Stabilization Time in Minority Processes

Pál András Papp
ETH Zürich, Switzerland
apapp@ethz.ch

Roger Wattenhofer
ETH Zürich, Switzerland
wattenhofer@ethz.ch

Abstract

We analyze the stabilization time of minority processes in graphs. A minority process is a dynamically changing coloring, where each node repeatedly changes its color to the color which is least frequent in its neighborhood. First, we present a simple $\Omega(n^2)$ stabilization time lower bound in the sequential adversarial model. Our main contribution is a graph construction which proves a $\Omega(n^{2-\epsilon})$ stabilization time lower bound for any $\epsilon > 0$. This lower bound holds even if the order of nodes is chosen benevolently, not only in the sequential model, but also in any reasonable concurrent model of the process.

2012 ACM Subject Classification Mathematics of computing → Graph coloring; Theory of computation → Self-organization; Theory of computation → Distributed computing models

Keywords and phrases Minority process, Benevolent model

Digital Object Identifier 10.4230/LIPIcs.ISAAC.2019.46

Related Version An archive version is available at `https://arxiv.org/abs/1907.02131`.

© Pál András Papp and Roger Wattenhofer;
licensed under Creative Commons License CC-BY

1 Introduction

If you google "bad wifi", one advice you will get for sure is to choose the least crowded frequency in order to minimize interference with your neighbors. Unfortunately, this least crowded frequency may change again if some of your neighbors do the same.

Frequency allocation is a familiar example of *minority processes* in graphs: given a graph, a set of colors, and an initial coloring of the nodes with these colors, a minority process is a process where each node, when given the chance to act, modifies its color to a color that has the smallest number of occurrences in its neighborhood. This results in a dynamically changing coloring, which is essentially a form of distributed automata. Minority processes arise in various fields of economics [12] or social science [3] when players are motivated to differentiate from each other, but they also emerge in cellular biology [4] or crystallization mechanics [2].

A minority process is said to stabilize when no node has an incentive to change its color anymore. The aim of the paper is to understand how long it takes until such a minority process reaches a stable state. We study the process in several different models, some of them sequential, some concurrent. In sequential models, when only one node at a time can change its color, stabilization time depends on the choice of the order of nodes. Hence, the model can further be subdivided into three cases, depending on whether the order of acting nodes is specified benevolently (trying to minimize stabilization time), adversarially (trying to maximize stabilization time), or randomly.

On the other hand, in concurrent models, multiple nodes are allowed to switch their color at the same time. However, if two (or more) neighboring nodes continuously keep on forcing each other to switch their color, the system may never stabilize. The simplest such example is a graph of two connected nodes that have the same initial color, and keep on switching to the same new color in every step. We also study concurrent models that exclude this behavior, as it is unrealistic in many application areas where neighbors are unlikely to switch at the exact same time.

In any model where simultaneous neighboring switches are excluded, it is easy to prove a $O(n^2)$ upper bound on stabilization time for minority processes. Initially, some (maybe even all) of the at most $O(n^2)$ edges in the graph are monochromatic (i.e., they have a *conflict*). When a node switches its color to the minority color in its neighborhood (but its neighbors do not change color in the same step), then the number of conflicts on the adjacent edges strictly decrease. Since the original number of conflicts is $O(n^2)$ and the overall number of conflicts decreases by 1 at least in each step, the number of steps is limited to $O(n^2)$.

However, this raises a natural question: are there example graphs that exhibit this naive upper bound? Or is there a significantly lower (e.g. linear) upper bound on stabilization time in some models? While these questions are already answered for the "dual" problem of majority processes (when nodes switch to the most frequent color in their neighborhood), for the case of minority processes, they have remained open so far.

The main contributions of the paper are constructions that prove lower bounds on stabilization time of minority processes. As a warm-up, we present a simple example in Section 4 which shows that in the sequential adversarial model, stabilization may take $\Theta(n^2)$ steps. Our main result is a construction proving that stabilization can also take superlinear time in the sequential benevolent case. We first present a graph and an initial coloring in Section 5 where any selectable sequence lasts for $\Omega(n^{3/2})$ steps. Then in Section 6, we outline how a recursive application of this technique leads to a stabilization time of $\Omega(n^{2-\epsilon})$ for any $\epsilon > 0$, almost matching the upper bound of $O(n^2)$. This is an interesting contrast

to majority processes, where stabilization time is bounded by $O(n)$ in the benevolent case. Furthermore, our construction shows that this almost-quadratic lower bound holds not only in the sequential model, but also in any reasonable concurrent setting.

2 Related work

While there is a wide variety of results on both minority and majority processes, majority processes have been studied much more extensively. Recently, [5] has shown that stabilization time in majority processes can be superlinear both in the synchronous model, and in the sequential model if the order is chosen by an adversary. However, [5] has also shown that stabilization always happens in $O(n)$ time in the sequential benevolent model. In case of majority processes in weighted graphs, a $2^{\Theta(n)}$ lower bound on stabilization time was also shown in [11].

