Adaptive Majority Problems for Restricted Query Graphs and for Weighted Sets*

Gábor Damásdi[c], Dániel Gerbner[a], Gyula O.H. Katona[a], Balázs Keszegh[a,c], Dániel Lenger[c], Abhishek Methuku[b], Dániel T. Nagy[a], Dömötör Pálvölgyi[c], Balázs Patkós[a], Máté Vizer[a], Gábor Wiener[d]

[a] Alfréd Rényi Institute of Mathematics, Hungarian Academy of Sciences

[b] École Polytechnique Fédérale de Lausanne

[c] MTA-ELTE Lendület Combinatorial Geometry Research Group, Eötvös Loránd University

[d] Dept. of Computer Science and Information Theory, Budapest Univ. of Technology and Economics

March 21, 2019

Abstract

Suppose that the vertices of a graph G are colored with two colors in an unknown way. The color that occurs on more than half of the vertices is called the *majority color* (if it exists), and any vertex of this color is called a *majority vertex*. We study the problem of finding a majority vertex (or show that none exists), if we can query edges to learn whether their endpoints have the same or different colors. Denote the least number of queries needed in the worst case by $m(G)$. It was shown by Saks and Werman that $m(K_n) = n - b(n)$, where $b(n)$ is the number of 1's in the binary representation of n.

In this paper we initiate the study of the problem for general graphs. The obvious bounds for a connected graph G on n vertices are $n - b(n) \le m(G) \le n - 1$. We show that for any tree T on an even number of vertices we have $m(T) = n - 1$, and that for any tree T on an odd number of vertices, we have $n - 65 \le m(T) \le n - 2$. Our proof uses results about the weighted version of the problem for K_n, which may be of independent interest. We also exhibit a sequence G_n of graphs with $m(G_n) = n - b(n)$ such that G_n has $O(nb(n))$ edges and n vertices.

*Research supported by the Lendület program of the Hungarian Academy of Sciences (MTA), under grant number LP2017-19/2017, the János Bolyai Research Fellowship of the Hungarian Academy of Sciences, the National Research, Development and Innovation Office – NKFIH under the grants K 116769, K 124171, SNN 129364 and KH 130371 and by the BME-Artificial Intelligence FIKP grant of EMMI (BME FIKP-MI/SC).

1 Introduction

Given a set X of n balls and an unknown coloring of X with a fixed set of colors, we say that a ball $x \in X$ is a *majority ball* if its color class contains more than $|X|/2$ balls. The *majority problem* is to find a majority ball (or show that none exists). In the basic model of majority problems, one is allowed to ask queries of pairs (x, y) of balls in X to which the answer tells whether the color of x and y is the same or not, which we denote by SAME and DIFF, respectively. The answers are given by an *Adversary* whose goal is to force us to use as many questions as possible. It is an easy exercise to see that if the number of colors is two, then in a non-adaptive search (all queries must be asked at once) the minimum number of queries to solve the majority problem is $n-1$, unless n is odd, in which case $n-2$ queries suffice. On the other hand, Fisher and Salzberg [8] proved that if we do not have any restriction on the number of colors, $\lceil 3n/2 \rceil - 2$ queries are necessary and sufficient to solve the majority problem adaptively (the answer to a query is known before asking the next one). If the number of colors is two, then Saks and Werman [15] proved that the minimum number of queries needed in an adaptive search is $n - b(n)$, where $b(n)$ is the number of 1's in the binary form of n (we note that there are simpler proofs of this result, see [1, 13, 16]). There are several other generalizations of the problem, which include more colors [2, 4, 9, 11], larger queries [3, 4, 6, 7, 10, 11, 12], non-adaptive [1, 5, 9], weighted versions [9].

In the present paper we study the adaptive majority problem for two colors when we restrict the set of pairs that can be queried to the edges of some graph G on n vertices. The original majority problem, where we can ask any pair, corresponds to $G = K_n$. To distinguish between the version when we are restricted to the edges of a graph, and the original, unrestricted version, we call the colored objects *vertices* and *balls*, respectively.

Notice that it is possible to solve the majority problem (with any number of queries) if and only if G is connected when n is even, and if and only if G has at most two components when n is odd. For any such graph, denote the minimum number of queries needed to solve the majority problem in the worst case by $m(G)$. Obviously we have $n - b(n) = m(K_n) \leq m(G) \leq n - 1$ (moreover, $m(G) \leq n - 2$ when n is odd). Our main results are the following.

Theorem 1.1. *For every tree T on an even number n of vertices $m(T) = n - 1$ and for every tree T on an odd number n of vertices $m(T) \geq n - 65$.*

The constant 65 is probably far from optimal, it is possible that $m(T) \geq n-3$ holds for every tree, but we could not even prove this for paths.

We also study the least number of edges a graph must have if we can solve the majority problem as fast as in the unrestricted case, i.e., when $m(G) = n - b(n)$.

Theorem 1.2. *For every n, there is a graph G with n vertices and $O(nb(n))$ edges such that $m(G) = n - b(n)$.*

It would be interesting to determine whether this bound can be improved to $O(n)$, or show a superlinear lower bound.

The proof of Theorem 1.1 uses a *weighted* version of the original (i.e., $G = K_n$ case of the) majority problem, which is defined in the next section. We think these results are interesting on their own.

In the following, we always suppose that only two colors are used, which we call red and blue. When both colors contain the same number of balls, then we call the coloring *balanced*.

2 Weighted majority problems

Now we define a new variant of the majority problem, where the balls are given different weights. More precisely, given k balls with non-negative integer weights $w_1, \dots, w_k$, a ball i is a (weighted) *majority ball* if the weight of its color class is more than $\sum_{i=1}^k w_i/2$. The (weighted) *majority problem* is to find a majority ball (or show that none exists).

Note that during the running of an adaptive algorithm solving the non-weighted majority problem, at any point the information obtained so-far can be represented by a graph whose vertices are all balls, and the queries asked are edges labeled with DIFF or SAME. Since now we study the majority problem only for two colors, we can deduce from the labels of the edges the color partition inside every component. Denote the difference between the sizes of the color classes in each component by w_i. Finishing the algorithm from a given state is equivalent to solving the majority problem with the weights w_i.[1] Similarly, in the weighted version when we ask a ball with weight w_i and a ball with weight w_j, we can consider the answer as merging the two balls into a ball with weight $w_i + w_j$ or $|w_i - w_j|$, depending on the answer. We will say that the new ball *contains* the two previous balls.

