Average probe complexity in quorum systems

Yehuda Hassin, David Peleg

Department of Computer Science and Applied Mathematics, The Weizmann Institute of Science, Rehovot 76100, Israel

Received 12 January 2003; received in revised form 5 September 2005
Available online 15 December 2005

Abstract

This paper discusses the probe complexity of randomized algorithms and the deterministic average case probe complexity for some classes of nondominated coteries, including majority, crumbling walls, tree, wheel and hierarchical quorum systems, and presents upper and lower bounds for the probe complexity of quorum systems in these classes.

© 2005 Elsevier Inc. All rights reserved.

Keywords: Quorum systems; Nondominated coteries; Probe complexity

1. Introduction

1.1. Background and motivation

Many tasks in distributed computing make use of quorum systems. A quorum system for a set $U$ of elements is a collection $S$ of pairwise intersecting subsets of $U$. Quorums were used for the problems of mutual exclusion, data replication protocols and distributed access control and signatures [3,18,19]. Specifically, quorum systems were used for increasing the availability and efficiency of replicated data [8], and as a mechanism for granting permission to perform certain critical tasks in an exclusive manner [1,10].

* Corresponding author.

E-mail addresses: hassin@mail.jce.ac.il (Y. Hassin), david.peleg@weizmann.ac.il (D. Peleg).

1 Supported in part by a grant from the Israel Ministry of Science and Art.

0022-0000/$ – see front matter © 2005 Elsevier Inc. All rights reserved.
doi:10.1016/j.jcss.2005.10.003
In all the known applications for quorum systems, a quorum is employed by accessing each of its elements. The current paper considers situations in which the elements (or processors) of the system may occasionally fail and stop functioning. In this setting, the user cannot just pick an arbitrary quorum, since one (or more) of its elements may have failed. Rather, the user must pick a live quorum, i.e., a quorum all of whose elements are currently nonfaulty. If no live quorum exists then the task cannot be performed. Obviously, at any given moment the system can be in one of two states: either it currently contains a live quorum or it does not. The user may learn the state of the system by probing the elements one by one until finding a witness, in the form of either a live quorum or an evidence that no live quorum exists.

Our current work aims at studying algorithms that minimize the number of probed elements before a witness is found. The number of probes necessary in the worst case is termed the probe complexity of the quorum system. Deterministic algorithms for the worst case model were examined in [15]. Many of the known constructions of quorum systems were shown to be evasive, i.e., of probe complexity \( n = |U| \). In contrast, in the current paper it is shown that randomized algorithms may enjoy improved probe complexity in the worst case model compared to that achieved by deterministic algorithms. In addition, we explore the probe complexity of quorum systems in a probabilistic model where each processor may fail independently with probability \( p \). This may be a natural assumption to use when modeling distributed systems, as in such systems the individual processors are typically separate and independently functioning.

1.2. Related work

Previous work on quorum systems can be divided into three categories. First, as mentioned earlier, quorum systems were used as a tool for various applications in distributed computing. Second, various constructions of quorum systems were suggested, ranging from simple systems that use voting to define quorums to more complex ones based on elaborate combinatorial structures. Some of the systems analyzed in this paper were introduced in [1,2,8–10,14].

A third line of research concerns the properties of quorum systems. A number of measures were suggested in order to evaluate the efficiency of a quorum system. In particular, the load parameter of quorum systems measures how work is divided in the worst case between processors [6,12]. The availability parameter estimates the probability that a live quorum exists in a probabilistic model [13]. Finally, the probe complexity was first introduced and analyzed, for the deterministic worst case model, in [15]. In particular, a universal algorithm of probe complexity \( O(c^2) \) is shown for the class of \( c \)-uniform quorum systems (i.e., systems in which the size of each subset is \( c \)). No theoretical results were known for the average probe complexity (in a probabilistic model). In [4,11], various heuristics were suggested and tested for some specific types of quorum systems.

1.3. Our results

We first study probe complexity in a probabilistic model, where each node may fail with some fixed probability \( p \). Among other results, we show that in the probabilistic
model the probe complexity of some quorum systems is significantly better than their probe complexity in the worst case model.

For example, consider the family \((n_1, \ldots, n_k)\)-CW of crumbling walls quorum systems on \(k\) rows due to [14]. In the worst case model, it was proved in [15] that all elements must be probed. In contrast, we prove that in the probabilistic model only \(O(k)\) nodes are probed on the average, where generally \(k\) needs not depend on \(n\), the number of elements in the universe. This gap may be interpreted as saying that for crumbling wall systems, the worst-case scenario occurs only with very low probability.

Another example examined reveals a polynomial gap between different \(p\) values. Specifically, for the Tree system of [1], we present an algorithm with average probe complexity \(O(n \log_2(1 + p))\).

Our final example is the hierarchical quorum system (HQS) of [8]. HQS belongs to the class of nondominated (ND) quorum systems, defined formally in the next section, and its quorums are of uniform size. The lower bound we establish for HQS in Section 3.4 proves that not all \(c\)-uniform ND coteries have probabilistic probe complexity of \(O(c)\), and also demonstrates the fact that the probabilistic probe complexity of an ND coterie can be asymptotically larger than its average quorum size.

We then reexamine the worst case model, focusing on randomized algorithms. In general, randomized algorithms yield better probe complexity, although for most of the examples considered here, only a constant factor improvement is achieved. We use the theorem of [20] to establish lower bounds for randomized algorithms. For example, for the Tree system mentioned above, we show an algorithm requiring at most \(5n/6\) expected probes and a lower bound of \(2n/3\) on the number of expected probes.

Our main results are summarized in Table 1.

### Table 1

The probe complexity of various nondominated coteries in the worst case model with randomized algorithms and in the probabilistic model with \(p = \frac{1}{2}\).

<table>
<thead>
<tr>
<th>Quorum systems</th>
<th>Maj</th>
<th>Triang</th>
<th>Tree</th>
<th>HQS</th>
</tr>
</thead>
<tbody>
<tr>
<td>Probabilistic model</td>
<td>Lower bound</td>
<td>(n - \theta(\sqrt{n}))</td>
<td>(2k - \theta(\sqrt{k}))</td>
<td>-</td>
</tr>
<tr>
<td></td>
<td>Upper bound</td>
<td>(n - \theta(\sqrt{n}))</td>
<td>(2k - 1)</td>
<td>(O(n^{0.585}))</td>
</tr>
<tr>
<td>Randomized model</td>
<td>Lower bound</td>
<td>(n - 1 + o(1))</td>
<td>(\frac{n+1}{2})</td>
<td>(2n/3)</td>
</tr>
<tr>
<td></td>
<td>Upper bound</td>
<td>(n - 1 + o(1))</td>
<td>(\frac{n+1}{2} + \log k)</td>
<td>(5n/6)</td>
</tr>
</tbody>
</table>

2. Definitions and preliminaries

2.1. Quorum systems

Let \(U = \{1, \ldots, n\}\) be a finite universe of \(n\) elements. A set system \(S = \{S_1, \ldots, S_m\}\) is a collection of subsets of \(U\). A quorum system is a set system satisfying the intersection property, i.e., s.t. \(S_i \cap S_j \neq \emptyset\) for every \(1 \leq i < j \leq m\). A coterie is a quorum system \(S\) with the minimality property, i.e., satisfying \(S \not\subset R\) for every two quorums \(S, R \in S\).
Let $R, S$ be coteries over the same universe. We say that $R$ dominates $S$, denoted $R \succ S$, if $R \neq S$ and for every $S \in S$ there is some $R \in R$ such that $R \subseteq S$. A coterie $S$ is nondominated (ND) if there is no coterie $R$ such that $R \succ S$.

A set $R$ is a transversal of a set system $S$ if $R \cap S \neq \emptyset$ for every $S \in S$.

**Lemma 2.1.** [3] Let $S$ be an ND coterie and let $R$ be a transversal of $S$. Then there exists a quorum $S \in S$ such that $S \subseteq R$.

Alternatively, a quorum system can be viewed as a monotone boolean function.