Other aspects of majority processes have also been studied thoroughly, especially in the synchronous model. Results on majority processes include basic properties [8], their behavior on random graphs [6], complexity results on determining stabilization time [10], minimal sets of nodes that dominate the process [7], and the existence of stable states in the process [1].

In contrast to this, the dynamics of minority processes has received less attention. The stabilization of minority processes has only been studied in special classes of graphs, including tori, cycles, trees and cliques [14, 15, 16]. These studies are mostly conducted only in the synchronous or the sequential random model. More importantly, these results study a different variant of the minority process, which considers the closed neighborhood of nodes, and thus can result in significantly larger (possibly exponential) stabilization time, even in the unweighted case. An experimental study of the processes on grids is also available in [14].

In weighted graphs, it has recently been shown in [13] that stabilization of minority processes can take $2^{\Theta(n)}$ steps in various models, matching a straightforward exponential upper bound in the weighted case. However, the constructions of [13] use exponentially large node or edge weights to obtain these results; as such, the same techniques are not applicable in the unweighted case.

Besides these studies on the dynamics of the process, there are also numerous theoretical results on stable states in minority processes. These include complexity results on deciding the existence of different stable state variants [12], characterization of infinite graphs with a stable state [17], and analysis of price of anarchy in such states as local minima [12]. In the work of [9], it is also shown that slightly modified minority processes, based on distance-2 neighborhood of nodes, can provide better local minima at the cost of larger (but still polynomial) stabilization time.

However, in contrast to majority processes, the stabilization time of minority processes in general unweighted graphs has remained unresolved so far.

3 Definitions and background

3.1 Models

In the paper, we primarily focus on the following models:

A. Sequential adversarial: In every step, only one node switches. The order of nodes is specified by an adversary who maximizes stabilization time.

B. Sequential benevolent: In every step, only one node switches. The order is specified by a benevolent player who minimizes stabilization time.

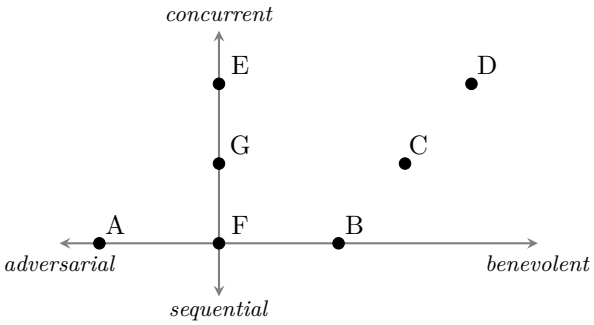

Figure 1 Properties of the listed models

C. Independent benevolent: In every step, the benevolent player is allowed to choose any independent set of switchable nodes, and switch them simultaneously.

D. Free benevolent: In each step, the benevolent player is allowed to choose any set of switchable nodes, and switch them simultaneously.

However, our lower bounds extend to a range of other popular models:

E. Concurrent synchronous: In every step, all switchable nodes switch simultaneously.

F. Sequential random: In every step, only one node switches, chosen uniformly at random among the switchable nodes.

G. Concurrent random: In every step, every switchable node switches with probability p, independently from other nodes.

An intuitive illustration of these models is shown in Figure 1. The vertical axis shows how concurrent a model is, the horizontal shows how wide is the set of opportunities it grants the player to speed up / slow down stabilization. In the case of majority processes, models A and E are shown to take superlinear time to stabilize for some graphs, but model B always stabilizes in linear time [5]. However, we prove that for minority processes, even model B can take superlinear time. Models C and D grant even wider sets of possible (concurrent) moves for the benevolent player, which may drastically reduce the number of steps in some cases; however, we show that the same lower bound holds even if such moves are available.

Note that models A, B, C and F exhibit a natural $O(n^2)$ upper bound on stabilization time, as the overall number of conflicts decreases in each step by at least 1. On the other hand, models D, E or G may allow neighboring nodes to switch at the same time, and thus in these models, some nodes may keep on endlessly changing colors. However, our constructions specifically ensure that connected nodes are never switchable at the same time, and thus for these particular graphs, the process stabilizes in any of the models.

Through most of the analysis in the paper, we focus on the sequential models. We first show a simple construction with $\Theta(n^2)$ stabilization time in model A. We then present a more complex construction to first show $\Omega(n^{3/2})$, and then $\Omega(n^{2-\epsilon})$ stabilization time in model B. It then follows from a few observations that these latter constructions also have the same stabilization time in models C and D. Since model D provides the widest set of opportunities from all models, this implies the same lower bound for each of the listed models.

3.2 Preliminaries

Throughout the paper, we consider simple, unweighted, undirected graphs. Graphs are denoted by G, their number of nodes by n, and the maximum degree in the graph by Δ.

Given a graph G on the vertex set V, an *independent set* is a subset of V such that no two nodes in this subset are connected. A *coloring* of the graph with k colors is the assignment of one of the colors (numbers) from $\{1, 2, ..., k\}$ to each of the nodes. If two nodes share an edge and are assigned the same color, then the nodes have a *conflict* on this edge.