A set of k balls with given weights $w_1, \dots, w_k$ can be represented by a vector $\underline{w} = (w_1, \dots, w_k)$. We denote the number of queries needed to solve the weighted majority problem in the worst case by $m(\underline{w})$. So, with this notation, the result of Saks and Werman for the non-weighted problem can be written as $m(1, \dots, 1) = k - b(k)$. Note that $m(\underline{w}) \leq k - 1$ and if $\sum_{i=1}^k w_i$ is odd, then $m(\underline{w}) \leq k - 2$ (if $k \geq 2$).

The weighted problem was first studied in [9], where the following proposition (which also implies the result of Saks and Werman) was proved, generalizing a result of [13] (which built on [14]) about the non-weighted variant. Let $\mu(k)$ denote the largest l such that 2^l divides k (and define $\mu(0) = \infty$). For $\underline{w} = (w_1, \dots, w_k)$ denote by p the number of balanced colorings and by p_i the number of (non-balanced) colorings such that w_i is in the majority class.[2]

Proposition 2.1. ***(i)*** *$m(\underline{w}) \geq k - \mu(p)$.*

(ii) *$m(\underline{w}) \geq k - 1 - \mu(p_i)$ for every $i \leq k$.*

[1]This is explained in more details in Section 3.

[2]Beware that in [9] a slightly different notation was used, where p denoted the number of balanced 2-partitions, which is half of the number of balanced colorings, and part (ii) of Proposition 2.1 was not explicitly stated.

It was also shown in [9] that $m(\underline{w}) = k - \mu(p)$ for $\mu(p) \leq 2$, but not in general, e.g., for $\underline{w} = \{1, 2, 3, 4, 5, 6, 7\}$ we have 8 balanced colorings, but $m(\underline{w}) = 5 > 7 - \mu(8)$.

Our main results about the weighted majority problem are exact bounds for some special $\underline{w}$. They are based on the following lemma.

Lemma 2.2. ***(i)*** *If $w_1 = \cdots = w_{2^n} = 1$ and $\sum_{i=1}^k w_i = 2^{n+1}$, then $m(\underline{w}) = k - 1$.*

(ii) *If $w_1 = \cdots = w_{2^n} = 1$, $\sum_{i=1}^k w_i = 2^{n+1} + 1$ and $w_k \neq 2^n$, then $m(\underline{w}) = k - 2$.*

Note that $w_k \neq 2^n$ implies $k > 2^n + 1$, as $w_i \neq 0$.

Proof. For both statements we use Proposition 2.1. Let us start with **(i)**. We are going to calculate $\mu(p)$. First color only the balls whose index is from $\{2^n + 1, \dots, k\}$. Denote by B and R, respectively, the blue and red balls whose index is from $\{2^n + 1, \dots, k\}$. Let $x := |\sum_{i \in B} w_i - \sum_{i \in R} w_i|$. Note that x is an even integer and there are $2\binom{2^n}{2^{n-1}-x/2}$ ways in which we can color $\{1, \dots, 2^n\}$ to make the coloring balanced. It is easy to see that this number is divisible by 4 except for the case $x = 2^n$, when this number is exactly 2. This means that the number of balanced colorings is 2 mod 4. Thus Proposition 2.1 (i) gives $m(\underline{w}) \geq k - 1$, and since $m(\underline{w}) \leq k - 1$ always holds, we have equality.

The proof of **(ii)** goes similarly. We are going to calculate $\mu(p_k)$. We define B and R in the same way. Without loss of generality we can assume that ball k is blue and let $y = \sum_{i \in B} w_i - \sum_{i \in R} w_i$. Then the number of colorings of the first 2^n balls such that blue is the majority color is $\sum_{i=2^{n-1}-\lfloor y/2 \rfloor}^{2^n} \binom{2^n}{i}$. If $y < 2^n$, each term here is divisible by 2, except the last one, while if $y > 2^n$, everything is divisible by 2. There are 2^{k-1-2^n} ways to color the balls from $\{2^n + 1, \dots, k - 1\}$ out of which $y > 2^n$ only once. This means that the number of colorings (of *all* the balls) where blue is the majority color is odd when ball k is blue, and of course the same is true when ball k is red. Thus Proposition 2.1 (ii) gives $m(\underline{w}) \geq k - 2$, and since $m(\underline{w}) \leq k - 2$ holds whenever $\sum_{i=1}^k w_i$ is odd, we have equality. □

Corollary 2.3. ***(i)*** *If $w_1 = \cdots = w_{2^n+2s} = 1$ and $\sum_{i=1}^k w_i = 2^{n+1} + 2s$, then $m(\underline{w}) \geq k - 1 - s$.*

(ii) *If $w_1 = \cdots = w_{2^n+2s} = 1$, $\sum_{i=1}^k w_i = 2^{n+1} + 2s + 1$ and $w_k \neq 2^n$, then $m(\underline{w}) \geq k - 2 - s$.*

Proof. We only prove **(i)** - the proof of **(ii)** goes the same way. First, we prove the weaker statement that $m(\underline{w}) \geq k - 1 - 2s$. We reveal of $2s$ balls of weight 1 that half of them are red and half of them are blue, and then apply Lemma 2.2 for the remaining balls.

For the bound $m(\underline{w}) \geq k - 1 - s$ we need one more trick. We run the adversarial algorithm that gives the lower bound in Lemma 2.2, until the first ball of weight 1 is queried. When this happens, then we reveal that it is red, and we reveal another ball of weight 1 that it is blue. We do this s times, and after that proceed according to the adversarial algorithm. □

Call a vector $\underline{w} = (w_1, \dots, w_k)$ *hard* if $m(\underline{w}) = k - 1$ and $\sum_{i=1}^n w_i$ is even, or $\sum_{i=1}^n w_i$ is odd and $m(\underline{w}) = k - 2$. Thus Lemma 2.2 states that the vectors satisfying its conditions are hard.