**Definition 1.** Let $S$ be a quorum system. Let $x_1, \ldots, x_n$ be boolean variables corresponding to the elements of the universe. Then the characteristic boolean function of $S$ is $f_S : \{0, 1\}^n \rightarrow \{0, 1\}$ defined by

$$f_S(x_1, \ldots, x_n) = \bigvee_{s \in S} \bigwedge_{i \in s} x_i.$$

Clearly $f_S$ is monotone, and $f_S = 1$ iff all the variables corresponding to some quorum have the value 1. A minterm for a monotone boolean function is an assignment to the variables with the property that (1) the function is evaluated to 1 but (2) if we change one variable from 1 to 0 then the root evaluation changes to 0. From our definition it can be seen that each assignment of 1’s to the variables of some quorum $Q$ and 0’s to $U \setminus Q$ has the properties of a minterm. Hence the minterms of $f_S$ are the quorums. The properties of such functions are discussed extensively in [7].

### 2.2. Examples of ND coterie families

Let us now describe some of the known examples of ND coterie families analyzed later on. All of these systems are defined over the universe $U = \{1, \ldots, n\}$.

The majority system $\text{Maj}$ is defined in [18]. It consists of the collection of all sets of $(n + 1)/2$ elements, where $n$ is odd.

The $\text{Wheel}$ system, defined in [6], consists of the sets $\{1, i\}$ for $i \geq 2$ and the set $\{2, \ldots, n\}$.

The $\text{crumbling walls}$ (CW) family is defined in [14]. A quorum system of this family, denoted $(n_1, \ldots, n_k)$-CW, is constructed as follows. The elements of $U$ are logically arranged in a two-dimensional structure composed of $k$ rows of varying widths, where row $i$ has width $n_i$, such that $\sum n_i = n$. A quorum consists of one full row $j$ and one representative element from every row occurring below row $j$. It is known that if the width of the first row is 1 and the other rows are of width greater than 1 then the quorum system is an ND coterie. The Wheel system mentioned above can be represented as a special case of a $(1, n - 1)$-CW system. Another known subfamily of CW is the $\text{Triang}$ system [2,9], which is a $(1, 2, \ldots, d)$-CW, namely, where row $i$ is of width $i$. An example of the Triang system is shown in Fig. 1.

The $\text{Tree}$ system is defined in [1]. The elements of the universe $U$ are arranged in a complete binary tree. A quorum in the system is defined recursively as follows. A quorum
is either the union of the root with some quorum defined over one of its subtrees, or the union of two quorums, one defined over each of the two subtrees of the tree. An example of a quorum in the Tree system is depicted in Fig. 2.

The Hierarchical quorum system (HQS) is suggested in [8]. The elements of this system are organized as the leaves of a complete ternary computation tree. The internal nodes act as 2-of-3 majority gates. Each element is assigned 0 or 1. The quorums for this system are determined from the minterms of the tree where the computational tree represents a boolean function and every minterm defines a quorum by taking the union of elements that were assigned the value 1 (see the explanation above in Definition 1). An example is depicted in Fig. 3.

2.3. Probe complexity

Consider a setting where the elements of a quorum system represent processors, which may occasionally fail. We adopt the following terminology. Each element is colored with either red (indicating that the processor has failed) or green (indicating a live processor).
We say that a set of elements is red (respectively green) if all of its elements are. The task we deal with is to find a witness to the state of the system, i.e., either a green quorum, or if there is no such quorum, an evidence to this fact, in the form of a red transversal. By Lemma 2.1, for an ND coterie, such a red transversal must contain a red quorum. Hence, it suffices to look for a monochromatic quorum.

We are interested in algorithms that minimize the number of probed elements before a witness is found. The input to our algorithms is some coloring of the elements. Probing an element \( i \) reveals its color, denoted \( c(i) \). The algorithms are allowed to be adaptive, i.e., the next element to be probed can be selected according to the outcome of the previous probes. Such algorithms were described in [15] by means of a binary rooted tree, referred to as a probe strategy tree. Every node of the tree is labeled with an element, and the two outgoing edges are marked by the two possible outcomes of the probe, namely, green or red. Every leaf node is also marked by green or red, and represents the output of the execution of the algorithm, namely finding a green or red quorum.

A worst case model for studying probe complexity was considered in [15]. In this model, the probe complexity of an algorithm \( T \) for an ND coterie \( S \) is defined as

\[
P_C(T, S) = \text{Depth}(T),
\]

where \( \text{Depth}(T) \) is the depth of the tree \( T \), defined as the number of nodes in the longest path from the root to a leaf in \( T \). The probe complexity of an ND coterie \( S \) is

\[
P_C(S) = \min_T \{P_C(T, S)\},
\]

where the minimum is taken over all possible probe strategy trees \( T \). This also equals the number of probes performed by the best deterministic algorithm over the worst case input. Alternatively, one can also think about this tree as representing a game between two players. Starting from the root, the adversary chooses the right or the left edge in order to maximize the length of the path taken, and the player chooses the labels of the nodes in order to minimize it.

**Lemma 2.2.** [15] The ND coterie systems Maj, Wheel, CW and Tree have probe complexity \( n \).

In the first part of this paper we study probe strategies in the probabilistic model. In this model, each element is colored red with probability \( p \) and green with probability \( 1 - p \). The availability of a quorum system \( S \), denoted \( F_p(S) \), is the probability that the system does not contain a green quorum. We use the following facts, proven in [13].

**Fact 2.3.** [13]

1. For any \( p \leq \frac{1}{2} \), the availability of any ND coterie is bounded from above by \( p \).
2. For any \( S \in \text{NDC} \), \( F_p(S) + F_{1-p}(S) = 1 \).

The probe complexity of an ND coterie \( S \) in the probabilistic model is defined as \( P_C_p(S) \), where again the minimum is taken over all probe strategy trees \( T \), except this time the labels on the edges of the rooted binary trees are \( p \) and \( 1 - p \), and the probe complexity is the expected depth of \( T \), defined as follows. Let \( L(T) \) denote the set of
Fig. 4. An example for a decision tree to Maj3. Each internal node represents a probes and the bit on each edge represents the outcome of the probe. Each leaf is marked by either “+” or “−” representing respectively the fact that the probing algorithm has obtained a live quorum or has confirmed that none exists.

leaves of $T$, and for a leaf $l$ let $\mathbb{P}(l)$ be the probability to reach $l$, i.e., the product of the probabilities marking the path from the root to $l$. Then for any $T$ this expected value is $\sum_{l \in L(T)} \text{Depth}(l) \cdot \mathbb{P}(l)$.

In the second part of this paper we deal with the worst case model, but concentrate on randomized algorithms. We give algorithms for a number of ND coterie systems. Moreover, we use the method of [20] in order to establish lower bounds for randomized algorithms. The probe complexity for randomized algorithms is defined as for the deterministic case, except not for one tree but for some distribution $\mu$ on a collection of trees, which computes the state of the system. Formally, we define the randomized probe complexity of a given tree distribution $\mu$ to be

$$PC_R(\mu, S) = \max_c \{\mathbb{E}_{\mu}(\text{Depth}(T_c))\},$$

for $T_c$ being the length of the branch of $T$ corresponding to an input $c$. The randomized probe complexity for $S$ is defined, as in the deterministic case, for the optimal randomized algorithm (or tree distribution), namely,

$$PC_R(S) = \min_{\mu} \{PC_R(\mu, S)\}.$$

We next present a simple example for calculating the probe complexity in the various models defined formally above. The quorum system we choose is Maj3 where $U = \{1, 2, 3\}$ and $S = \{\{1, 2\}, \{2, 3\}, \{1, 3\}\}$, namely, all the subsets of size 2. The probe strategy for Maj3 is described formally in the decision tree depicted in Fig. 4.

It can easily be seen that the longest path is of length 3 and that the average path length is $2.5$. As for a randomized strategy we really have 6 possible decision trees. Each such algorithm first chooses at random a permutation on $\{x_1, x_2, x_3\}$ and then probes the elements in that order. Averaging the length of the longest path, namely, the number of probes for some input, on these 6 trees we get $\frac{20}{3}$. The results are summarized as follows.