Our process consists of discrete time steps (*states*), where we have a current coloring of the graph in every state. When a node v is currently colored c_1, but there exists a color c_2 such that the neighborhood of v contains strictly less nodes colored c_2 than nodes colored c_1, then the node is *switchable* (since the node could reduce its number of conflicts by changing its color). The process of v changing its color is *switching*. Nodes always make locally optimal solutions, that is, they switch to the color which is least frequent in their neighborhood. In case of multiple optimal colors, related work on majority processes considers different tie-breaking rules. However, our constructions ensure that a tie can never occur, and thus our bounds hold for any tie-breaking strategy.

The *minority process* is a sequence of steps, where each step is described by a set of nodes that switch. Note that we only consider valid steps, where every chosen node is switchable.

A state is *stable* when no node in the graph is switchable; a system *stabilizes* if it reaches a stable state. *Stabilization time* is the number of steps until the process stabilizes. Note that in case of model E, papers studying majority processes often use a different definition of stabilization, based on periodicity. However, our constructions ensure that the process always ends in a stable state, thus for the graphs in the paper, the two definitions of stabilization are equivalent.

In our examples, we will consider the case of having only two available colors, black and white. However, as discussed in Section 3.3, our lower bounds are easy to generalize to any number of colors.

The restriction to two colors allows us to introduce some helpful terminology. Consider a node v at a given state of the process. If v has v_s neighbors with the same color as v, and v_o neighbors with the opposite color, the number $v_o - v_s$ is called the *balance* of v. Note that if one of the neighbors of v switches, then the balance of v either increases or decreases by 2 (which shows that the parity of the balance of v can never change). The definition also implies that v is switchable if and only if its balance is negative. Switching v changes the sign of its balance.

3.3 General tools in the constructions

Groups. We use the notion *group* to refer to a set of nodes that have the same initial color and the exact same set of neighbors (hence, groups are independent sets). Groups are, in fact, only a tool to consider certain nodesets together as one entity for simpler presentation. They will be shown as only one node with double borders in the figures, with the size of the group indicated in brackets.

In the adversarial case, we will only consider sequences that switch groups together (i.e consecutively in any order). In the benevolent case, groups will be switched together in the sense that if a node in the group switches, then all other nodes in the group will also switch before any neighbor of the group becomes switchable; this property is enforced by the graph construction. The more complicated definition in the benevolent case is due to the fact that we have to consider every possible sequence that the player can choose. Technically, in some sequences, a group might not be switched consecutively (it might be interrupted by switches in other, distant parts of the graph), but the outcome will still be equivalent to switching them consecutively.

Fixed nodes. Given a graph G, let us add two more set of nodes F_w, F_b to the graph such that $|F_w| = |F_b| = n + 1$, and v_w and v_b are connected for all $v_w \in F_w$, $v_b \in F_b$. Let the color of F_w and F_b initially be white and black, respectively. The nodes in F_w and F_b will be referred to as *fixed nodes*, and we will connect them to some of the nodes in our original graph. Note that these fixed nodes already have $n + 1$ neighbors of the opposite color, and can never have more neighbors of the same color (as they can have at most n neighbors G), so their color is indeed fixed and they can never switch.

Such fixed nodes are widely used in our construction; we can allow any node in G to have up to $\Delta + 1$ fixed neighbors of either color. The introduction of fixed nodes increases the graph size only by a constant factor (to $3n + 2$), so all lower bounds expressed as a function of n will still be of the same magnitude as a function of $3n + 2$. Therefore, for ease of presentation, we still use n to denote the number of nodes in the graph *without* the extra fixed nodes, and express our bounds as a function of n.

Fixed node neighbors are denoted by squares in the figures, with the multiplicity written beside the square (if more than 1). We always draw separate squares for distinct nodes, even though the corresponding fixed node sets might overlap. This is because fix node connections are thought of as a "property" of the node, introducing an offset into its initial balance.

Generalization to more colors. While the paper discusses the case of two colors, a simple idea allows a generalization to any constant number of colors k. Assume we have a construction G on n nodes, showing a lower bound on stabilization time with two colors; we can simply add sets of nodes F_3, F_4, ..., F_k of size $\Delta + 1$ such that they form a complete multipartite graph, and connect all these new nodes to all nodes in G. Let us color the nodes in F_i with color i.

None of the original nodes in G will ever assume any of the colors 3, 4, ..., k, since they always have $\Delta + 1$ neighbors of these colors, while they have strictly less (at most Δ) neighbors of colors 1 and 2. Nodes in F_i will never have any incentive to switch, since they have no conflicts at all. Thus the process will behave as if the graph only consisted of G with colors 1 and 2. As the new nodes only increase the graph size by a constant factor, we receive an example with the same magnitude of running time, but with k colors.

With the same technique, our lower bound of $\Omega(n^{3/2})$ can also be generalized to the case of up to $\Theta(\sqrt{n})$ colors; details of this are discussed in Appendix B.