Observation 2.4. *Let $\underline{w} = (w_1, w_2, \dots, w_k)$ be a hard vector with $w_1 = w_2$ and let $\underline{w}' = (2w_1, w_3, w_4, \dots, w_k)$. Then $\underline{w}'$ is also hard.*

Proof. If $k = 2$, then $\underline{w}'$ is hard by definition. Let us assume that $k \geq 3$ and $\underline{w}'$ is not hard. We will show that $\underline{w}$ is not hard either. We ask w_1 and w_2 in the first query. If the answer is DIFF, we can obviously finish with $k-3$ further queries if $\sum_{i=1}^{k} w_i$ is even and $k-4$ further queries if $\sum_{i=1}^{k} w_i$ is odd, thus $\underline{w}$ cannot be hard. If the answer is SAME, we apply our algorithm for $\underline{w}'$ to reach the same conclusion using that $\underline{w}'$ is not hard. $\square$

Question 2.5. *Does the reverse direction also hold in the above observation?*

Also, the respective statement might hold when $m(\underline{w})$ is smaller, but we do not know of other, generally applicable sufficient conditions.

Combining Observation 2.4 with Lemma 2.2, we obtain the following statement.

Lemma 2.6. *If $w_1, \dots, w_j$ are each powers of two and*

(i) *$\sum_{i=1}^{j} w_i = 2^n$ and $\sum_{i=1}^{k} w_i = 2^{n+1}$, then $\underline{w}$ is hard, i.e., $m(\underline{w}) = k-1$.*

(ii) *$\sum_{i=1}^{j} w_i = 2^n$, $\sum_{i=1}^{k} w_i = 2^{n+1}+1$ and $k > 2^n + 1$, then $\underline{w}$ is hard, i.e., $m(\underline{w}) = k-2$.*

Note that **(i)** of Lemma 2.2 states that if the sum of the weights in $\underline{w}$ is a power of two, and at least half of that weight is given by balls of weight 1, than $\underline{w}$ is hard. Lemma 2.6 shows that weight 1 can be replaced by any weights that are powers of two, and **(ii)** of Lemma 2.2 can be similarly improved. If Observation 2.4 held for non-hard vectors as well, then we could obtain a similar improvement of Corollary 2.3 from Lemma 2.6. In general, one can ask the following.

Question 2.7. *What is the complexity of computing $m(\underline{w})$?*

The next subsection, contrary to its title, is not so relevant for our proof, but it helps to understand better what happens before the algorithm ends.

2.1 Relevant balls

Given $\underline{w} = (w_1, \dots, w_k)$, we say that w_i is *relevant*[3] if there is a coloring of the other balls such that the color of w_i changes what the majority color is, or whether a majority color exists. In other words, there is a coloring of the other balls, such that either w_i is red means red is majority and w_i is blue means blue is majority, or one color of w_i means there is no majority, the other color means there is majority. In this subsection we prove some simple facts about relevant balls. We start with some simple observations.

[3]In [9] the property that every ball is relevant was called *non-slavery*.

Proposition 2.8. ***(i)*** *If $m(\underline{w}) = 0$, then either there is no relevant ball and the answer is that there is no majority, or there is one relevant ball and that is the majority ball.*

(ii) *If we obtain $\underline{w}'$ from $\underline{w}$ by any answer to a query (a, b) such that b is non-relevant, then $m(\underline{w}') = m(\underline{w})$.*

(iii) *If we increase the weight of a relevant ball, it remains relevant. In other words, if we obtain $\underline{w}'$ from $\underline{w}$ by replacing one ball w_i with w_i' such that $w_i' > w_i$, and w_i is relevant in $\underline{w}$, then w_i' is relevant in $\underline{w}'$.*

(iv) *For any $\underline{w}$ there is a threshold $t > 0$ such that w_i is relevant if and only if $w_i > t$.*

(v) *If a ball x is relevant before a query Q not containing x, there is at least one answer to Q such that x is still relevant afterwards. If a ball x is relevant before a query (x, y), then after the answer SAME the resulting ball with weight $x + y$ is relevant.*

(vi) *For any query (a, b) there is an answer such that the number of relevant balls decreases by at most two.*

Proof. To prove **(i)**, observe that if we color all the balls blue, there is a majority, unless all the balls have zero weight, in which case there is no relevant ball. Thus if there is no majority in any coloring of $\underline{w}$, then there cannot be relevant balls. If there is a majority ball a, then it has to be the only relevant ball. Indeed, if b is relevant, then changing the color of b must change the answer, unless b was the answer.

To prove **(ii)**, let c be the ball that the answer to the query (a, b) gives, thus it has weight either $a + b$ or $a - b$. Any algorithm that gives a solution for $\underline{w}$ gives a solution for $\underline{w}'$ and vice versa, where asking a ball that contains a is replaced by the query that contains c, and ignoring queries that involve b. Therefore, $m(\underline{w}') = m(\underline{w})$.

To prove **(iii)**, assume for a contradiction that w_i' is not relevant and take a coloring of the other balls that shows this. But then taking the same coloring for the other balls also shows that w_i is not relevant, a contradiction.

To prove **(iv)**, observe first that it is equivalent to the statement that a non-relevant ball cannot have larger or equal weight than a relevant ball. Indeed, this is obviously implied by **(iv)**, and if this holds, than t can be chosen as the largest weight of a non-relevant ball. Now assume a is relevant and $w(b) \geq w(a)$. Then consider the coloring that shows a is relevant, and replace b with a. This coloring clearly shows b must be also relevant.

To prove **(v)**, assume x is not in the query and consider a coloring of the other balls such that the color of x decides the majority. In that coloring the balls in the query have different or same color; answer accordingly. Then the same coloring shows c is still relevant.

Assume now x is in the query (x, y) and the answer is SAME. Consider a coloring of the other balls such that the color of x decides the majority. Taking the same coloring, the new ball $x + y$ will decide majority.

To prove **(vi)**, consider the relevant ball x not in the query with the smallest weight. By **(v)** there is an answer such that x remains relevant. As other relevant balls have larger weight, they also remain relevant, except for a and b (whose total weight can go below the weight of c). $\square$

We remark that **(v)** of the above proposition gives a new proof of a proposition from [9], which states that if all the n balls are relevant, we need at least $\lfloor n/2 \rfloor$ queries.

