}. Supposing that n, k, and C are powers of two, we can adopt the classical approach of Wegman and Carter [44] and consider a random polynomial over the finite field $\mathrm{GF}(2^m)$, where $m \in \{\log_2 k, \dots, \log_2 n\}$ (depending on the level). In this case, one attains r-wise independence while storing r elements of $\mathrm{GF}(2^m)$ (or $O(r \log n)$ bits), and performing $O(r)$ additions and multiplications in $\mathrm{GF}(2^m)$ to evaluate the hash. As a result, with $r = \Theta(\log k)$, we get

$$\mathsf{T}_{\text{hash}} = O(\log k), \qquad \mathsf{S}_{\text{hash}} = O(\log k \cdot \log n) \tag{20}$$

under the assumption that operations in $\mathrm{GF}(2^m)$ can be performed in constant time. Hence, Theorem 11 gives $O(k \log k \cdot \log n)$ decoding time and $O(k \log n + \log k \cdot \log^2 n)$ storage. Different trade-offs can also be attained using more recent hash families that can attain r-wise independence with an evaluation time significantly less than r [40, 41].

The error probability of $O\big(\frac{\log n}{k^c}\big)$ is slightly worse than the $O(k^{-c})$ scaling of Theorem 1; in particular, $o(1)$ error probability is only guaranteed when $k = (\log n)^{\Omega(1)}$. We expect that this requirement could be avoided via a refined analysis, e.g., by characterizing the behavior of the random variables N_{total} and N_{leaf} used in the proof of Theorem 1. In addition, the following variants of Theorem 11 can already be attained with almost no additional effort:

- We can allow $c = \omega(1)$ in the statement of Theorem 11, but at the expense of the number of tests and decoding time increasing by a multiplicative $\Theta(c)$ factor.
- It is straightforward to show that $\mathbb{E}[N_{\text{total}}] = O\big(k \log \frac{n}{k}\big)$ and $\mathbb{E}[N_{\text{leaf}}] = O(k)$, and one can use Markov's inequality to deduce that $N_{\text{total}} = O\big(k^{1+c} \log \frac{n}{k}\big)$ and $N_{\text{leaf}} = O(k^{1+c})$ with probability $1 - O(k^{-c})$. For any constant $c > 0$, this approach leads to $O(k^{-c})$ error probability with $O(k \log n)$ tests, but the decoding time increases to $O(\mathsf{T}_{\text{hash}} k^{1+c} \log n)$, and the storage increases to $O(k^{1+c} \log^2 n + \mathsf{S}_{\text{hash}} \log n)$.

In the remainder of the section, we provide the proof of Theorem 11.

3.2 Analysis

3.2.1 Auxiliary variance calculation (case $\widetilde{C} = 1$)

We first study the case $\widetilde{C} = 1$ (i.e., no repetitions), as the case $\widetilde{C} > 1$ will then follow easily.

Recall that the algorithm maintains an estimate of the possibly defective (PD) set at each level. We will give conditions under which the size of this set remains at $O(k)$ throughout the course of the algorithm. For $\ell = \ell_{\min} = \log_2 k$, we trivially have at most $k \le 4k$ PD items. We will use an induction argument to show that every level has at most $4k$ PD items, with high probability.

Consider two non-defective nodes indexed by u, v at a given level ℓ having $k' \le k$ defective nodes, let $\mathcal{D}_u, \mathcal{D}_v$ denote the respective events of hashing into a test containing one or more defectives, and let D_u, D_v be the corresponding indicator random variables. The dependence of these quantities on ℓ is left implicit. We condition on all of the test placements performed at the earlier levels, accordingly writing $\mathbb{E}_\ell[\cdot]$ and $\mathrm{Var}_\ell[\cdot]$ for the conditional expectation and conditional variance. In accordance with the above induction idea, we assume that there are at most $4k$ PD nodes at level ℓ.

▶ **Lemma 12** (Mean and Variance Bounds). *Under the preceding setup and definitions, if there are at most $4k$ PD nodes at level ℓ, then we have the following when $\widetilde{C} = 1$ and $C \ge 8$:*

$$\mathbb{E}_\ell\left[\sum_u D_u\right] \le \frac{k}{2} \tag{21}$$

$$\mathrm{Var}_\ell\left[\sum_u D_u\right] = c_{\mathrm{var}} k, \tag{22}$$

where the sums are over all non-defective PD nodes at the ℓ-th level, and $c_{\mathrm{var}} > 0$ is a universal constant.