4 Sequential adversarial model

We first present a simple example where model A takes $\Omega(n^2)$ steps. Our construction, shown in Figure 2, consist of a group P of size m (for some parameter m), initially colored white, and $2m$ distinct nodes A_1, A_2, ..., A_{2m}, such that A_i is initially colored black for odd values of i and white for even i. Let us connect all nodes A_i to P, and add one more fixed black node that is connected only to P. Finally, let us connect each A_i to $m + 1$ fixed nodes of the same color as A_i. Recall that although the figure shows multiple squares, there are in fact only $n + 1$ fixed black and $n + 1$ fixed white nodes in the graph altogether.

In this graph, P has a balance of 1 initially, while black A_i have a balance of -1 and white A_i have a balance of $-(2m + 1)$. Note that even after execution begins, until A_i is switched for the first time, it will have $m + 1$ fixed neighbors of the same color and at most m neighbors of the opposite color (depending on the current color of P), and thus a negative balance. Therefore, each A_i is switchable anytime if it has not been switched before.

Consider the following sequence of adversarial moves in this graph: the player first decides

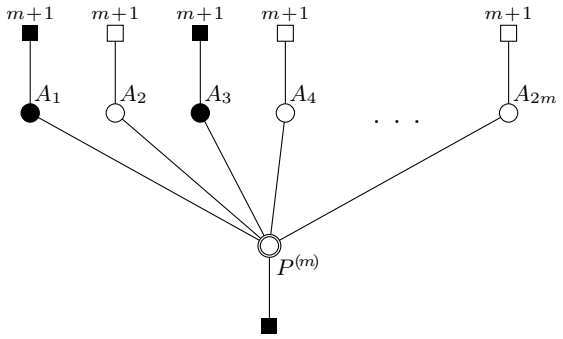

Figure 2 Construction with an adversarial sequence of $\Theta(n^2)$ switches

to switch A_1, then P, then A_2, then P again, then A_3, P, ..., A_{2m}, and finally P again. As each A_i is used only once, they are clearly all switchable. As for P, its balance first changes from 1 to -1, when changing A_1 to white, but increases back to 1 when we switch P itself. Then it changes to -1 once again after changing A_2, so it is switchable again, and so on: each time we switch an A_i, we change it to the same color that P currently has, decreasing P's balance to -1, which increases back to 1 again as we switch P. Therefore, this strategy is indeed a sequence of valid switches.

Since P contains m nodes and is switched $2m$ times in this sequence, this alone contributes to $2m^2$ switches. Altogether, we have $3m$ nodes in the graph (without fixed nodes), allowing us a choice of $m = \frac{n}{3}$. This gives us a sequence with at least $\frac{2}{9}n^2$ steps.

▶ **Theorem 1.** *There exists a graph construction with $\Omega(n^2)$ stabilization time in model A.*

5 Construction for benevolent models

We now presents a construction with $\Omega(n^{3/2})$ stabilization time in benevolent models. Note that it is much more involved to find an example where benevolent models take $\omega(n)$ steps, since in such a construction, we have to ensure that any possible sequence lasts for a long time. In order to have an easy-to-analyze construction, our graph will, at any point in time, contain only one, or a small set of nodes that are switchable, and switching this or these nodes enables the next such set of nodes (i.e., makes them switchable). This way, the switchable point "propagates" through the graph, and the benevolent player has no other valid move than to follow this path of propagation that has been designed into the graph.

The general idea behind the construction is to have a linearly long chain of nodes which is propagated through multiple times. After each such round, the propagation enters a different branch of further nodes; this branch resets the chain for the following round, and then also triggers the following round of propagation (as outlined later in Figure 11).

Due to the complexity of the construction, we do not describe it directly; instead, we define smaller functional elements (*gadgets*) that execute a certain task. We then use these gadgets as building blocks to put our example graph together. This section outlines the tasks and main properties of the gadgets; a detailed description and analysis of each gadget can be found in Appendix B. While the concrete gadget designs are specific to minority processes, they are built on general ideas and techniques for benevolent models; as such, we hope they may inspire similar solutions in the analysis of related processes or cellular automata.

When describing a gadget, the edges connecting the gadget to other nodes in the graphs are drawn as dashed lines in the figures, with the external node usually denoted by v (possibly

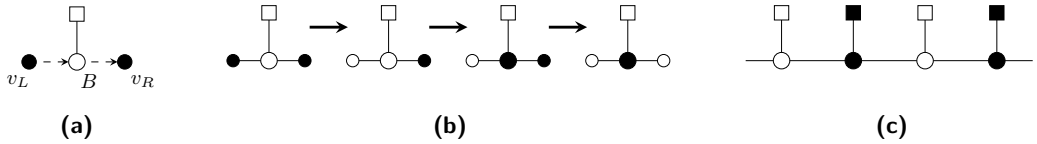

Figure 3 Simple relay gadget (a), the steps of its operation (b), and a chain of relays (c)

with some subscript). Although our graph is undirected, we often refer to such edges as *input* or *output* edges of the gadget, and also show this direction in the figures. This will refer to the role that the external node plays in the functionality of the gadget. That is, whenever the gadget is used in our constructions, it is triggered by (some of) its input nodes switching, and upon completing its task, the gadget makes (some of) its output nodes switchable.