Proposition 2.9. *Before the last query of an optimal algorithm, there are either two relevant balls, they are of equal weight and there are no other balls with non-zero weight, or there are three relevant balls, and any query that compares two of them to finishes the algorithm.*

Proof. Proposition 2.8 implies that there are at most three relevant balls. Observe that if there are two relevant balls a and b in $\underline{w}$, they must have the same weight. Indeed, if $w(a) > w(b)$ and the total weight m of the other balls is smaller than $w(a) - w(b)$, then a is the only relevant ball. If $m \geq w(a) - w(b)$, then color b red, a blue and go through the other balls in increasing order of their weight, without the last ball c. We give each of them the color which has the smaller weight at that point. The first ball gets the color red, but as $m \geq w(a) - w(b)$, at one point the total weight of red balls becomes at least the total weight of blue balls. From that point, the difference between the classes is at most the weight of the current ball, which is at most $w(c)$. This coloring shows c is relevant.

It is left to show that if there are exactly three relevant balls, a, b and c, querying any two of them (say a and b) finishes the algorithm. Let m be the sum of the weights of the other balls. If $m = 0$, we are done unless both $a+b$ and $|a-b|$ are equal to c, which means $b = 0$, but a ball with zero weight cannot be relevant. Thus we can assume $m > 0$. We have $a \leq b + c + m$, otherwise we are done. But we also have $a + m \leq b + c$, because the other balls are not relevant. Moreover, if $a + m = b + c$, then again, the other balls would be relevant, thus we have $a + m < b + c$. This implies $c \geq a - b + m$, thus we are done if the answer is DIFF, as c is a majority ball. We also have $a + b + m \geq c$, otherwise we are done without the last query, and it implies $a + b \geq c + m$, moreover $a + b > c + m$, otherwise the remaining balls are relevant. Thus we are done if the answer is SAME. ☐

3 Graphs

Let us start this section with describing in detail how the weighted majority problems are connected to the majority problem on graphs. Consider an algorithm solving the majority problem on a graph G. Let G^i be the subgraph of G formed by the first i queries. We call the vertex set of a connected component of G^i a *q-component*. Observe that knowing the answer to the first i queries, for every q-component U we know a partition of U into two monochromatic subset U_1 and U_2. Let the weight of U be $w^i(U) = \big||U_1| - |U_2|\big|$. If the $i+1$st query is (u, u') with $u \in U$ and $u' \in U'$, where U and U' are q-components in G^i, then U and U' are merged into a q-component with vertex set $U \cup U'$ in G^{i+1}. Moreover, its weight $w^{i+1}(U \cup U')$ is either $w^i(U) + w^i(U')$, or $|w^i(U) - w^i(U')|$, depending on the answer to the $i+1$st query (u, u'). For other q-components U'' we have $w^{i+1}(U'') = w^i(U'')$. Hence an algorithm that finishes solving the majority problem on G after the kth query also solves the weighted majority problem for the vector having the weights of the q-components of G^k as coordinates. However, this does not work in the other direction, as we do not have the restriction of the graph structure in the weighted problem. Thus we can only prove upper bounds for $m(G)$ this way.

We will omit i and simply talk about $w(U)$ instead of $w^i(U)$ because i will be always clear from the context. For a ball $u \in U$, let $w(u) := w(U)$. If a q-component X has weight zero, we say that X is *balanced*. Similarly to vectors, we say that a graph G on n vertices is *hard* if $m(G) = n - 1$ for even n and $m(G) = n - 2$ for odd n.

Proposition 3.1. *Every tree T on an even number n of vertices is hard, i.e., $m(T) = n - 1$.*

Proof. We show more: the adversary can pick in advance a coloring c of the vertices such that no matter what edge is missing from the queries, we cannot find out if there is a majority or not. All this coloring needs to satisfy is that we have $n/2$ blue balls, and if we remove any edge from T, both the resulting subtrees are unbalanced, i.e., the number of blue and red balls is not the same in them. Indeed, if a query is not asked from T, then the Adversary can either claim that the real coloring of the vertices is c and thus no majority vertex exists, or the coloring coincides with c on one component, but is exactly the flipped version of c on the other component and thus there is majority.

Equivalently, we want to find a balanced 2-coloring such that each edge of T cuts it into two non-balanced parts. We start with an arbitrary balanced coloring of T. If an edge connecting a red ball u and a blue ball v cuts T into two balanced parts, we can simply change the color of u to blue and the color of v to red. Observe that any other edge e cuts T into two parts such that u and v belong to the same part, hence it does not change whether e cuts T into balanced parts.

Let us assume now that u and v are both red, and the edge uv cuts T into two balanced parts A and A'. Then we change the color of every ball in A. We claim that for any edge e that cuts T into parts B and B', it does not change whether B and B' are balanced. Indeed, either B is completely inside A', in which case no color inside B is changed, or B contains A, in which case some colors have changed, but the number of blue balls turning red is the same as the number of red balls turning blue, as A is balanced. As B' is balanced if and only if B is balanced, B' is also unaffected. Now u and v has different colors, so we can again exchange their color.

Hence we obtained that we can decrease the number of edges that cut T into balanced parts. It is easy to see that the coloring remains balanced. After applying this operation finitely many times we obtain the desired coloring, finishing the proof. $\square$

Surprisingly, it is much harder to give a lower bound for trees on an odd number of vertices. For paths, for example, we have $m(P_n) = n - b(n)$ for all odd $n \leq 13$, while $m(P_{15}) = 12 = n - b(n) + 1 = n - 3$. (This we have verified with a computer program.) We conjecture that $n - 3$ might be a lower bound for all trees, but we can only prove the weaker bound $n - 65$.

To prove the lower bound of $n - 65$ for odd n, we start with a lemma that gives another proof for Proposition 3.1. First, we introduce a notation. In a graph G, for a subset of its vertices $X \subset V$ we denote by $\delta(X)$ the *parity* of the number of edges between X and $V \setminus X$. If G is a tree and X is a connected subset of vertices, then $\delta(X)$ equals the parity of the number of components of $V \setminus X$.