The proof is given in Appendix B. Given this result, we easily deduce the following.

▶ **Lemma 13.** *For $\widetilde{C} = 1$ and $C \ge 8$, conditioned on the ℓ-th level having at most $4k$ possibly defective (PD) nodes, the same is true at the $(\ell+1)$-th level with probability $1 - O\big(\frac{1}{k}\big)$.*

Proof. Among the PD nodes at the ℓ-th level, at most k are defective, amounting to at most $2k$ children at the next level. By Lemma 12 and Chebyshev's inequality, with probability $1 - O\big(\frac{1}{k}\big)$, at most k non-defective nodes are marked as PD, thus also amount to at most $2k$ additional children at the next level, for a total of $4k$. ◀

3.2.2 Analysis of the error probability

At the first level $\ell = \log_2 k$, we trivially have $k \le 4k$ possibly defective (PD) nodes. For $\widetilde{C} = 1$, using Lemma 13 and an induction argument, the same follows for all levels simultaneously with probability at least $1 - O(k^{-1} \log n)$. For $\widetilde{C} > 1$, we note that since we have $\widetilde{C}$ repetitions at each level and only keep the nodes whose tests are *all* positive, the $1 - O(k^{-1})$ behavior becomes $1 - O(k^{-\widetilde{C}})$ due to the independence of the repetitions. Hence, the expression $1 - O(k^{-1} \log n)$ for $\widetilde{C} = 1$ generalizes to $1 - O\big(k^{-\widetilde{C}} \log n\big)$.

The analysis of the final level in Section 2.3.4 did not rely on $h(\cdot)$ being a fully independent hash function, but rather, only relied on a collision probability of $\frac{1}{2k}$ between any two given items. Since this condition still holds for any pairwise (or higher) independent hash family, we immediately deduce the same conclusion: Conditioned on the final level having $O(k)$ nodes marked as possibly defective, and by choosing C' appropriately in the algorithm description, we attain $O(k^{1-c'})$ error probability at this level for any fixed $c' > 0$.

Combining the above and setting $\widetilde{C} = c$ and $c' = 1 + c$, we attain the desired scaling $O(k^{-c} \log n)$ in the theorem statement.

3.2.3 Number of tests, decoding time, and storage

The remaining claims of Theorem 11 are established as follows:

- *Number of tests:* The number of tests is the same as in the fully independent case, possibly with a modified implied constant if $\widetilde{C} > 1$ and/or a different choice of C is used.
- *Decoding time*: The analysis of the decoding time is similar to the fully independent case (see Section 2.3.5), but each hash takes $\mathsf{T}_{\rm hash}$ time to compute. Hence, the decoding time is $O\big(\mathsf{T}_{\rm hash} k \log n\big)$.
- *Storage*: We use $O(1)$ hash functions at each level except the last, and $O(\log k)$ hash functions at the final level, for a total of $O\big(\log \frac{n}{k} + \log k\big) = O(\log n)$, requiring $O(\mathsf{S}_{\rm hash} \log n)$ storage. In addition, under the high-probability event that there are $O(k)$ possibly defective nodes at each level, their storage requires $O(k)$ integers, or $O(k \log n)$ bits. Hence, the total storage is $O(k \log n + \mathsf{S}_{\rm hash} \log n)$.

4 Conclusion

We have presented a novel non-adaptive group testing algorithm ensuring high-probability (for-each) recovery with $O(k \log n)$ scaling in both the number of tests and decoding time. In addition, we presented a low-storage variant with similar guarantees depending on the hash family used. An immediate open question for this variant is whether similar guarantees hold for $O(1)$-wise independent hash families. In addition, even for the fully independent version, it would be of significant interest to develop a variant that is robust to random noise in the test outcomes (see Footnote 2 on Page 4).

References

1 Matthew Aldridge. The capacity of Bernoulli nonadaptive group testing. *IEEE Transactions on Information Theory*, 63(11):7142–7148, 2017.