Naturally, as in the entire graph, the role of the two colors is always interchangeable within the gadgets. Therefore, we only present each such gadget in one color variant.

Due to the complexity of the construction, we have also verified its correctness through implementing the process. A discussion of these simulations is available in Appendix C.

Simple relay. As our most basic tool to propagate the only possible point of switching, we use the *simple relay* shown in Figure 3a. A simple relay consists of a base node B, connected to a fixed node of the same color, and two further nodes outside of the gadget, which initially have the opposite color as B. Until neither of v_L and v_R switch, B has positive balance and cannot switch either. However, as soon as v_L switches to the color of B, B becomes switchable, and as B switches, this propagates the point of change to its other neighbor v_R (as shown in Figure 3b).

Note that connecting alternating-colored relays into a chain already gives a simple example of linear stabilization time (see Figure 3c). If the leftmost (white) relay's base node is connected to a fixed white node, then the only available sequence of moves is to switch the base nodes in the relays one by one from left to right, resulting in a sequence of n steps.

Through the concept of input and output nodes, relays essentially allow us to connect other, more sophisticated gadgets in our constructions. If some gadget has an output node v_1 and another gadget has an input node v_2, we can add a chain of relays between v_1 and v_2, ensuring that once v_1 switches, it will be followed by v_2 eventually. Due to this role, relays are not shown explicitly in our final overview figure of the construction, but only represented by arrows, indicating the direction of propagation between more complex gadgets.

Rechargeable relay. A more sophisticated version of a relay is the *rechargeable relay* shown in Figure 4a. In such a relay, node B is extended by an upper node U, a control group C of size 2, and two recharge nodes R_1, R_2, the role of which are interchangeable. Besides v_L and v_R, the nodes R_1 and R_2 also have edges to some external nodes. It is always ensured that the initial balance of R_1 and R_2 from these upper neighbors (that is, with C ignored) is exactly 3.

As in case of a simple relay, if v_L switches, then B itself can switch, followed by v_R. Now assume that in this "used" phase of the relay, some outside circumstance changes 3 neighbors of node R_2 from black to white, and thus its balance changes from the current value of 5 to -1 (the relay is *recharged*). Then R_2 can switch to black, making C and in turn U switch, too. Finally, assume that some other outside circumstance then changes the balance of R_2 from 5 to -1 again (known as *resetting* the relay); then R_2 will switch back to white (with a new balance of 1), and we end up in the initial state of a rechargeable relay of the opposite

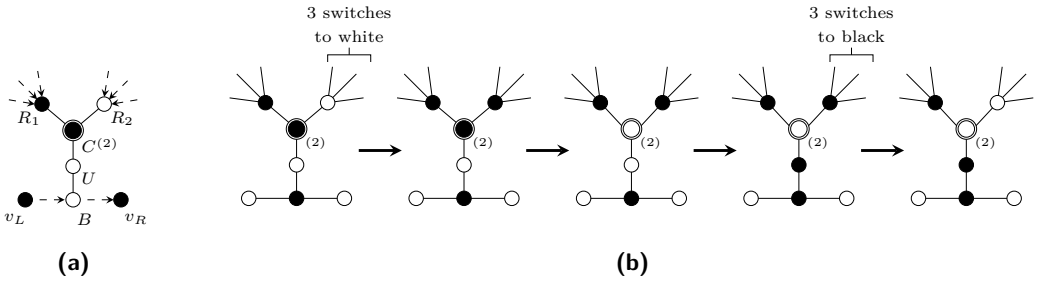

Figure 4 Rechargeable relay gadget (a) and the steps of its operation (b)

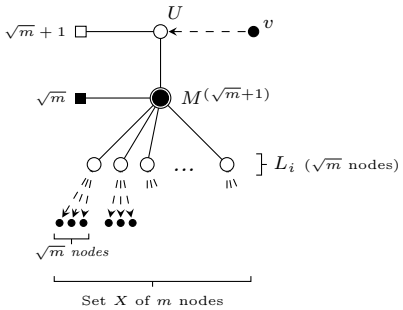

Figure 5 Basic recharging system

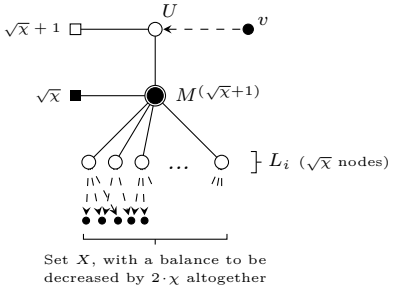

Figure 6 Generalized recharging system

color. The steps of the process are shown in Figure 4b.

This is exactly the essence of this gadget: it is a relay which can be used the same way multiple times. Connecting such gadgets into a chain in the same fashion as Figure 3c, we get a chain that can propagate the point of change not only once, but multiple times if "recharged" through their upper connections between two such propagations.

Recharging system. The rechargeable relay suggests that it is useful to have a tool to "recharge" some nodes, i.e. to decrease their balance by switching some of their neighbors to the color they currently have. To execute this task efficiently on many nodes, we present a *recharging system*.