Lemma 3.2. *We can answer to queries in any graph G such that for the weight $w(X)$ of any q-component $X \subsetneq V$ we have $0 \leq w(X) \leq 2$, and*
(i) *if $|X|$ is odd, $w(X) = 1$, and*
(ii) *if $|X|$ is even, $w(X) = 2\delta(X)$.*

Note that if T is a tree on an even number of vertices, then $0 \leq w(X) \leq 2$ implies that the game goes on until we have at most one non-balanced q-component. Assume at that point there would also be some balanced q-components. Observe that there is a tree-structure on the components of a tree. Then at least one of the balanced q-components would be a leaf-component, but that contradicts condition **(ii)**. Thus there can be only one component, which implies Proposition 3.1.

For trees on an odd number of vertices, a similar argument cannot work, as for example in P_n, it can happen that the first two vertices form a q-component of weight 2, followed by $(n-3)/2$ pairs of vertices that each form a balanced q-component, and the last vertex is a q-component of weight 1. In this case we have solved the majority problem with only $(n-1)/2$ queries. In fact, according to the conditions of Lemma 3.2, our algorithm would answer exactly so that it would produce such weights for the q-components. For paths, there is no way to keep the weight function bounded without allowing an arbitrarily number of adjacent balanced q-components; but if this happened, then we could merge all the q-components to their left, and all the q-components to their right, so that only two non-balanced q-components remain - after this we are done if n is odd, saving an arbitrarily large number of queries. This is why the proof will be more complicated for trees on an odd number of vertices; we will need to use our results about weighted balls.

Proof of Lemma 3.2. Initially the conditions are satisfied. Suppose that the query is between two q-components, X and Y.

If $|X| + |Y|$ is odd, then exactly one of $w(X)$ and $w(Y)$ equals 1, while the other equals 0 or 2, so we can achieve $w(X \cup Y) = 1$ to satisfy condition **(i)**.

If $|X|$ and $|Y|$ are both odd, then we can choose the weight of $X \cup Y$ to be 0 or 2; one of those is equal to $2\delta(X)$.

If $|X|$ and $|Y|$ are both even, then since $\delta(X \cup Y) = \delta(X) + \delta(Y) - 2|E(X,Y)| = \delta(X) + \delta(Y) \bmod 2$, we have $w(X \cup Y) = w(X) + w(Y) \bmod 4$. Observe that $w(X) + w(Y)$ is 0, 2 or 4. Thus we can answer so that $w(X \cup Y)$ becomes 0, 2 or 0, respectively, to satisfy condition **(ii)**. □

For the lower bound of $n - 65$ for trees on an odd number of vertices, we need another ingredient. Before that, we prove a simpler result that contains an important ingredient of the proof, and is of independent interest.

Theorem 3.3. *Let $n = 2^k + l$, where $l < 2^k$. If G has a set U of vertices such that $|U| \leq 2^{k-2}$ and the components of $G \setminus U$ are single vertices (i.e., every edge is incident to a vertex in U), then G is hard, i.e., $m(G) = n - 1$ if n is even and $m(G) = n - 2$ if n is odd.*

Proof. Denoting by $w(X)$ the weight of a q-component X, we initially have $\sum_X w(X) = n$. Adversary will maintain in the first part of the algorithm that $w(X) \neq 0$ for every q-component X.

Let us now describe a strategy of the Adversary for the first part of the algorithm. Whenever we compare some $v \in G \setminus U$ with a $u \in U$ such that $w(u) \geq 2$, the answer is such that the weight of the new q-component is $w(\{u, v\}) = w(u) - 1$, thus $\sum_X w(X)$ decreases by 2. In every other case the answer is such that the weights are added up, i.e., $\sum_X w(X)$ remains the same.

Introduce the potential function $\Psi = \sum_X w(X) + |\{X \mid X \cap U \neq \emptyset,\, w(X) = 1\}|$. The Adversary's strategy is such that every time we compare some $v \in G \setminus U$ with a $u \in U$, the function Ψ decreases by at least 1. Since initially $\Psi = n + |U|$, after $|U| + l$ queries involving some vertex of $V \setminus U$, we would have $2^k \geq \Psi \geq \sum_X w(X)$. But Adversary stops executing this algorithm the moment we have $\sum_X w(X) = 2^k$ or $\sum_X w(X) = 2^k + 1$; this surely happens, as $\sum_X w(X)$ can only decrease by 2.

Let us consider the vertices from $G \setminus U$ that were merged into some q-components (i.e. those that appeared in queries). Let x denote the number of those where the total weight did not decrease when they first appeared in a query, and y denote the number of those where the total weight did not decrease when they first appeared in a query. Then we have $x \leq y + |U|$. Indeed, consider a q-component containing a vertex $u \in U$, and observe that whenever the weight of this component increased by merging it with a vertex from $G \setminus U$, the next time its weight decreased.

This implies that at the point where Adversary stops executing the algorithm, the number of vertices in $G \setminus U$ that have not appeared in any query is at least $n - |U| - (|U| + l) \geq 2^{k-1}$. We are done by applying Lemma 2.2 to the current q-components as weighted balls. (So even if we could compare any two vertices from now, we still could not solve the majority problem with less queries.) □

Now we present the result that we use for the lower bound. The proof of Theorem 3.4 is based on a similar idea as that of Theorem 3.3, but is more involved.[4]

Theorem 3.4. *Let $n = 2^k + l$, where $l < 2^k$ and n is odd. If G has a set U of vertices such that $|U| \leq 2^{k-3} - 1$, each component T of $G \setminus U$ is a tree that has only one vertex $r(T)$ that is adjacent to some vertices from U, and every path from $r(T)$ to another vertex of T has at most p vertices, then $m(G) \geq n - 1 - 2p$.*

Proof. We start as in the proof of Theorem 3.3. We denote by $w(X)$ the weight of a q-component X, and for a vertex u of G, $w(u)$ denotes the weight of the q-component containing u. We initially

[4]Instead of the potential function argument, the proof of Theorem 3.3 could also be finished in the same way as the proof of Theorem 3.4.

have $\sum_X w(X) = n$. The Adversary will maintain in the first part of the algorithm that $w(X) \neq 0$ for every q-component X that intersects U.