2 Matthew Aldridge. Individual testing is optimal for nonadaptive group testing in the linear regime. *IEEE Transactions on Information Theory*, 65(4):2058–2061, April 2019.

3 Matthew Aldridge, Leonardo Baldassini, and Oliver Johnson. Group testing algorithms: Bounds and simulations. *IEEE Transactions on Information Theory*, 60(6):3671–3687, June 2014.

4 Matthew Aldridge, Oliver Johnson, and Jonathan Scarlett. Group testing: An information theory perspective. *Foundations and Trends in Communications and Information Theory*, 15(3–4):196–392, 2019.

5 George K. Atia and Venkatesh Saligrama. Boolean compressed sensing and noisy group testing. *IEEE Transactions on Information Theory*, 58(3):1880–1901, March 2012.

6 Leonardo Baldassini, Oliver Johnson, and Matthew Aldridge. The capacity of adaptive group testing. In *IEEE International Symposium on Information Theory*, pages 2676–2680, July 2013.

7 Burton H. Bloom. Space/time trade-offs in hash coding with allowable errors. *Communications of the ACM*, 13(7):422–426, 1970.

8 Steffen Bondorf, Binbin Chen, Jonathan Scarlett, Haifeng Yu, and Yuda Zhao. Cross-sender bit-mixing coding. In *International Conference on Information Processing in Sensor Networks*, 2019.

9 Steffen Bondorf, Binbin Chen, Jonathan Scarlett, Haifeng Yu, and Yuda Zhao. Sublinear-time non-adaptive group testing with $O(k \log n)$ tests via bit-mixing coding, 2019. arXiv:1904.10102.

10 Sheng Cai, Mohammad Jahangoshahi, Mayank Bakshi, and Sidharth Jaggi. Efficient algorithms for noisy group testing. *IEEE Transactions on Information Theory*, 63(4):2113–2136, 2017.

11 Chun Lam Chan, Sidharth Jaggi, Venkatesh Saligrama, and Samar Agnihotri. Non-adaptive group testing: Explicit bounds and novel algorithms. *IEEE Transactions on Information Theory*, 60(5):3019–3035, May 2014.

12 Mahdi Cheraghchi. Noise-resilient group testing: Limitations and constructions. In *International Symposium on Fundamentals of Computation Theory*, pages 62–73, 2009.

13 Mahdi Cheraghchi and Vasileios Nakos. Combinatorial group testing and sparse recovery schemes with near-optimal decoding time, 2020. arXiv:2006.08420.

14 Raphaël Clifford, Klim Efremenko, Ely Porat, and Amir Rothschild. Pattern matching with don't cares and few errors. *Journal of Computer and System Sciences*, 76(2):115–124, 2010.

15 Amin Coja-Oghlan, Oliver Gebhard, Max Hahn-Klimroth, and Philipp Loick. Information-theoretic and algorithmic thresholds for group testing. In *International Colloquium on Automata, Languages and Programming*, 2019.

16 Amin Coja-Oghlan, Oliver Gebhard, Max Hahn-Klimroth, and Philipp Loick. Optimal group testing, 2020. arXiv:1911.02287.

17 Graham Cormode and S. Muthukrishnan. An improved data stream summary: the count-min sketch and its applications. *Journal of Algorithms*, 55(1):58–75, 2005.

18 Graham Cormode and S. Muthukrishnan. What's hot and what's not: Tracking most frequent items dynamically. *ACM Transactions on Database Systems*, 30(1):249–278, March 2005.

19 Robert Dorfman. The detection of defective members of large populations. *Annals of Mathematical Statistics*, 14(4):436–440, 1943.

20 Dingzhu Du and Frank K. Hwang. *Combinatorial group testing and its applications*, volume 12. World Scientific, 2000.

21 Meyer Dwass. The total progeny in a branching process and a related random walk. *Journal of Applied Probability*, 6(3):682–686, 1969.

22 Arkadii G. D'yachkov and Vladimir V. Rykov. A survey of superimposed code theory. *Problems of Control and Information*, 12(4):1–13, 1983.