1. $PC(\text{Maj3}) = \max \{\text{Depth}(T_c)\} = 3$;
2. $PC_R(\text{Maj3}) = \max \{\mathbb{E}_\mu(\text{Depth}(T_c))\} = \frac{20}{3}$;
3. $PPC(\text{Maj3}) = \mathbb{E}_\mu(\text{Depth}(T_c)) = \frac{20}{3}$. 
2.4. Technical lemmas

Let us now present five technical tools needed for our later analysis. Consider an $N \times N$ grid. Suppose that a random walk process starts at the lower left corner $(0,0)$, and at each time step the process continues to the right with probability $p$ and upwards with probability $q = 1 - p$. We ask what is the expected time $T$ it takes the process to reach the right or the top boundary of the grid. The answer is given by the following lemma (whose proofs are deferred to Appendix A).

**Lemma 2.4.** For $p$ constant,

$\mathbb{E}(T) = \begin{cases} 
2N - \theta(\sqrt{N}), & p = q = \frac{1}{2}, \\
\frac{N}{q} + o(1), & p < q.
\end{cases}$

Secondly, let us calculate a product formula.

**Lemma 2.5.** Let $a, c$ and $0 < b < 1$ be constants. Let $B = 1/(1-b)$. For an integer $h$,

$$
\prod_{i=1}^{h} (a + c \cdot b^i) \leq e^{\frac{Bc}{2}} \cdot a^h.
$$

Our third technical lemma involves the solution of a recursion formula. The following fact can be easily proved by induction.

**Fact 2.6.** The recursion $f(h) = b_h + a_h \cdot f(h-1)$ solves to

$$
f(h) = b_h + f(0) \cdot \prod_{i=1}^{h} a_i + \sum_{i=1}^{h-1} b_i \prod_{j=i+1}^{h} a_j.
$$

If $a_i = a$ and $b_i = b$ for all $1 \leq i \leq h$ then the above formula simplifies to

$$
f(h) = f(0) \cdot a^h + b \cdot \sum_{i=0}^{h-1} a^i.
$$

Our fourth technical lemma is a generalization of the following fact.

**Fact 2.7.** [5] Consider an urn containing $r$ red elements and $g$ green elements, and suppose elements are taken out one by one without replacement. Then the expected number of trials until obtaining the first red element is $\frac{r+g+1}{r+1}$.

**Lemma 2.8.** Consider an urn containing $n$ elements of which $r$ are red and $g$ are green, and suppose elements are taken out one by one without replacement. Then the expected number of trials until obtaining the $j$th red element is $\frac{j(n+1)}{r+1}$.

Our fifth technical lemma concerns the expected number of trials before getting elements from both colors.
Lemma 2.9. Consider an urn containing \( r \) red elements and \( g \) green elements, and suppose elements are taken out one by one without replacement. Then the expected number of trials until obtaining elements of both colors is \( 1 + \frac{r}{g+1} + \frac{g}{r+1} \).

3. Probe complexity in the probabilistic model

In this section we study the probabilistic model, give some global bounds on probe complexity in this model for all ND coteries, and analyze the probabilistic probe complexity of some specific known constructions of ND coterie families.

Let us first establish the following simple lower bound on the probabilistic probe complexity \( \mathcal{PPC}_p(S) \) for every ND coterie \( S \) with minimal quorum size \( c \).

Lemma 3.1. For every ND coterie \( S \) with minimal quorum size \( c \), the probabilistic probe complexity of \( S \) is bounded from below by

\[
\mathcal{PPC}_p(S) \geq \begin{cases} 
2c - \Theta(\sqrt{c}), & p = q = \frac{1}{2}, \\
\frac{c}{q} + o(1), & p < q.
\end{cases}
\]

Proof. The probing algorithm must find a monochromatic witness. Since the witness must contain a quorum, its size is at least that of the minimal quorum, \( c \). Since the probability of each element to be green is \( q \), and these events are independent, the expected number of probes necessary until collecting a monochromatic set of size \( c \) is provided by Lemma 2.4 with \( N = c \). The lemma follows.

3.1. The majority system

For the Maj system there is a simple and asymptotically optimal algorithm, which operates by picking an arbitrary element at each step, until a witness is found. Optimality follows, as will be shown next, since the elements are totally symmetric at each stage. We need to have a monochromatic set of size \( N = \frac{n+1}{2} \). Again the situation resembles that of Lemma 2.4, where the probed element turning to be colored green (respectively red) corresponds to making a right (respectively up) step. We get a bound of \( n - \Theta(\sqrt{n}) \) if \( p = \frac{1}{2} \) and \( \frac{n}{2q} \) otherwise. But by Lemma 3.1, this is also the lower bound for the system. Hence we have the following.

Proposition 3.2. The probabilistic probe complexity of the Maj system is

\[
\mathcal{PPC}_p(Maj) = \begin{cases} 
n - \Theta(\sqrt{n}), & p = q = \frac{1}{2}, \\
\frac{n}{2q} + o(1), & p < q.
\end{cases}
\]

3.2. The crumbling walls system

In this section we describe an algorithm named \texttt{Probe}_CW for finding a witness for an ND coterie of the crumbling walls family, \( S \in \text{CW} \). The algorithm, described formally
Algorithm \text{Probe\_CW} \\
\textbf{Input:} A quorum system \( S = (1, n_2, \ldots, n_k) \)-CW. \\
\textbf{Output:} a witness \( W \).

1. \text{Probe} the unique element \( v_1 \) in the first row. \\
2. \text{Mode} \leftarrow c(v_1); W \leftarrow \{v_1\}. \\
3. for \( i \leftarrow 2 \) to \( k \) do:
   - \text{Probe elements of row} \( i \) until an element of color \text{Mode}, denoted \( v_i \), is found, or the end of the row is reached.
   - If an element is found then \( W \leftarrow W \cup \{v_i\} \).
   - else do:
     - (a) \( W \leftarrow \) all elements of row \( i \).
     - (b) \text{Mode} \leftarrow \neg \text{Mode}. \\

Fig. 5. Algorithm \text{Probe\_CW} for finding a witness for \text{CW}.

in Fig. 5, examines the rows of \( S \) one by one, starting from the first element of the top row, denoted \( v_1 \), and going down towards the bottom row \( k \). In stage \( i \), the algorithm probes the elements of row \( i \). While doing so, the algorithm maintains a monochromatic set \( W \) consisting of a full (previously probed) row \( j \) and one element from each row from row \( j + 1 \) to row \( i - 1 \). At any given moment, the algorithm is in one of two modes, corresponding to the color of \( W \), i.e., \text{Mode} \in \{\text{green}, \text{red}\}$. Denote the opposing mode by \( \neg \text{Mode} \). The algorithm attempts to find one element of the same color as its mode from row \( i \). If the algorithm succeeds in finding such an element, then it continues to the next row, with the element found at row \( i \) added to \( W \). If the algorithm does not succeed, then it erases the current set \( W \), and instead sets \( W \) to contain all the elements of row \( i \), which is also monochromatic, except with the opposite color. The algorithm subsequently changes its mode to this color. The algorithm now continues in the same fashion to the next row. After the last row is scanned, either one element of the color of \( W \) is found and \( W \) is a monochromatic quorum (consisting of a full row \( j < k \) and one element from each row below it) or the chosen quorum is the last row \( k \) itself.

We proceed with an analysis of Algorithm \text{Probe\_CW}.

**Theorem 3.3.** Let \( S \in \text{CW} \) be an ND coterie with \( k \) rows. Then for every \( p \), the probabilistic probe complexity of Algorithm \text{Probe\_CW} is bounded by

\[
\mathcal{PPC}_P(\text{Probe\_CW}, S) \leq 2k - 1.
\]

**Proof.** Let \( X_i \) denote the number of probed elements in row \( i \), and let \( X = \sum_i X_i \). The expected number of probed elements, \( \mathbb{E}(X) \), can be bounded as follows.