Whenever two elements inside some component T of $G \setminus U$ are queried, Adversary can answer according to Lemma 3.2 (applied only to T). This way the weight of any $X \subset G \setminus U$ will be at most 2. The crucial property is that the balanced q-components of T will always separate some positive weight part of T from $r(T)$. This way they are "in the way" to compare these parts with the rest of the graph, so they cannot be simply ignored. The strategy of the Adversary will be to make sure that the game cannot end while there are many unbalanced q-components. After there are only few unbalanced q-components the game might end, but in this case we will show that the graph could be made into a single q-component by adding $2p$ further edges to it. This shows that at most these many queries can be saved.

Also, in case we merge all of T into one q-component, Adversary would like to avoid $w(T) = 0$. This cannot happen if T has an odd number of vertices; if T has an even number of vertices, Adversary adds an (imaginary) extra degree one vertex $r'(T)$ to T that is adjacent only to $r(T)$, to obtain T', and apply Lemma 3.2 to T'. Since $r'(T)$ is never compared with anything, merging all of T into a q-component cannot give $w(T) = 0$, because $T' \setminus T$ has only one edge, $\{r'(T)\}$. Therefore, in case the whole tree T is merged, we get $w(T) = 2$.

Whenever we compare some $v \in G \setminus U$ with a $u \in U$ such that $w(u) \geq 3$, Adversary answers such that the weight of the new q-component is $w(uv) = w(u) - w(v)$, thus $\sum_X w(X)$ decreases by $2w(v) \leq 4$. In every other case Adversary answers so that the weights are added up, i.e., $\sum_X w(X)$ remains the same. This way the weight of a q-component can never exceed 4, unless we merge two q-components that both intersect U. Because of this, we can conclude that $\sum_{X: X \cap U \neq \emptyset} w(X) \leq 4|U| \leq 2^{k-1}$.

Adversary stops executing this algorithm the moment we have $\sum_X w(X) = 2^k + 1$ or $2^k + 3$; this surely happens, as $\sum_X w(X)$ is odd and can decrease by at most 4. We would also like to reduce the less pleasant case of $\sum_X w(X) = 2^k + 3$ to the former one. If there is an X with $w(X) = 1$, then Adversary can just reveal that this X differs from another to decrease $\sum_X w(X)$ by 2. But this might not be the case. Because of this, we slightly modify the strategy of the Adversary described earlier as follows.

We keep track of the number of q-components that are disjoint from U, and whose weight is exactly 1. Notice that this number cannot increase and can decrease by at most two with each query. If this number ever reaches 1 or 2, then we look at $\sum_X w(X) \bmod 4$. If $\sum_X w(X) \equiv 3 \bmod 4$, then Adversary reveals that one of the q-components disjoint from U whose weight is exactly 1 is red and some other ball is blue to have $\sum_X w(X) \equiv 1 \bmod 4$. (We pick this other ball from a q-components that has weight 2 (there are many such that among the ones disjoint from U) to make sure we do not create additional balanced q-components.) Moreover, from now on when we compare one of the q-components disjoint from U with weight 1 to some $u \in U$, Adversary will answer that their color is the same to keep $\sum_X w(X) \bmod 4 \equiv 1$ (even if $w(u) \geq 3$). As this happens at most twice, we will have now the bound $\sum_{X: X \cap U \neq \emptyset} w(X) \leq 4|U| + 2 \leq 2^{k-1}$, and can run the algorithm until we get $\sum_X w(X) = 2^k + 1$.

At this point, we can apply Lemma 2.6 to obtain that after we know a majority vertex, we have at most two non-balanced q-components. Moreover, these two q-components need to cover U, as the weights of sets intersecting U stays positive throughout the algorithm. We want to argue that by now almost all the edges have been asked. It is enough to show that these two remaining q-components can be connected by at most p edges.

The only difficulty is that (at most) one of these q-components might be a part of a tree component of $G \setminus U$ that is separated by balanced q-components from the other q-component (that intersects U). But even this can be connected to U with at most p edges (connecting an appropriate vertex of U through some $r(T)$ to another vertex of T) because of the last condition in Theorem 3.4. We also need to count p edges for the vertex that was (possibly) revealed during the algorithm.

To summarize, instead of asking all $n-1$ edges, we might save $2p$, because at the end there might be three q-components, one from U, one that we need not query at the end, and possibly one that was revealed during the algorithm. $\square$

Remark. We could get a better constant by considering, as a parameter instead of p, the maximum possible number of those edges from $r(T)$ to another vertex of T that are part of a path going through only balanced q-components, but are not contained in the balanced q-components. This would give something like $n-p$ as a lower bound. One could probably even obtain about $n-p/2$ as a lower bound if the case $\sum_X w(X) \equiv 3 \bmod 4$ is dealt with using a better trick.

To finish the proof for trees on an odd number of vertices, we need the following folklore generalization of the concept of centroid for trees, known as *centroid decomposition.*

Proposition 3.5. *In every tree on n vertices there is a subset of at most $2n/p$ vertices U such that every component of $G \setminus U$ has at most p vertices.*

By picking $p = 32$, we obtain a U satisfying the conditions of Theorem 3.4 with these parameters. This finishes the proof of Theorem 1.1.

3.1 Non-deterministic complexity of odd trees

We have seen that it is much harder to prove the lower bound $m(T) \geq n - 65$ for trees of odd order n than the lower bound $m(T) \geq n-1$ for trees of even order n. Somewhat even more surprisingly, there is a significant difference between the so-called non-deterministic complexities of trees of even and odd order. The non-deterministic complexity $m_{nd}(G)$ of a graph G is defined as the minimum number of queries needed to find a majority vertex in the worst case, provided we know the color of each vertex beforehand from an unreliable source and we just have to verify (some of) this information. Let us observe that in the proof of Proposition 3.1 we actually showed $m_{nd}(T) = n-1$ for any tree T of even order n.