23 Arkady D'yachkov, Ilya Vorobyev, Nikita Polianskii, and Vladislav Shchukin. Bounds on the rate of superimposed codes. In *IEEE International Symposium on Information Theory*, June 2014.

24 William Feller. *An introduction to probability theory and its applications*. Wiley, 1957.

25 Antonio Fernández Anta, Miguel A. Mosteiro, and Jorge Ramón Muñoz. Unbounded contention resolution in multiple-access channels. In *Distributed Computing*, volume 6950, pages 225–236. Springer Berlin Heidelberg, 2011.

26 Anna C. Gilbert, Mark A. Iwen, and Martin J. Strauss. Group testing and sparse signal recovery. In *Asilomar Conference on Signals, Systems, and Computers*, pages 1059–1063, October 2008.

27 Anna C. Gilbert, Martin J. Strauss, Joel A. Tropp, and Roman Vershynin. One sketch for all: Fast algorithms for compressed sensing. In *ACM-SIAM Symposium on Discrete Algorithms*, pages 237–246, New York, 2007.

28 Catherine A. Hogan, Malaya K. Sahoo, and Benjamin A. Pinsky. Sample pooling as a strategy to detect community transmission of SARS-CoV-2. *Journal of the American Medical Association*, April 2020.

29 Frank K. Hwang. A method for detecting all defective members in a population by group testing. *Journal of the American Statistical Association*, 67(339):605–608, 1972.

30 Huseyin A. Inan, Peter Kairouz, Mary Wootters, and Ayfer Ozgur. On the optimality of the Kautz-Singleton construction in probabilistic group testing. *IEEE Transactions on Information Theory*, 65(9):5592–5603, September 2019.

31 Piotr Indyk, Hung Q. Ngo, and Atri Rudra. Efficiently decodable non-adaptive group testing. In *ACM-SIAM Symposium on Discrete Algorithms*, 2010.

32 Piotr Indyk and Eric Price. K-median clustering, model-based compressive sensing, and sparse recovery for earth mover distance. In *ACM Symposium on Theory of Computing*, pages 627–636, 2011.

33 Oliver Johnson, Matthew Aldridge, and Jonathan Scarlett. Performance of group testing algorithms with near-constant tests-per-item. *IEEE Transactions on Information Theory*, 65(2):707–723, February 2019.

34 Kangwook Lee, Ramtin Pedarsani, and Kannan Ramchandran. SAFFRON: A fast, efficient, and robust framework for group testing based on sparse-graph codes. *IEEE Transactions on Signal Processing*, 67(17):4649–4664, September 2019.

35 M. B. Malyutov and P. S. Mateev. Screening designs for non-symmetric response function. *Mathematical Notes of the Academy of Sciences of the USSR*, 29:109–127, 1980.

36 Mikhail B. Malyutov. The separating property of random matrices. *Mathematical Notes of the Academy of Sciences of the USSR*, 23(1):84–91, 1978.

37 Hung Q. Ngo, Ely Porat, and Atri Rudra. Efficiently decodable error-correcting list disjunct matrices and applications. In *International Colloquium on Automata, Languages and Programming*, 2011.

38 Ely Porat and Amir Rothschild. Explicit nonadaptive combinatorial group testing schemes. *IEEE Transactions on Information Theory*, 57(12):7982–7989, 2011.

39 Jonathan Scarlett and Volkan Cevher. Phase transitions in group testing. In *ACM-SIAM Symposium on Discrete Algorithms*, 2016.

40 Alan Siegel. On universal classes of fast high performance hash functions, their time-space tradeoff, and their applications. In *IEEE Symposium on Foundations of Computer Science*, 1989.

41 Mikkel Thorup. Fast and powerful hashing using tabulation. *Communications of the ACM*, 60(7):94–101, 2017.

42 Claudio M Verdun, Tim Fuchs, Pavol Harar, Dennis Elbrächter, David S Fischer, Julius Berner, Philipp Grohs, Fabian J Theis, and Felix Krahmer. Group testing for SARS-CoV-2 allows for up to 10-fold efficiency increase across realistic scenarios and testing strategies. URL: `https://www.medrxiv.org/content/10.1101/2020.04.30.20085290v2`, 2020.