Observe that the subcollection of elements composed only of rows \( 1, \ldots, i - 1 \) also forms a CW coterie, denoted \( S_{i-1} \), with an availability probability \( \mathcal{F}_p(S_{i-1}) \). When starting to scan row \( i \), we have at hand a monochromatic set which is a witness for \( S_{i-1} \). Hence the probability for the algorithm to be in \text{Mode} = \text{red} (respectively \text{green}) is exactly \( \mathcal{F}_p(S_{i-1}) \) (respectively \( 1 - \mathcal{F}_p(S_{i-1}) \)). Since the rows are finite the expected time it takes to find a green (respectively red) element is bounded from above by a geometric random variable with parameter \( q \) (respectively \( p \)). Hence

\[
\mathbb{E}(X_i) \leq \mathcal{F}_p(S_{i-1}) \cdot \frac{1}{p} + (1 - \mathcal{F}_p(S_{i-1})) \cdot \frac{1}{q} = \frac{(q - p)\mathcal{F}_p(S_{i-1}) + p}{pq}.
\]
Since this expression is symmetric in $p$ and $q$, by Fact 2.3(2), we can assume that $p \leq 1/2$, and using Fact 2.3(1) we obtain
\[
E(X_i) \leq \frac{(q - p)F_p(S_{i-1}) + p}{pq} \leq \frac{(q - p)p + p}{pq} = 2,
\]
and subsequently conclude that
\[
E(X) = 1 + \sum_{i=2}^{k} E(X_i) \leq 1 + 2(k - 1) = 2k - 1. \square
\]

Remark. Note that the result does not depend on the number of the elements, but only on the number of rows, although in the deterministic model the probe complexity is $n$.

We conclude with some bounds concerning some specific subclasses of CW. Since the Wheel system can be described as a $(1, n - 1)$-CW with only 2 rows, we conclude:

**Corollary 3.4.** The probabilistic probe complexity of the Wheel system is bounded from above by
\[
\mathcal{PPC}_p(\text{Probe}_\text{CW}, \text{Wheel}) \leq 3.
\]

The Triang quorum system is a $(1, 2, \ldots, k)$-CW. Hence we have the following.

**Corollary 3.5.** The probabilistic probe complexity of the Triang system is bounded from above by
\[
\mathcal{PPC}_p(\text{Probe}_\text{CW}, \text{Triang}) \leq 2k - 1.
\]

The bound is away from $\mathcal{PPC}_p(\text{Triang})$ by a factor of at most $(1 + o(1))$ in case $p = 1/2$; if $p < 1/2$ then the factor is at most $2q$.

**Proof.** Since there are $k$ rows, using Theorem 3.3, $\mathcal{PPC}_p(\text{Probe}_\text{CW}, \text{Triang}) = 2k - 1$. But by Lemma 3.1, if $p = 1/2$, then no algorithm can do better than $2k - \theta(\sqrt{k})$, since Triang is a $k$-uniform quorum system. The proof for $p < 1/2$ is similar. \square

3.3. The tree quorum system

The algorithm for finding a witness for the tree system in the probabilistic model, called $\text{Probe}_\text{Tree}$, is recursive. For a tree system of height $h$, the algorithm starts by probing the root $r$. Then the algorithm tries to find, recursively, a witness $W_R$, i.e., a monochromatic quorum, for the right subtree. If the color of $W_R$ is as the color of the root $r$, then a witness for the whole tree has been found ($W_R \cup \{r\}$) and the algorithm stops. If their color is not the same, then a witness ($W_L$) is found for the left subtree. Necessarily, the color of $W_L$ must be the same as either the color of the root $r$ or the color of $W_R$, the witness of the right subtree. Either way, we have obtained a witness for the whole tree (namely, $W_L \cup \{r\}$ or $W_L \cup W_R$).
Proposition 3.6. The expected time for finding a monochromatic quorum in the Tree system by Algorithm \texttt{Probe\_Tree} is
\[
\mathcal{PPC}_p(\texttt{Probe\_Tree}, \text{Tree}) = O(n^{\log_2(1+p)}).
\]

Proof. We give a recursive analysis for the expected time complexity of Algorithm \texttt{Probe\_Tree}. We rely on the fact that the algorithm saves unnecessary probes. In particular, it always examines the root, but does not always check both sub-roots. Let \( T_p(h) \) denote the expected number of probed elements in a tree of height \( h \) until obtaining a witness. Let \( F_p(h) \) denote the probability that a tree system of height \( h \) is not available. Then \( T_p(h) \) obeys the following recursive relation:
\[
T_p(h) = 1 + q \left[ T_p(h - 1) + F_p(h - 1)T_p(h - 1) \right] + p \left[ T_p(h - 1) + (1 - F_p(h - 1))T_p(h - 1) \right].
\]
Since the expression is symmetric in \( p \) and \( q \), we can assume without loss of generality that \( p \leq 1/2 \). For such \( p \), \( F_p(h) \leq (p + 1/2)^h \), as is proved in [15]. We deduce that
\[
T_p(h) \leq 1 + \left(1 + p + (1 - 2p)(p + 1/2)^{h-1}\right) \cdot T_p(h - 1),
\]
and solving the recursion using Fact 2.6 and Lemma 2.5 gives the bound.

Corollary 3.7. For any \( 0 \leq p \leq 1 \),
\[
\mathcal{PPC}_p(\texttt{Probe\_Tree}) = O(n^{0.585}).
\]

3.4. The hierarchical quorum system

In this section we present a probing algorithm for HQS with optimal probabilistic probe complexity. Denoting the size of quorums in the system (which is uniform) by \( c \), we show that \( \mathcal{PPC}(\text{HQS}) \) is asymptotically larger than \( c \) (in particular, \( \mathcal{PPC}(\text{HQS}) = \Omega(c^{1.3}) \)).

This bound is interesting in that it refutes two plausible conjectures one might make concerning ND coteries in general and \( c \)-uniform ND coteries in particular. First, it may seem plausible that \( c \)-uniform ND coteries enjoy better probabilistic probe complexity than general coteries, and in particular, one might possibly conjecture that such a coterie \( \tilde{S} \) may have a probing algorithm with \( \mathcal{PPC}(S) = O(c) \), as happens for the Triang system for instance. The lower bound for HQS refutes this conjecture. Secondly, one might conjecture that the probabilistic probe complexity of an ND coterie is bounded from above (asymptotically) by its average quorum size. The lower bound refutes this conjecture as well.

Algorithm \texttt{Probe\_HQS} is first presented and analyzed, and then we prove its optimality. The algorithm tries to evaluate the root of the tree. It recursively evaluates two children first. If both leaves are of the same color, then the color of the root is the same. Only if the colors of the children are different then the algorithm evaluates the third child as well.
Theorem 3.8. The expected time for finding a monochromatic quorum in the HQS system by Algorithm \text{Probe}_\text{HQS} is

\[ PP \mathcal{C}_p(\text{Probe}_\text{HQS}, \text{HQS}) = \begin{cases} n^{\log_3 \frac{5}{2}} = n^{0.834}, & p = \frac{1}{2}, \\ O(n^{\log_3 2}) = O(n^{0.63}), & o.w. \end{cases} \]

Proof. Let \( h = \log_3 n \) be the height of the tree. Let \( T(h) \) denote the expected number of probed elements by Algorithm \text{Probe}_\text{HQS} and let \( \mathcal{F}_p(h) \) denote the availability of a tree of height \( h \). The following formula calculates \( T(h) \) recursively,

\[ T(h) = 2 \cdot T(h-1) + 2 \cdot \mathcal{F}_p(h-1) \cdot (1 - \mathcal{F}_p(h-1)) \cdot T(h-1). \]

If \( p = 1/2 \), then since \( \mathcal{F}_{1/2}(h) = \frac{1}{2} \) for all \( h \), we get

\[ T(h) = \frac{5}{2} \cdot T(h-1), \]

and the theorem follows. For \( p < \frac{1}{2} \) we use the bound \( \mathcal{F}_p(h) \leq p(3p - 2p^2)^h \) of [19], to conclude that

\[ T(h) \leq 2 \cdot T(h-1) + 2 \cdot \mathcal{F}_p(h-1) \cdot T(h-1) \]
\[ \leq 2 \cdot T(h-1) + 2 \cdot p(3p - 2p^2)^{h-1} \cdot T(h-1) \]
\[ = \left( 2 + 2p(3p - 2p^2)^{h-1} \right) \cdot T(h-1) \]
\[ = \prod_{i=1}^{h} \left( 2 + 2p(3p - 2p^2)^i \right). \]

Using Lemma 2.5 with \( a = 2, b = 3p - 2p^2 \) and \( c = 2p \), we obtain \( T(h) = O(n^{\log_3 2}) \). \( \square \)

We next prove the optimality of Algorithm \text{Probe}_\text{HQS} for the case \( p = \frac{1}{2} \), using similar technique as [17]. Algorithm \text{Probe}_\text{HQS} probes the leaves from left to right, such that every time a subtree root can be evaluated, the algorithm skips the remaining leaves in this subtree and does not probe them.