For the first version of this gadget, assume a setting where there is a set X of m black nodes, and we want to decrease the balance of each of these nodes by 2 (i.e., change exactly one white neighbor of each of them to black). A *basic recharging system*, shown in Figure 5, can execute this task while using only $O(\sqrt{m})$ nodes. The gadget is organized into 3 levels: a single node U in the upper level, a group M of $\sqrt{m}+1$ nodes in the middle level, and $\sqrt{m}$ distinct nodes L_i in the lower level. Each lower level node is connected to $\sqrt{m}$ different nodes in X, thus exactly covering the nodes of X. The gadget operates in a top-to-bottom fashion: once v switches, U turns black, followed by M turning white. Once all nodes in M are switched, the nodes L_i all decide to switch, too.

The key idea in the design of the gadget is that each node L_i has strictly more neighbors in M than in X. This ensures that as long as M is black, the nodes L_i always have a positive balance, regardless of the current color of their neighbors in X. Therefore, no node in the gadget can ever switch before the node U is triggered.

We can use this insight to create a similar gadget for a more general setting. Assume that we similarly have a set X of m black nodes, but instead of decreasing their balance by 2, we want to decrease the balance of each node in X by some specific (possibly different)

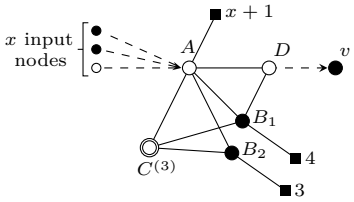

Figure 7 AND gate

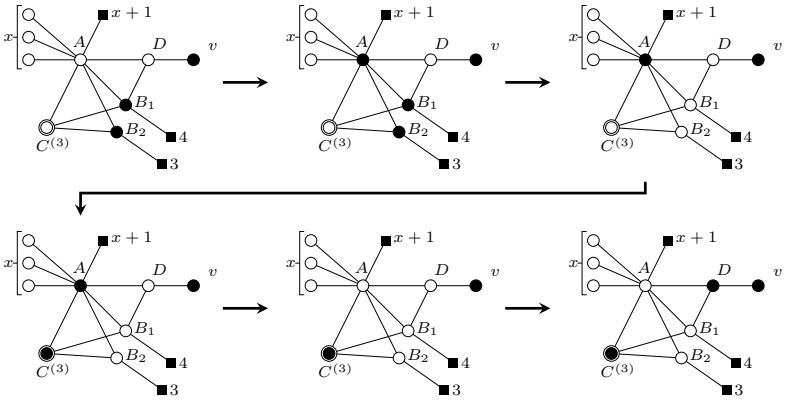

Figure 8 Operation of an AND gate. In the end, node D switches to black, making v switchable.

even value, denoted by $2x_1$, $2x_2$, ..., $2x_m$ (i.e., for the j^{th} node in X, we want to change x_j of its white neighbors to black). Let us denote the sum $\sum_{j=1}^{m} x_j$ of these values by χ.

We can achieve this using a similar construction, shown in Figure 6. In this *generalized recharging system*, we allow multiple nodes L_i to be connected to the same node in X: if a node in X has a corresponding value $2x_j$, then it has exactly x_j neighbors in the lower level of the system. This ensures that once all the nodes L_i switch, the new balance of each node in X is exactly as desired. The number of nodes in the gadget can be minimized by placing $\sqrt{\chi}$ nodes L_i in the lower level, each with $\sqrt{\chi}$ neighbors in X; this way, the overall number of edges going into the set X from the gadget is exactly χ as required. To ensure that the neighborhood of each L_i is dominated by M, we choose the size of group M to be $\sqrt{\chi} + 1$.

AND gate. Another ingredient we use is an AND gate. As its name suggests, this gadget has x input edges from a set of nodes X, and once all nodes in X have switched to the same color (say, white), the gadget triggers a change in another part of the graph.

Note that we could achieve this functionality with a single node, by carefully setting its initial balance such that it switches exactly when all inputs have the desired color. However, AND gates are used to "check" the state of specific nodes in the construction, and as such, it is unfortunate that this check also affects the nodes that are being checked: once the node in this simple AND gate switches, the balance of all input nodes in X will increase by 2. It would be much better to have a gadget that can perform this task without having any effect on the nodes in X.

For this purpose, consider the gadget in Figure 7, which is connected to the nodes in X on the input side and a black node v on the output side. Once all nodes in X are white, node A switches, followed by B_1 and B_2, and then by C. With C switched, A decides to switch back to its original color white. However, since now both A and B_1 are white, this finally

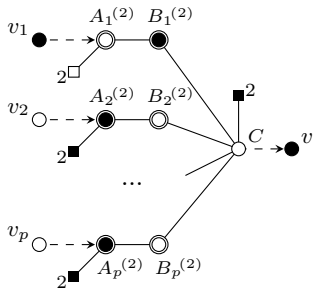

Figure 9 Join gadget

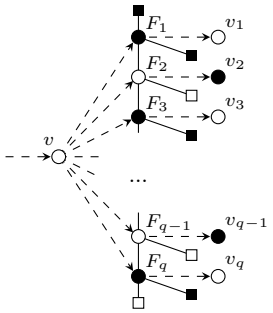

Figure 10 Fork gadget

switches D to black, triggering a change in the output node v (Figure 8). The usefulness of the gadget lies in the fact that by the end of this sequence, A is switched back to its original color, and thus the balance of nodes in X is again the same as it was in the beginning.