43 Roman Vershynin. Introduction to the non-asymptotic analysis of random matrices, 2010. `arXiv:1011.3027`.

44 Mark N. Wegman and J. Lawrence Carter. New hash functions and their use in authentication and set equality. *Journal of Computer and System Sciences*, 22(3):265–279, 1981.

45 Idan Yelin, Noga Aharony, Einat Shaer-Tamar, Amir Argoetti, Esther Messer, Dina Berenbaum, Einat Shafran, Areen Kuzli, Nagam Gandali, Tamar Hashimshony, Yael Mandel-Gutfreund, Michael Halberthal, Yuval Geffen, Moran Szwarcwort-Cohen, and Roy Kishony. Evaluation of COVID-19 RT-qPCR test in multi-sample pools. URL: `https://www.medrxiv.org/content/early/2020/03/27/2020.03.26.20039438`, 2020.

A Discussion on the SAFFRON Algorithm

While the SAFFRON algorithm [34] is based on sparse-graph codes, we find it most instructive to compare against the simplified singleton-only version [4, Sec. 5.4], as the more sophisticated version does not attain better scaling laws (though it may attain better constant factors).

Singleton-only SAFFRON is briefly outlined as follows. One forms $O(k \log k)$ "bundles" of tests of size $2\log_2 n$ each, and assigns each item to any given bundle with probability $O\big(\frac{1}{k}\big)$. If a given item is assigned to a given bundle, then its $(\log_2 n)$-bit description is encoded into the first $\log_2 n$ tests (i.e., the item is included in the test if and only if its bit description contains a 1 at the corresponding location), and its bit-wise complement is encoded into the last $\log_2 n$ tests. The following decoding procedure ensures the identification of any defective item for which there exists a bundle in which it is included without any other defective items:

- For each bundle, check whether the first $\log_2 n$ test outcomes equal the bit-wise negation of the second $\log_2 n$ outcomes.
- If so, add the item with bit representation given by the first $\log_2 n$ outcomes into the defective set estimate.

Due to the first step, the second step will never erroneously be performed on a bundle without defective items, nor on a bundle with multiple defective items.

In [34], a decoding time of $O(k \log k \cdot \log n)$ is stated under the assumption that reading the $2 \log_2 n$ bits takes $O(\log n)$ time. However, if the associated $2 \log_2 n$ tests are stored in memory as two "words" of length $\log_2 n$ each, then a word-RAM model of computation only incurs $O(1)$ time per bundle, or $O(k \log k)$ time overall.

We also note that SAFFRON only requires $O(k \log k \cdot \log n)$ bits of storage, amounting to negligible storage overhead beyond the test outcomes themselves. This is because the only item indices stored are those added to the estimate of the defective set, and there are at most k such indices (or $k \log_2 n$ bits) due to the fact that SAFFRON makes no false positives.

B Proof of Lemma 12 (Mean and Variance Bounds)

For ease of notation, we leave the subscripts $(\cdot)_\ell$ implicit throughout the proof, but the associated conditioning is understood to apply to all probabilities, expectations, variance terms, and so on.

We first prove (21). The event $\mathcal{D}_u$ occurs if u is hashed into the same bin as any of the $k' \le k$ defective nodes. Since we are hashing into $\{1, \dots, Ck\}$ and the hash family is (at least) pairwise independent, each collision occurs with probability $\frac{1}{Ck}$. Hence, by the union bound, u is in a positive test with probability at most $\frac{1}{C}$, and (21) follows from the assumption that there are at most $4k$ PD nodes and $C \ge 8$.

As for (22), we first characterize $\mathrm{Cov}[D_u, D_v]$, writing

$$\mathrm{Cov}[D_u, D_v] = \mathbb{E}[D_u D_v] - \mathbb{E}[D_u]\mathbb{E}[D_v] \tag{23}$$

$$= \mathbb{P}[\mathcal{D}_u \cap \mathcal{D}_v] - \mathbb{P}[\mathcal{D}_u]\mathbb{P}[\mathcal{D}_v] \tag{24}$$

$$= \mathbb{P}[\mathcal{D}_u] + \mathbb{P}[\mathcal{D}_v] - \mathbb{P}[\mathcal{D}_u \cup \mathcal{D}_v] - \mathbb{P}[\mathcal{D}_u]\mathbb{P}[\mathcal{D}_v]. \tag{25}$$

We proceed by bounding $\mathbb{P}[\mathcal{D}_u]$ (the same bound holds for $\mathbb{P}[\mathcal{D}_v]$) and $\mathbb{P}[\mathcal{D}_u \cup \mathcal{D}_v]$ separately.