Definition. An algorithm \( A \) is called \( n \)-good if, whenever a descendant of a node \( x \) on level \( n \) or less is probed during the execution of \( A \), the value of \( x \) is evaluated before the probing of any other leaf which is not a descendant of \( x \).

Note that Algorithm \text{Probe}_\text{HQS} is \( h \)-good, where \( h \) is the height of the tree (levels are numbered from leaves to root). Moreover, note that any \( h \)-good algorithm will have the same probabilistic probe complexity as Algorithm \text{Probe}_\text{HQS}.

Theorem 3.9. Algorithm \text{Probe}_\text{HQS} has optimal probabilistic probe complexity, i.e.,

\[ PP \mathcal{C}_{\frac{1}{2}}(\text{Probe}_\text{HQS}, \text{HQS}) = PP \mathcal{C}_{\frac{1}{2}}(\text{HQS}). \]
Proof. The proof is based on inductively converting any algorithm $A$ which is optimal and $(k - 1)$-good but not $k$-good into an algorithm $C$ which is $k$-good and has the same probabilistic probe complexity as $A$. For the sake of the proof argument, we also define another algorithm $B$ which is based on $A$. Note that algorithm $B$ does not need to be constructed explicitly.

Since $A$ is $(k - 1)$-good but not $k$-good, there is a node $x$ on level $k$ such that on some instance, at some point during the probe process, the algorithm reaches $x$, but before the evaluation of $x$ is completed, the algorithm departs the subtree rooted at $x$ and continues to other parts of the tree. Consider the first time that this happens during the execution, and let $\beta$ denote the path followed by the algorithm $A$ in the decision tree $T_A$ corresponding to it, until reaching this point of departure. This path reaches some node $r$ in $T_A$, and from now on we consider only the probabilistic probe complexity of the subtree rooted at $r$, denoted $PPC_2(A^\beta, HQS)$. Our goal is to modify $A$ into an algorithm $C$ which avoids this first “violation of goodness” and yet maintains the same probabilistic probe complexity (or better). Repeating this process will gradually eliminate all violations and eventually yield a $k$-good algorithm.

The children of $x$ are on level $k - 1$, so by the inductive hypothesis they are evaluated before $A$ shifts its attention to other parts of the tree. We separate our calculations to two cases. In the first case, Algorithm $A$ evaluates exactly one child of $x$ and then moves to other parts of the tree. In the second case, $A$ evaluates two children of $x$ and then moves to other parts of the tree (even if $x$ is not yet evaluated, since the colors of the two children are not the same). We prove the claim only for the second case; for the first case the proof is similar.

We make use of the following notation regarding the operation of $A$. Let $\alpha$ denote the expected number of probes for evaluating a child of $x$. Let $1 - p_1$ denote the probability that $A$ evaluates the tree without getting back to $x$. Let $\gamma$ denote the expected number of probes performed by $A$ after leaving $x$ until either evaluating the tree or returning to $x$. Let $V_1$ denote the expected number of probes needed to evaluate the tree after the first two children of $x$ had the same color. Let $V_2$ denote the expected number of probes needed for $A$ to evaluate the tree after returning to $x$ and knowing the value of $x$ (since another child was probed).

We describe the two alternative algorithms $B$ and $C$ based on $A$. Algorithm $B$ operates in the same way as $A$ until the point where $A$ enters the subtree of $x$. At that point, it does not enter the subtree of $x$, but rather starts probing the part of the tree that $A$ examines after probing two children of $x$. Then it probes the first two children of $x$ and if they are not in the same color (which happens with probability $\frac{1}{2}$) it probes the third child before it continues as $A$.

Algorithm $C$ evaluates the first two children of $x$, and if $x$ is still not evaluated, it probes the third child of $x$ and continues as $A$ from that point on.

The flow of the three algorithms and the associated probabilities with which these paths are taken are described in Fig. 6.

Let us evaluate the probabilistic probe complexity of the three algorithms, and prove that the probabilistic probe complexity of $C$ is no higher than that of $A$. Immediately from the description of $A, B$ and $C$ we get
Fig. 6. Description of the three algorithms in the second case.

\[ \mathcal{PPC}_1(A^\beta, HQS) = 2\alpha + \frac{1}{2}V_1 + \frac{1}{2}\gamma + \frac{1}{2}p_1(\alpha + V_2). \]  
(1)

\[ \mathcal{PPC}_1(B^\beta, HQS) = \gamma + p_12\alpha + \frac{1}{2}p_1V_2 + \frac{1}{2}p_1(\alpha + V_2). \]  
(2)

\[ \mathcal{PPC}_1(C^\beta, HQS) = 2\alpha + V_1 + \frac{1}{2}\alpha. \]  
(3)

By the optimality of \( A \) we know that \( \mathcal{PPC}_1(A^\beta, HQS) \leq \mathcal{PPC}_1(B^\beta, HQS) \), hence by Eqs. (1) and (2),

\[ 2\alpha + \frac{V_1}{2} + \frac{\gamma}{2} + \frac{p_1}{2}(\alpha + V_2) \leq \gamma + p_1 : 2\alpha + \frac{p_1 \cdot V_2}{2} + \frac{p_1}{2}(\alpha + V_2). \]

Rearranging, we get

\[ 2\alpha(1 - p_1) \leq \frac{\gamma}{2} - \frac{V_1}{2} + \frac{p_1V_2}{2}. \]

As \( \alpha \) and \( 1 - p_1 \) are both positive, clearly also

\[ \frac{\alpha}{2}(1 - p_1) \leq \frac{\gamma}{2} - \frac{V_1}{2} + \frac{p_1V_2}{2}. \]

Rearranging and adding \( 2\alpha + V_1/2 \) to both sides, we get

\[ \frac{5\alpha}{2} + V_1 \leq \frac{\gamma}{2} + \frac{V_1}{2} + 2\alpha + \frac{p_1\alpha}{2} + \frac{p_1V_2}{2}. \]

hence \( \mathcal{PPC}_1(C^\beta, HQS) \leq \mathcal{PPC}_1(A^\beta, HQS) \), completing the proof of the claim.
It follows that any optimal probing algorithm for HQS can be gradually transformed (changing it for each node \( x \) where \( k \)-goodness is violated and for each initial sequence of probes \( \beta \) that leads to such violation), into one that is \( h \)-good, where \( h \) is the height of the tree, and is still optimal. Such algorithm will have the same probabilistic probe complexity of Algorithm \( \text{Probe\_HQS} \), thus completing the proof of the theorem. \( \square \)

4. Randomized algorithms

We now turn to worst case analysis of probing algorithms. Probe complexity was studied in the worst case model only for deterministic algorithms [15]. In this section we give almost tight lower and upper bounds for the randomized probe complexity of some classes of ND coteries. Our main tool for proving lower bounds for randomized algorithms is Yao’s theorem [20]. The theorem says that the expected time of a randomized algorithm \( A_1 \) with any distribution \( D_1 \) on random strings is always bounded from below by the expected time of the best deterministic algorithm \( A_2 \) on inputs coming from any distribution \( D_2 \). Hence in order to lower bound a randomized algorithm it suffices to find some (difficult) distribution on the input \( D_2 \) on which every deterministic algorithm will behave badly on the average.