Join and fork gadgets. Finally, we need two small gadgets in the construction to fork and join the control sequence at the ends of our main relay chain.

A join gadget, shown in Figure 9, connects a specific number of input nodes v_i to an output node v. When an input node v_i switches, then so does A_i and then B_i in the corresponding input branch, which also switches C and triggers node v. Then when v_{i+1} later switches at some point, the same thing happens to the next input branch and C again, only with the two colors swapping roles. Thus if the nodes v_1, v_2, ... are switched one after another in this order, then each of these input switches make the output node v switch again.

The fork gadget of Figure 10, on the other hand, is responsible for receiving triggers from a given input node v, and directing the propagation to a new branch (a new output node v_i) every time. When v first switches, only v_1 becomes switchable. Similarly, after v is switched for the i^{th} time, only v_i becomes switchable, and thus the gadget triggers the i^{th} branch of output.

Assembling the pieces. Our final graph construction (shown in Figure 11) has two defining parameters m and r. The base of the construction is a chain of m rechargeable relays, connected to a join gadget of r branches and a fork gadget of $r-1$ branches. For each $i \in \{1, ..., r-1\}$, we add a sequence of gadgets (a *branch*) to connect the i^{th} output of the fork to the $i+1^{\text{th}}$ input of the join gadget, which is responsible for recharging the relay chain.

Each branch consists of recharging systems connected to our main chain. First let us consider the rechargeable relays where node U is currently white (either the even or the odd ones; relays at positions of the same parity are all in the same state). We first need a recharging system to recharge all these relays, and then we need another system to reset the relays. We need similarly 2 recharging systems for the other half of the relays which are in the opposite color phase.

Finally, we need to force the player to indeed execute these changes on the relays. For that, we insert an AND gate after each recharging system, which checks if all switchable nodes have indeed been switched before moving on. The output of the AND gate is then used to enable the next recharging systems (or the next input of the join gadget).

This construction ensures that the player has no other choice than to go through the relay chain, follow the next branch from the fork, recharge and reset all the relays, and start

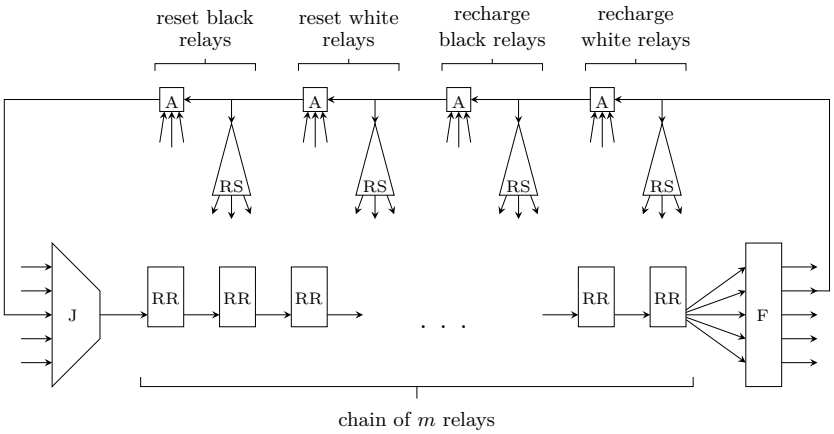

Figure 11 Overview of the whole construction, with one branch shown in detail. Rechargeable relays (RR), Recharging systems (RS), AND gates (A), Joins (J) and Forks (F) are explicitly shown.

going through the relay chain again. Since the chain consists of m relays and it is traversed r times in this process, the switches in the chain add up to $m \cdot r$ steps altogether.

Of course, one also needs to introduce a starting point (initially switchable node) into the construction. This can be done by replacing v_1 in the join gadget by a fixed white node.

Let us consider the number of nodes in the construction. Since rechargeable relays consist of constantly many nodes, the size of the relay chain is $O(m)$. The size of the join and fork gadgets is $O(r)$. Finally, each of the $r-1$ recharging branches consist of constantly many recharging systems, AND gates and simple relays; since the latter two have constant size, branch size is dominated by the size of the recharging systems. Each such system is connected to $\frac{m}{2}$ relays, and thus needs to reduce the balance of $O(m)$ nodes by a constant value of 6. This implies that each recharging system needs $O(\sqrt{m})$ nodes.

This shows that we can choose $r = \Theta(\sqrt{m})$ and $m = \Theta(n)$ for our parameters. Our graph then contains $O(m) + O(r) + r \cdot O(\sqrt{m}) = O(n)$ nodes, so it is indeed a valid setting with the proper choice of constants.

To investigate runtime, it is enough to consider the switches in the main relay chain. Each of the $\Theta(n)$ relays has a base node that is switched $\Theta(\sqrt{n})$ times, adding up to a total of $\Omega(n^{3/2})$ switches.