Probability of the individual event

Fix a non-defective node u. Let $h(\cdot)$ denote the random hash function with output values in $\{1, \dots, Ck\}$, and for each defective node indexed by $i \in \{1, \dots, k'\}$ (where k' is the total number of defective nodes at the level under consideration), let B_i be the "bad" event that $h(i) = h(u)$. We apply the inclusion-exclusion principle, which is written in terms of the following quantities for $j = 1, \dots, k'$:

$$T_j = \sum_{1 < i_1 < \dots < i_j < k'} \mathbb{P}[B_{i_1} \cap \dots \cap B_{i_j}]. \tag{26}$$

If the hash function is $(j+1)$-wise independent, this simplifies to

$$T_j = \binom{k'}{j} \left(\frac{1}{Ck}\right)^j. \tag{27}$$

Hence, if the hash function is $(j_{\max}+1)$-wise independent for some $j_{\max}$, then the inclusion-exclusion principle gives

$$\mathbb{P}[\mathcal{D}_u] = \mathbb{P}\bigg[\bigcup_{i=1,\dots,k'} B_i\bigg] \tag{28}$$

$$\le \sum_{j=1}^{j_{\max}} (-1)^{j+1} T_j \tag{29}$$

$$= \sum_{j=1}^{j_{\max}} \binom{k'}{j} (-1)^{j+1} \Big(\frac{1}{Ck}\Big)^j \tag{30}$$

for odd-valued $j_{\max}$, and the reverse inequality for even-valued $j_{\max}$. Using the fact that $\sum_{j=1}^{k'} \binom{k'}{j}(-1)^{j+1}\big(\frac{1}{Ck}\big)^j = 1 - \big(1-\frac{1}{Ck}\big)^{k'}$, we can write (30) as

$$\mathbb{P}[\mathcal{D}_u] \le 1 - \Big(1 - \frac{1}{Ck}\Big)^{k'} - \sum_{j_{\max}+1}^{k'} \binom{k'}{j} (-1)^{j+1} \Big(\frac{1}{Ck}\Big)^j. \tag{31}$$

The final term can then be bounded as follows in absolute value:

$$\bigg| \sum_{j_{\max}+1}^{k'} \binom{k'}{j} (-1)^{j+1} \Big(\frac{1}{Ck}\Big)^j \bigg| \le \sum_{j_{\max}+1}^{k'} \binom{k'}{j} \Big(\frac{1}{Ck}\Big)^j \tag{32}$$

$$\le \sum_{j_{\max}+1}^{\infty} \Big(\frac{1}{C}\Big)^j \tag{33}$$

$$= \frac{(1/C)^{j_{\max}+1}}{1-1/C}, \tag{34}$$

where (33) uses $\binom{k'}{j} \le (k')^j$ and $k' \le k$, and (34) applies the geometric series formula. Assuming $C \ge 2$, we can further upper bound the above expression by $(1/C)^{j_{\max}}$, and hence by any target value δ_0 provided that $j_{\max} \ge \log_C \frac{1}{\delta_0}$. Recall also that (31) is reversed for even-valued $j_{\max}$, so loosening the preceding requirement to $j_{\max} \ge \lceil \log_C \frac{1}{\delta_0} \rceil + 1$ gives

$$1 - \Big(1 - \frac{1}{Ck}\Big)^{k'} - \delta_0 \le \mathbb{P}[\mathcal{D}_u] \le 1 - \Big(1 - \frac{1}{Ck}\Big)^{k'} + \delta_0. \tag{35}$$

Since u is arbitrary, the same bound also holds for $\mathbb{P}[\mathcal{D}_v]$.