We start with stating an elementary lower bound for any ND coterie with maximal quorum size \( m \).

**Theorem 4.1.** For any ND coterie \( S \) with maximal quorum size \( m \), the probe complexity of any randomized algorithm is at least \( \mathcal{PC}^R(S) \geq m \).

**Proof.** The algorithm must return a witness, where the size of the witness may be of size \( m \) in the worst case. This happens, for example, in case the elements of a largest quorum are colored green while all other elements are colored red. \( \square \)

4.1. The majority system

For the Maj system we give a tight analysis showing the following.

**Theorem 4.2.** The probe complexity of any randomized algorithm for the Maj system is

\[
\mathcal{PC}^R(\text{Maj}) = n - \frac{n - 1}{n + 3}.
\]

**Proof.** Let \( n = 2k + 1 \), and let us choose as the hard distribution all the possible colorings of the elements, in which \( k + 1 \) elements are colored red and \( k \) elements are colored green. Any algorithm needs to probe elements until it finds the \( k + 1 \) elements that were colored red and hence are the majority. Even if the algorithm designer is told in advance about the hard distribution we have chosen, the deterministic algorithm knows in every stage only the number of remaining elements that are colored red, but the remaining elements are totally symmetric in the sense that their probability of being red is equal, so it does not matter
which is probed first. Hence the situation is exactly as in Lemma 2.8 with parameters \( r = j = k + 1 \) and \( g = k \), which yields the lower bound

\[
\frac{j(r+g+1)}{r+1} = \frac{(k+1)(2k+2)}{k+2} = 2k+1 - \frac{2k}{2k+4} = n - \frac{n-1}{n+3}.
\]

Let us now present an Algorithm \( R_{\text{Probe Maj}} \) that achieves this bound. Algorithm \( R_{\text{Probe Maj}} \) probes elements uniformly at random. For the analysis, note that if there are initially \( r \geq k + 1 \) elements colored red and \( g \) elements colored green then Algorithm \( R_{\text{Probe Maj}} \) stops when it finds \( k + 1 \) red elements. The situation is exactly the same as in Lemma 2.8 with \( j = k + 1 \) so we get that the probe complexity of Algorithm \( R_{\text{Probe Maj}} \) for input with \( r \) elements colored red is

\[
\frac{r(r+g+1)}{r+1}.
\]

The worst case input for Algorithm \( R_{\text{Probe Maj}} \) is when \( r = k + 1 \) hence achieving again the same bound. \( \square \)

4.2. The crumbling walls system

For the Crumbling Walls family we present a randomized algorithm which is almost tight. Algorithm \( R_{\text{Probe CW}} \) for a \((n_1, \ldots, n_k)\)-CW starts at the lowest row, \( k \). It probes elements at random until finding elements from both colors or exhausting the entire row. If the whole row is monochromatic then the algorithm stops, otherwise it continues to the next (higher) row. The algorithm always succeeds in obtaining a witness since when it terminates it has seen a monochromatic row as well as one element of the same color from each row below it. Now we analyze the expected time for this algorithm.

In each row \( i \), Algorithm \( R_{\text{Probe CW}} \) probes elements until finding elements from both colors. Using Lemma 2.9 with \( r \) red elements, \( g \) green elements and \( n_i = g + r \), we deduce the expected number of probes. The maximum of this expected number of probes is attained at \( g = 1 \) or \( r = 1 \). We conclude with the next corollary.

**Corollary 4.3.** The expected number of probes in a row \( i \) with \( r + g = n_i \) elements is at most \( \frac{n_i + 1}{2} + \frac{1}{n_i} \).

Let \( m \) be the length of the maximal row.

**Theorem 4.4.** The probe complexity of Algorithm \( R_{\text{Probe CW}} \) for \( S = (1, n_2, \ldots, n_k)\)-CW is

\[
\mathcal{PC}_R(R_{\text{Probe CW}}, S) = \max_j \left\{ n_j + \sum_{i=j+1}^k \left( \frac{n_i + 1}{2} + \frac{1}{n_i} \right) \right\}.
\]

Hence for any CW coterie \( S \) the probe complexity of Algorithm \( R_{\text{Probe CW}} \) is bounded by

\[
\mathcal{PC}_R(R_{\text{Probe CW}}, S) \leq \frac{m + n + 2k}{2}.
\]

**Proof.** Algorithm \( R_{\text{Probe CW}} \) terminates when it encounters a monochromatic row \( j \). Before reaching this row it probes elements at random in each row until finding elements
from both colors. By Corollary 4.3, the expected number of probes in row \( i > j \) is less than \( \frac{n_j + 1}{2} + \frac{1}{n_i} \). The first part of the theorem follows. We bound this value by

\[
\max_j \left\{ n_j + \sum_{i=j+1}^{k} \left( \frac{n_i + 1}{2} + \frac{1}{n_i} \right) \right\} \leq m + \frac{n - m + k}{2} + \frac{k}{2} = \frac{m + n + 2k}{2}. \quad \Box
\]

**Corollary 4.5.**

1. \( \mathcal{PC}^R(\text{R}_\text{Probe}_\text{CW}, \text{Triang}) \leq \frac{n+k}{2} + \log k. \)
2. \( \mathcal{PC}^R(\text{R}_\text{Probe}_\text{CW}, \text{Wheel}) = n - 1. \)

**Proof.** Using Theorem 4.4, it is easy to check that for the Wheel system the maximum is attained at \( j = 2 \) and for the Triang system the maximum is attained at \( j = 1 \). \( \Box \)

We now prove a lower bound for the randomized probe complexity of a CW system.

**Theorem 4.6.** For any \((1, n_2, \ldots, n_k)\)-CW system,

\[
\mathcal{PC}^R((1, n_2, \ldots, n_k)\text{-CW}) \geq \frac{n+k}{2}.
\]

**Proof.** As a difficult distribution on the input for a deterministic probing algorithm \( B \), we choose the uniform distribution restricted to inputs containing exactly one green element in each row. The expected number of probes in row \( i \) equals the number of elements probed randomly before finding the green element, which is, by Fact 2.7, \( \frac{n_i + 1}{2} \). Hence

\[
\mathcal{PC}^R((1, n_2, \ldots, n_k)\text{-CW}) \geq \sum_{i=1}^{k} \frac{n_i + 1}{2} = \frac{n+k}{2}. \quad \Box
\]

4.3. The tree system

Since the Tree system has a quorum of size \( n/2 \) (consisting of all the leaves of the tree), any worst case algorithm must probe more than \( n/2 \) elements. We prove that there is a randomized algorithm whose expected number of probes in the worst case is \( 5n/6 \). This is better than in the deterministic case, where it is known that \( \mathcal{PC}(\text{Tree}) = n \) (Lemma 2.2). We also prove a slightly stronger lower bound of \( \mathcal{PC}^R(\text{Tree}) \geq 2n/3 \).

In a tree system of height \( h \) there are \( n = 2^{h+1} - 1 \) elements. Algorithm \text{R}_\text{Probe}_\text{Tree} \) chooses (recursively) uniformly at random one of three possibilities. The first two possibilities are to probe the root with one of its subtrees and examine the other subtree only if a witness is not found. The third possibility is to probe the two subtrees first and examine the root only if a witness is not found.

**Theorem 4.7.** The randomized probe complexity of Algorithm \text{R}_\text{Probe}_\text{Tree} is bounded from above by \( \frac{5n}{6} + \frac{1}{6} \).
Proof. One of the three possibilities must always reveal a witness. With probability at least $1/3$ the algorithm will choose it, and in the worst case this possibility is the last to be explored. Denote by $T_h$ the randomized probe complexity of Algorithm $R_{\text{Probe\_Tree}}$ for the Tree system with a tree of height $h$. Hence we calculate recursively,

$$T_h \leq \frac{2}{3} \cdot (1 + 2T_{h-1}) + \frac{1}{3} \cdot 2T_{h-1} = \frac{2}{3} + 2T_{h-1}.$$  

Using Fact 2.6 with $T_0 = 1$, $a = 2$ and $b = \frac{2}{3}$ we get

$$T_h \leq 2^h + \frac{2}{3} \cdot \sum_{i=0}^{h-1} 2^i = 2^h + \frac{2}{3} \cdot (2^h - 1)$$

$$= \frac{n + 1}{2} + \frac{2}{3} \cdot \left( \frac{n + 1}{2} - 1 \right) = \frac{5n}{6} + \frac{1}{6}. \qed$$

Next we prove a lower bound for $PC^R(\text{Tree})$, again using Yao’s theorem.