Proposition 3.6. *Let P be a path of order n, such that n is odd. Then $m_{nd}(P) = n - \Theta(\sqrt{n})$.*

Proof. Let us denote the i^{th} vertex of P by x_i for $i = 1, 2, \ldots, n$. For the lower bound let us suppose that $n = k^2 + 1$ for some even k (this is possible, since we are only interested in the order of magnitude of $n - m_{nd}(P)$) and let us call the batch of vertices $x_{(i-1)k+1}, x_{(i-1)k+2}, \ldots, x_{ik}$ Batch i for $i = 1, 2, \ldots, k$. Now let us color the vertices of Batch i red if and only if i is odd and let x_{k^2+1} be blue (just like the vertices of Batch k, since k is even). We claim that in order to find a majority vertex, one needs at least $n - 2k - 1$ comparisons (that is, one has to verify the result of this many comparisons). Assume to the contrary that fewer comparisons suffice. Then the number of edges not asked as queries is $p \geq 2k$, hence the number of q-components after the last query is $p + 1 \geq 2k + 1$. It is easy to see that the number of balanced q-components is at most $k - 1$, since a balanced q-component must contain both x_{ik} and x_{ik+1} for some $i < k$. Thus at least $k + 2$ of the q-components are unbalanced. It is also easy to see that the weight of any q-component is at most $k + 1$ (since k vertices of the same color are always followed and preceded by k vertices of the opposite color, except for the first k and last $k + 1$ vertices). Now if one could show a majority vertex, the weight of its q-component should be more than the sum of the weights of the other q-components, which is impossible, since the latter sum is at least $k + 1$. This finishes the proof of the lower bound.

Next we prove the upper bound. Consider any coloring of the vertices of P and let us denote the number of red (resp. blue) vertices among $\{x_1, x_2, \ldots, x_i\}$ by $R(i)$ (resp. $B(i)$) and let us suppose without loss of generality that $d := R(n) - B(n) > 0$ (notice that n is odd). Observe that if $d \geq \sqrt{n}$, then by asking the first $n - \lceil \frac{d}{2} \rceil$ edges (or any consecutive $n - \lceil \frac{d}{2} \rceil$ edges) of P, we obtain a q-component of weight at least $\lceil \frac{d}{2} \rceil + 1$ and $\lceil \frac{d}{2} \rceil - 1$ q-components of weight (and also cardinality) 1. Thus any majority vertex of the large q-component is also a majority vertex of the whole graph, and we are done. Therefore, we might assume that $d < \sqrt{n}$.

Let $D(i) := R(i) - B(i)$ (so $d = D(n)$), $\Delta := \max_{i=1}^{n} |D(i)|$, and let j be the smallest number, such that $|D(j)| = \Delta$. Since $d > 0$, we may suppose that $D(j) = \Delta$, otherwise we can reverse the order of the vertices and obtain a situation, where the similarly obtained $D(j')$ is positive (the value Δ would be different then, but d remains the same). Now we consider two cases, based on the value of Δ.

Case 1. $\Delta < 2\sqrt{n}$. Since all values $D(i)$ are in the interval $[-\Delta, \Delta]$, by the pigeonhole principle there must be a value v, for which $|\{i : D(i) = v\}| \geq \frac{n}{4\sqrt{n}+1} > \frac{\sqrt{n}}{5}$. It is obvious that if $D(a) = D(b)$, then the number of red and blue vertices are the same in the subpath between the vertices x_{a+1} and x_b. Let the elements of $\{i : D(i) = v\}$ be $i_1, i_2, \ldots, i_r$ and let us query all edges of P, except the edges $(x_{i_1}, x_{i_1+1}), (x_{i_2}, x_{i_2+1}), \ldots, (x_{i_r}, x_{i_r+1})$. In this way we obtain r or $r + 1$ q-components, of which $r - 1$ are balanced, therefore any majority vertex of the larger non-balanced q-component (which exists, since n is odd) is a majority vertex of the whole graph as well. Since $r > \frac{\sqrt{n}}{5}$, we are done with this case.

Case 2. $\Delta \geq 2\sqrt{n}$. Recall that j is the smallest number, such that $D(j) = \Delta$ and $d < \sqrt{n}$, i.e., the number of red vertices is Δ more than the number of blue vertices by x_j, but then the difference drops to d by the end of P. Thus there must exist a smallest number k, such that $k > j$ and $D(k) < \sqrt{n}$. Then the subpath P' between the vertices x_{j+1} and x_k contains at least $\Delta - \sqrt{n} \geq \sqrt{n}$ more blue vertices, than red vertices. Let now j_1 be the smallest index, such that the subpath of P' between x_{j+1} and x_{j_1} contains exactly one more of the blue vertices than the red vertices. Similarly, let j_2 be the smallest index, such that the subpath of P' between x_{j_1+1} and x_{j_2} contains one more of blue vertices than red vertices, and so on. It is clear that the indices $j_1, j_2, \dots, j_{\lfloor\sqrt{n}\rfloor}$ are well-defined. Now let us query all edges of P, except the edges $(x_j, x_{j+1}), (x_{j_1}, x_{j_1+1}), \dots, (x_{j_{\lfloor\sqrt{n}\rfloor}}, x_{j_{\lfloor\sqrt{n}\rfloor+1}})$. In this way we obtain $\lfloor\sqrt{n}\rfloor + 2$ q-components, such that one of them has weight Δ, $\lfloor\sqrt{n}\rfloor$ of them has weight 1, and one of them has weight smaller than $\lfloor\sqrt{n}\rfloor$, thus any majority vertex of the q-component of weight Δ is also a majority vertex of the whole graph, finishing the proof of Proposition 3.6. $\square$

3.2 Optimal graph with few edges

In this subsection we prove Theorem 1.2 which states that for every n there is a graph G with n vertices and $O(nb(n))$ edges such that $m(G) = n - b(n)$.

Proof of Theorem 1.2. Let F_k be the graph obtained from a path $v_1 v_2 \dots v_k$ by adding k degree 1 vertices, $u_1, u_2, \dots, u_k$, to it such that u_i is connected to v_i for each $1 \leq i \leq k$. Let $k = \lfloor n/2 \rfloor$ and G be the graph we obtain if we connect $v_{k-b(n)+1}, \dots, v_k$ to every other v_i.[5]

First we define an algorithm A_l for $l = 2^i$. It uses the edges of F_l and either gets back a DIFF answer at some point for an edge that connects two monochromatic components of the same size, or shows that F_l is monochromatic. Moreover, at any point it uses only the first j vertices of the path $v_1 \dots v_j$ and the leaves $u_1, \dots, u_j$ connected to them, for some j.