Probability of the union of two events

We can decompose $\mathbb{P}[\mathcal{D}_u \cup \mathcal{D}_v]$ as follows:

$$\mathbb{P}[\mathcal{D}_u \cup \mathcal{D}_v] = \Big(1 - \frac{1}{Ck}\Big)\mathbb{P}[\mathcal{D}_u \cup \mathcal{D}_v \,|\, h(u) \ne h(v)] + \frac{1}{Ck}\mathbb{P}[\mathcal{D}_u \cup \mathcal{D}_v \,|\, h(u) = h(v)] \tag{36}$$

$$= \mathbb{P}[\mathcal{D}_u \cup \mathcal{D}_v \,|\, h(u) \ne h(v)] + O\Big(\frac{1}{k}\Big). \tag{37}$$

Hence, we focus on the case $h(u) \ne h(v)$ in the following. Similar to the above, let B'_i be the "bad" event that $h(i) \in \{h(u), h(v)\}$, and define

$$T'_j = \sum_{1 < i_1 < \ldots < i_j < k'} \mathbb{P}[B'_{i_1} \cap \ldots \cap B'_{i_j} \,|\, h(u) \ne h(v)]. \tag{38}$$

If the hash function is $(j+2)$-wise independent, this simplifies to

$$T'_j = \binom{k'}{j}\Big(\frac{2}{Ck}\Big)^j, \tag{39}$$

where the factor of two comes from the possibility of colliding with either u or v. Following the same argument as above, we find that if (i) $j_{\max} \geq \lceil \log_{C/2} \frac{1}{\delta_0} \rceil + 1$, (ii) $C \geq 4$, and (iii) the hash function is $(j_{\max}+2)$-wise independent, then the following analog of (35) holds:

$$1 - \Big(1 - \frac{2}{Ck}\Big)^{k'} - \delta_0 \leq \mathbb{P}[\mathcal{D}_u \cup \mathcal{D}_v \,|\, h(u) \neq h(v)] \leq 1 - \Big(1 - \frac{2}{Ck}\Big)^{k'} + \delta_0. \tag{40}$$

Combining and simplifying

Setting $\delta_0 = \frac{1}{k}$ and combining the above findings, we deduce that for a $\Theta(\log k)$-wise independent hash,

$$\mathbb{P}[\mathcal{D}_u] = 1 - \Big(1 - \frac{1}{Ck}\Big)^{k'} + O\Big(\frac{1}{k}\Big), \tag{41}$$

$$\mathbb{P}[\mathcal{D}_u \cup \mathcal{D}_v] = 1 - \Big(1 - \frac{2}{Ck}\Big)^{k'} + O\Big(\frac{1}{k}\Big). \tag{42}$$

The idea in the following is to approximate $1 - \frac{\nu}{Ck} \approx e^{-\frac{1}{Ck}}$ for $\nu = 1, 2$, and substitute into (25). To make this more precise, we use the fact that $k' \leq k$ to write

$$\Big(1 - \frac{1}{Ck}\Big)^{k'} = \Big(e^{-\frac{1}{Ck} + O\left(\frac{1}{k^2}\right)}\Big)^{k'} \tag{43}$$

$$= e^{-\frac{k'}{Ck} + O\left(\frac{1}{k}\right)} \tag{44}$$

$$= e^{-\frac{k'}{Ck}} + O\Big(\frac{1}{k}\Big). \tag{45}$$

Applying a similar argument to $\big(1 - \frac{2}{Ck}\big)^{k'}$ and substituting into (25), we obtain

$$\mathrm{Cov}[D_u, D_v] = 2\Big(1 - e^{-\frac{k'}{Ck}}\Big) - \Big(1 - e^{-\frac{2k'}{Ck}}\Big) - \Big(1 - e^{-\frac{k'}{Ck}}\Big)^2 + O\Big(\frac{1}{k}\Big) \tag{46}$$

$$= O\Big(\frac{1}{k}\Big), \tag{47}$$

since the first three terms cancel upon expanding the square. The proof is concluded by writing

$$\mathrm{Var}\Big[\sum_u D_u\Big] = \sum_u \mathrm{Var}[D_u] + \sum_{u \neq v} \mathrm{Cov}[D_u, D_v] = O(k) \tag{48}$$

since $\mathrm{Var}[D_u] \leq \mathbb{E}[D_u] \leq \frac{1}{C}$, and there are at most $4k$ values of u by assumption.