Theorem 4.8. The randomized probe complexity of the Tree system satisfies $PC^R(\text{Tree}) \geq \frac{2}{3} \cdot (n + 1)$.

Proof. We first define a hard distribution of inputs on which the average number of probes for a deterministic algorithm is $\frac{2}{3} \cdot (n + 1)$. All the nodes on levels $i = 2, \ldots, h$ are colored green. There are $\frac{n+1}{4}$ subtrees of height one and size 3, with a root and two children, hanging from the nodes of level 2. Exactly two out of the three nodes in each such subtree are colored red. For each subtree we have three possibilities of choosing the two red nodes out of the three and the distribution is uniform and independent on each subtree. Hence in each tree in the input distribution there is a red witness and the algorithm has to find it. The witness is composed of the union of all the pairs of nodes colored red in each subtree.

Fixing a particular subtree $T'$, denote by $E_i$ (for $i \in \{1, 2, 3\}$) the event that the green node was probed on the $i$th probing, and denote by $X$ the random variable that counts the number of probes before finding two red nodes. Then the expected number of probes in $T'$ is

$$E(X) = \sum_{i=1}^{3} \frac{1}{3} \cdot E(X \mid E_i) = \frac{1}{3} \cdot (3 + 3 + 2) = \frac{8}{3}.$$

Summing over all $\frac{n+1}{4}$ such subtrees, the theorem follows. \qed

4.4. The hierarchical quorum system

A randomized algorithm for probing the HQS system, suggested by Ravi Boppana, is mentioned in [16]. The algorithm, named $R_{\text{Probe\_HQS}}$, is defined recursively as follows. Choose two children of the root at random and evaluate them recursively. If they agree on the color then finish, otherwise probe the third child. See Fig. 7 for more formal description. The analysis of [16] establishes the following proposition.
Algorithm $\text{R\_Probe\_HQS}(v)$

**Input:** An HQS with root $v$.

**Output:** a witness $W$.

1. Choose uniformly at random $v_1$ and $v_2$ two children of $v$.
2. Compute recursively $\text{R\_Probe\_HQS}(v_1)$ and $\text{R\_Probe\_HQS}(v_2)$.
3. If not computed to the same color then compute $\text{R\_Probe\_HQS}(v_3)$.

Fig. 7. Randomized Algorithm $\text{R\_Probe\_HQS}$ for finding a witness for HQS.

**Proposition 4.9.** [16] The randomized probe complexity of Algorithm $\text{R\_Probe\_HQS}$ is

$$\mathcal{PC}_R(\text{R\_Probe\_HQS}, \text{HQS}) = O(n^{\log_3 \frac{8}{3}}) = O(n^{0.893}).$$

The randomized probing Algorithm $\text{R\_Probe\_HQS}$ for the HQS system can be described recursively. In order to evaluate the root of the tree choose uniformly at random two children and evaluate the third child only if they do not agree.

We now propose an improved algorithm. The improvement is achieved by evaluating only one child and then trying to use the information collected from it, by realizing that now the value of the root is biased. If the value of the first child is 1 then since it is chosen uniformly at random the probability that the value of the tree is 1 is more than $1/2$ (since 2 children out of 3 have the majority value). Hence, when trying to evaluate the next child we first evaluate one grandchild. If the value of the grandchild is 0 we suspect that we are not in the right majority child and we skip to the second child. We prove that introducing this modification to Algorithm $\text{R\_Probe\_HQS}$ improves its randomized probe complexity from $O(n^{0.893})$ to $O(n^{0.887})$, getting closer to our lower bound of $O(n^{0.834})$.

Algorithm $\text{IR\_Probe\_HQS}$ is described formally in Fig. 8. Every node in the computation tree of the HQS has a value. Assigning values to the elements fixes the values to the nodes of the computation tree. Denote the value of any node $r$ by $\text{val}(r)$. The target of the algorithm is to evaluate the value of the root $r$. If the algorithm tries to evaluate the value of a node $x$ with height $h_x$ then it usually makes some recursive calls to itself with input nodes $y_j$ of height $h_x - 2$. When describing the algorithm, the term “to evaluate” the value of some node $r_i$ means to choose uniformly at random some order of the children of $r_i$, then evaluate recursively one after one the children of $r_i$, according to the chosen order, until the value of $r_i$ is found.

**Theorem 4.10.** The randomized probe complexity of Algorithm $\text{IR\_Probe\_HQS}$ is

$$\mathcal{PC}_R(\text{IR\_Probe\_HQS}, \text{HQS}) = O(n^{0.887}).$$

**Proof.** We break the proof of the theorem to two main lemmas. The first lemma finds the worst case input for Algorithm $\text{IR\_Probe\_HQS}$. The second lemma analyzes the expected number of recursive calls made by the algorithm. Finally, the average number of probes is computed by solving the recursive formula obtained in the second lemma.

We define a collection of inputs $P$. An input $y$ belongs to $P$ if it satisfies the following recursive property: If the root $r$ has value $\text{val}(r)$ then exactly two out of three of its children have value $\text{val}(r)$. 


Algorithm IR_Probe_HQS

Input: A quorum system HQS with a root $r$.
Output: The value of $r$.

1. Choose, uniformly at random, one child of the root $r$, denoted $r_1$.
2. Evaluate $r_1$.
3. Choose uniformly at random another child of $r$, denoted $r_2$.
4. Choose uniformly at random one child of $r_2$, denoted $r_{2,1}$, and recursively evaluate its value, $\text{val}(r_{2,1})$.
5. If $\text{val}(r_1) = \text{val}(r_{2,1})$ then
   • Continue to evaluate $\text{val}(r_2)$.
   • If $\text{val}(r_1) = \text{val}(r_2)$ then return $\text{val}(r_1)$
     else evaluate the third child $r_3$ and return $\text{val}(r_3)$.
6. else
   • Evaluate $\text{val}(r_3)$.
   • If $\text{val}(r_1) = \text{val}(r_3)$ then return $\text{val}(r_1)$
     else continue to evaluate $\text{val}(r_2)$ and return $\text{val}(r_2)$.

Fig. 8. Improved randomized Algorithm IR_Probe_HQS for finding a witness for HQS.

Lemma 4.11. The worst case input for Algorithm IR_Probe_HQS belongs to $P$.

Proof. Without loss of generality let us assume that the value of the root $r$ is 1. We next describe the worst case input for Algorithm IR_Probe_HQS in terms of the values of $r$’s grandchildren. The values of the grandchildren of two of the children of $r$ are $\{1, 1, 0\}$ and the values of the grandchildren of the third child of $r$ are $\{0, 0, 1\}$. Since we need to find the two children with the majority value the worst input for each of the majority children is $\{1, 1, 0\}$ for their children because otherwise the values are $\{1, 1, 1\}$ and only two grandchildren are needed to be evaluated (and not $2^2$). For the third child, first the worst case input should be such an input that evaluates the third child to 0 (because otherwise we again need only to evaluate any 2 children). If the children of the third child are with values $\{0, 0, 0\}$ then either we need to evaluate this child and then we evaluate recursively only 2 or if it was chosen as $r_2$ then we do not waste evaluations by doing step 5. □

In the following computation we use the next two facts, which can be concluded from Lemma 2.8 with the right parameters. First, the number of recursive evaluations needed to find the two children with value 1 where the third child value is 0, is $2^2$. Second, if 1 is found in the first evaluation then in order to find the next 1 we need another $1 \frac{1}{2}$ recursive evaluations on the average.