We define algorithm A_l recursively. Assume we have defined Algorithm A_l and we are given F_{2l}. The graph F_{2l} consists of two copies of F_l and an additional edge, where the first copy has vertices $v_1, \dots, v_l, u_1, \dots, u_l$, the second copy has the remaining vertices, and the additional edge is $v_l v_{l+1}$. Algorithm A_{l+1} runs algorithm A_l separately for the first, and then for the second copy of F_l, and finally asks (v_l, v_{l+1}). If for either copy of F_l we get back a DIFF answer at some point for an edge that connects two monochromatic components of the same size, we are done. Otherwise, both copies of F_l are monochromatic. In this case a DIFF answer to the last query connects two monochromatic components of the same size, while a SAME answer shows F_{2l} is monochromatic.

The algorithm showing $m(G) = n - b(n)$ for even n is as follows. It goes similarly to the algorithm showing $m(K_n) = n - b(n)$: we ask queries such that if the answer is DIFF, we obtain a balanced component (that we can discard), and otherwise we build larger and larger monochromatic components where the number of vertices is a power of 2. In the following algorithm, whenever the answer is DIFF, we stop.

[5]The similar construction would also work if we took the union of a path on n vertices with the biclique $K_{n-b(n),b(n)}$ but we have found our proof to be easier to present for the above graph G.

We start Step 1 with the query (v_k, u_k). Observe that these two vertices can be considered as the first vertices of a copy of F_{2^i} having the vertices $v_1, \dots, v_{2^{i-1}-1}, u_1, \dots, u_{2^{i-1}-1}$, for every $i \leq l = \lfloor \log n \rfloor$. Thus we run algorithm A_l. If we obtain a monochromatic component of size 2^l, that is a majority component, and we are done. If we obtain a DIFF answer, we continue with Step 2, which starts with asking $v_{k-1}u_{k-1}$. Observe that for any j, any consecutive part of F_k together with u_k and v_k forms a copy of F_l for some l. More precisely, there is a copy of F_l on the vertices $v_j, \dots, v_{j+l-2}, u_j, \dots, u_{j+l-2}, v_{k-1}, u_{k-1}$, provided $j + l - 2 < k - 1$. We continue with the vertex v_j if v_{j-1} has the largest index among those used in Step 1. Similarly to Step 1, we run algorithm A_l for the largest l possible, i.e., the largest l that is a power of 2 and is smaller than $k - j + 1$.

In general, for step i we take v_{k-i+1}, the first vertex v_j not used in Step $i - 1$, and $2^r - 2$ consecutive vertices next to it for the largest r possible without arriving back to v_{k-i+1}. Furthermore, we take u_m for every v_m we took. More precisely, we take $v_j, \dots, v_{j+2^r-2}, u_j, \dots, u_{j+2^r-2}$, v_{k-i+1} and u_{k-i+1}. This is a copy of F_{2^r}, thus we can run Algorithm A_{2^r} on it. If we obtain a monochromatic component, that is of the majority color, and we are done. Indeed, we took the largest r possible, thus more than half of the remaining vertices, and the components obtained in earlier steps are balanced.

Observe that if all the steps end with DIFF answers, we had at least $b(n)$ steps before using up all the vertices. Indeed, all the components have order that is a power of 2, and n cannot be written as the sum of less than $b(n)$ powers of 2. After Step $b(n)$, the remaining vertices (if there are any) form a connected graph, thus we can ask a spanning tree of that graph to find a majority vertex there. Altogether we have a forest of at least $b(n)$ components, thus asked at most $n - b(n)$ queries. $\square$

4 Questions

We collect below the most important questions that remain open.

- What is the complexity of computing $m(\underline{w})$ and $m(G)$?
- For which $\underline{w}' = (2w_1, w_3, w_4, \dots, w_k)$ does $m(\underline{w}') = m(\underline{w}) - 1$ hold?
- Does $m(T) \geq n - 3$ hold for every tree or path on n vertices?
- What is the least number of edges a graph on n vertices can have if $m(G) = n - b(n)$?

References

[1] M. Aigner, Variants of the majority problem, Disc. App. Math., 137 (2004), 3–25.

[2] M. Aigner, G. De Marco, M. Montangero, The plurality problem with three colors and more, Theor. Comp. Sci., 337 (2005), 319–330.

[3] A. M. Borzyszkowski, Computing majority via multiple queries, Theor. Comp. Sci., 539 (2014), 106–111.

[4] H. Chang, D. Gerbner, B. Patkós, Finding non-minority balls with majority and plurality queries, preprint, arXiv:1812.08850.

[5] F. Chung, R. Graham, J. Mao, A. Yao, Oblivious and Adaptive Strategies for the Majority and Plurality Problems, Computing and combinatorics, LNCS 3595, Springer, Berlin (2005), 329-338.

[6] G. De Marco, E. Kranakis, G. Wiener, Computing Majority with Triple Queries, Theor. Comp. Sci., 461 (2012), 17–26.

[7] D. Eppstein, D. S. Hirschberg, From Discrepancy to Majority, Algorithmica, 80 (2018), 1278-1297.

[8] M.J. Fisher, S.L. Salzberg, Finding a Majority Among n Votes, J. Algorithms, 3 (1982) 375–379.

[9] D. Gerbner, G. O.H. Katona, D. Pálvölgyi, B. Patkós, Majority and plurality problems, Disc. App. Math., 161 (2013), 813–818.

[10] D. Gerbner, B. Keszegh, D. Pálvölgyi, B. Patkós, M. Vizer, G. Wiener, Finding a non-minority ball with majority answers, Disc. App. Math., 219 (2017), 18–31.

[11] D. Gerbner, D. Lenger, M. Vizer, A plurality problem with three colors and query size three, preprint, arXiv:1708.05864.

[12] D. Gerbner, M. Vizer, Majority problems of large query size, Disc. App. Math., 254 (2019), 124–134.

[13] T.P. Hayes, S. Kutin, D. van Melkebeek, On the Quantum Black-Box Complexity of Majority, Algorithmica 34 (2002) 480–501.

[14] R. Rivest, J. Vuillemin, On recognizing graph properties from adjacency matrices, Theor. Comp. Sci., 3 (1976) 371–384.

[15] M. E. Saks, M. Werman, On computing majority by comparisons, Combinatorica 11 (1991), 383–387.

[16] G. Wiener, Search for a majority element, J. Stat. Plann. Inf., 100 (2002) 313–318.