Lemma 4.12. If Algorithm IR_Probe_HQS needs to evaluate a node $r$ of height $h$ then the algorithm makes (on the average) $\frac{189.5}{27}$ calls to nodes of height $h - 2$.

Proof. Let us calculate the average number of recursive calls to grandchildren made by Algorithm IR_Probe_HQS. The calculation has to be made on a worst case input. By Lemma 4.11 the calculation has to be made on any input in $P$.

Our computation is described as an average computation directed tree $F$ in Fig. 9. Every time a random decision is taken by the algorithm the tree node adds two edges to the two choices, with the corresponding probabilities written next to the edge. The number
Fig. 9. The expected number of recursive calls of Algorithm IR_Probe_HQS described as a directed tree $F$. Each rectangle describes the average number of recursive computations and each arrow describes some probabilistic decision made by the algorithm. The decision made and its probability are written next to the arrow.

of grandchildren recursive calls is written inside every (rectangle) node. It is now left to calculate the number of grandchildren recursive calls for each leaf of the tree $F$ multiplied by the probabilities on the edges leading from the root to the leaf (since we need to calculate the average).

We next present a detailed explanation to the left routes in $F$ as described in Fig. 9 and its correspondence to Algorithm IR_Probe_HQS. The explanation is done on the worst case input mentioned above. With probability $\frac{2}{3}$, the value of the first child $r_1$ is as the value of $r$ (the majority value). In order to evaluate $r_1$ evaluate recursively its children. We need to make $2 \cdot \frac{2}{3}$ recursive calls on the average. Next, with probability $\frac{1}{2}$ Algorithm IR_Probe_HQS chooses the next child $r_2$ to be again the child with same value as $r_1$, the majority value. Next we evaluate recursively one child, denoted $r_{2,1}$ of $r_2$. With probability $\frac{2}{3}$ the grandchildren $r_{2,1}$ has the same value as $r_2$, and in this case we continue to evaluate the value of $r_2$ by evaluating recursively the other grandchildren, hence the average number of recursive calls is $1 \frac{1}{2}$.

In summary, the average number of recursive calls, as described in $F$ is

$$
2 \cdot \left( \frac{2}{3} + 1 + \frac{1}{2} \cdot \frac{2}{3} \cdot \frac{1}{2} + \frac{1}{3} \cdot \left( 2 \cdot \frac{2}{3} + \frac{1}{2} \right) \right) + \frac{1}{2} \left( \frac{2}{3} \cdot \frac{2}{3} + \frac{1}{3} \cdot \left( 2 + 2 \cdot \frac{2}{3} \right) \right) = 189.5.
$$

Hence we conclude the following recursive probe complexity for Algorithm IR_Probe_HQS. Denoting the probe complexity of a tree of height $h$ by $g(h)$, one obtains

$$
g(h) = \frac{189.5}{27} \cdot g(h - 2),
$$

and the theorem is now proved using Lemma 2.6.
In the other direction, by the optimality of Algorithm Probe\_HQS for the uniform distribution and using Yao’s theorem we conclude the following.

**Corollary 4.13.** The randomized probe complexity of the HQS system satisfies $\mathcal{PCR}(\text{HQS}) = \Omega(n^{\log_3 5}) = \Omega(n^{0.834})$.

**Appendix A. Proof of technical lemmas**

**Proof of Lemma 2.4.** The process ends after performing exactly $N$ right steps or up steps. Hence in order to calculate $\mathbb{E}(T)$ it is sufficient to calculate the absolute expected difference between the number of right steps and up steps. Denoting this difference by $S$, we have $\mathbb{E}(T) = N + (N - S) = 2N - S$. Denote by $S_t$ the difference after $t$ steps. If $p = q = 1/2$ then $S_t$ equals the expected absolute value of the distance from the origin of a one-dimensional random walk taking right/left steps with probability $1/2$. It is well known that this value is $\theta(\sqrt{t})$. In our case $N < t < 2N$, and the result follows.

If $p < q$ then the probability that the random walk will reach the right border is exponentially small. Hence, we calculate the expected time it takes the process to make $N$ up steps. This is just the sum of $N$ geometric random variables with parameter $q$, i.e., the negative binomial distribution. Hence $\mathbb{E}(T) = Nq + o(1)$.

**Proof of Lemma 2.5.** Let us calculate this product.

$$\prod_{i=1}^{h} (a + c \cdot b^i) = a^h + a^{h-1} \sum_{i=1}^{k} c \cdot b^i + a^{h-2} \sum_{1 \leq i < j \leq k} c^2 \cdot b^i b^j + \cdots$$

$$\leq a^h + a^{h-1} \sum_{i=0}^{\infty} b^i + a^{h-2} c^2 \cdot \left( \sum_{i=0}^{\infty} b^i \right) \left( \sum_{j=0}^{\infty} b^j \right) + \cdots$$

$$= a^h \cdot \left( \sum_{i=0}^{h} \frac{(bc/a)^i}{i!} \right) \leq e \frac{bc}{a} \cdot a^h.$$

**Proof of Lemma 2.8.** Let $T_j$ be a random variable denoting the trial in which the $j$th red element is obtained. Let $B_j = T_j - T_{j-1}$ be a random variable denoting the number of trials it took to find the first red element after finding $j - 1$ red elements. After $T_{j-1}$ trials, the urn contains $R = r - (j - 1)$ red elements and $G = g - T_{j-1} + (j - 1)$ green elements. By Fact 2.7, it takes

$$\mathbb{E}(B_j) = \frac{R + G + 1}{R + 1} = \frac{n + 1 - \mathbb{E}(T_{j-1})}{r - j + 2}$$

trials until obtaining the next red element. By the definition of $B_j$ and the linearity of expectation we conclude

$$\mathbb{E}(T_j) = \mathbb{E}(T_{j-1}) + \frac{n + 1 - \mathbb{E}(T_{j-1})}{r - j + 2},$$
giving the recurrence
\[ E(T_j) = \frac{n+1}{r+2-j} + \frac{r+1-j}{r+2-j} E(T_{j-1}). \]

Let \( z = \frac{n+1}{r+2-j} \). We solve this for any \( j \leq r \) by using Fact 2.6 with \( b_i = \frac{n+1}{r+2-i}, a_i = \frac{r+1-i}{r+2-i} \) and \( E(T_0) = 0 \), to obtain
\[
E(T_j) = z + \sum_{i=1}^{j-1} \frac{n+1}{r+2-i} \prod_{l=i+1}^{j} \frac{r+1-l}{r+2-l} = z + (n+1) \cdot \sum_{i=1}^{j-1} \frac{1}{r+2-i} \cdot \frac{r+1-j}{r+1-i}
\]
\[
= z + (n+1) \cdot \left( r+1-j \right) \cdot \sum_{i=1}^{j-1} \left( \frac{1}{r+1-i} - \frac{1}{r+2-i} \right)
\]
\[
= z + (n+1) \cdot \left( r+1-j \right) \cdot \left( \frac{1}{r-j+2} - \frac{1}{r+1} \right)
\]
\[
= z + (n+1) \cdot \left( \frac{j}{r+1} - \frac{1}{r-j+2} \right) = \frac{(n+1)j}{r+1}.
\]

**Proof of Lemma 2.9.** Let \( X \) be a random variable counting the number of elements taken out until obtaining elements from both colors. Denote by \( E_g \) (respectively \( E_r \)) the event that the first element inspected was colored green (respectively red). If the first element is colored green then we have to calculate the time it takes to first encounter an element colored red in an urn with \( g-1 \) green elements and \( r \) red elements. By Fact 2.7 this time is \( \frac{g+r}{r+1} \). Hence
\[
E(X) = \mathbb{E}(X \mid E_g) + \mathbb{E}(X \mid E_r) = \frac{g}{g+r} \cdot \left( 1 + \frac{g+r}{r+1} \right) + \frac{r}{g+r} \cdot \left( 1 + \frac{g+r}{g+1} \right)
\]
\[
= 1 + \frac{g}{r+1} + \frac{r}{g+1}.
\]

**References**