▶ **Theorem 2.** *There exists a graph construction with $\Omega(n^{3/2})$ stabilization time in model B.*

Note that in the previous construction, whenever any of the base nodes of the relay chain are switchable, there is no other switchable node in the entire graph. This implies that even in the independent benevolent case, the player has no other option than to select this single node, so the number of minimal switches is $\Omega(n^{3/2})$ even if we assume the independent benevolent model.

In fact, one can observe that the construction also ensures that regardless of the choices of the player, the set of switchable nodes is always an independent set at any point in the process. Hence models C and D are in fact the same in this graph, and thus the lower bound also holds for model D. This then implies the same bound for all the remaining models.

▶ **Corollary 3.** *There is a graph construction with $\Omega(n^{3/2})$ stabilization time in models C–G.*

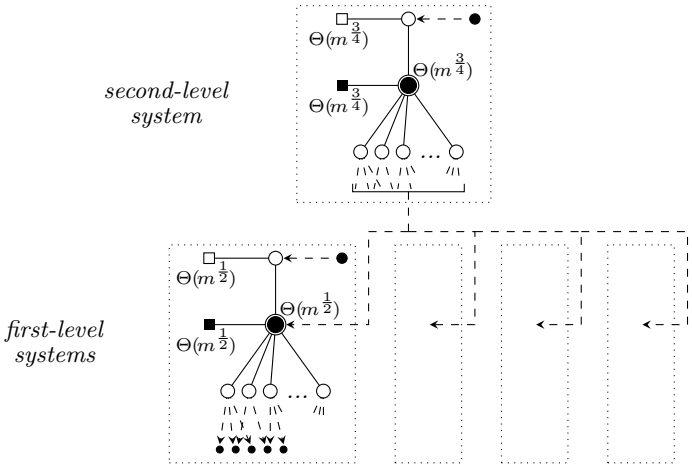

Figure 12 Connection of a second-level recharging system to first-level recharging systems. For simplicity, only the recharging of group M is shown (node U also has to be recharged).

6 Recursive construction

We now briefly outline the modification idea that provides the almost tight lower bound of $\Omega(n^{2-\epsilon})$. A more detailed discussion of the construction can be found in Appendix A.

The key idea is to make the recharging systems themselves also rechargeable, so that they can recharge the same output nodes repeatedly. Note that once a recharging system has been used, the color of its nodes is exactly that of a recharging system of the opposite color. Thus, if we reset the balance of each node in the system to its initial value, we can use the system again to recharge the same output nodes again. More specifically, given a used recharging system, we need to restore the balance of M and U to 1 in order to obtain a recharging system of the opposite color; then by triggering U again, we can use the system to recharge the nodes in X once more.

Therefore, we can add a layer of *second-level* recharging systems to recharge all the original (first-level) systems in the graph after all first-level system have been used, as illustrated in Figure 12. Recall that decreasing the sum of balances in a set of nodes by χ requires a recharging system of $O(\sqrt{\chi})$ nodes. We have $\Theta(\sqrt{m})$ first-level systems in our graph, each consisting of $\Theta(\sqrt{m})$ nodes, with a balance of $\Theta(\sqrt{m})$ after use; to reset each node in these systems to their default balance of 1, with $\chi = \Theta(m^{3/2})$, a second-level system requires $\sqrt{\chi} = \Theta(m^{3/4})$ nodes.

In order to keep the overall number of nodes in second-level systems in $O(m)$, we add $\Theta(m^{1/4})$ distinct second-level systems to our graph. When used, each of these second-level systems recharges all systems on the first level, which in turn allows us to propagate through the main relay chain $\Theta(m^{1/2})$ times again. Therefore, with $\Theta(m^{1/4})$ second-level systems in the construction, the first two levels already allow us to traverse the main relay chain $\Theta(m^{1/2}) \cdot \Theta(m^{1/4})$ times.

We can continue this technique in a recursive manner. Assume that we have $\Theta(m^{1/(2^i)})$ distinct i^{th}-level systems in the construction, each consisting of $\Theta(m^{1-1/(2^i)})$ nodes (which, therefore, all have a balance of $\Theta(m^{1-1/(2^i)})$ after they have been used). We can then use an $(i+1)^{\text{th}}$-level recharging system to recharge all of these i^{th}-level systems; since we now have $\chi = \Theta(m^{1/(2^i)}) \cdot \Theta(m^{1-1/(2^i)}) \cdot \Theta(m^{1-1/(2^i)}) = \Theta(m^{(2^{i+1}-1)/(2^i)})$, this requires a next level system of $\sqrt{\chi} = \Theta(m^{(2^{i+1}-1)/(2^{i+1})}) = \Theta(m^{1-1/(2^{i+1})})$ nodes. In order to keep the nodes in

this new level also in $O(m)$, we only add $\Theta(m^{1/(2^{i+1})})$ systems to the $(i+1)^{\text{th}}$-level.

Generally, these higher-level recharging systems fit into our construction in the following way. Every time when first-level systems have all been used, an extra branch is added to the construction, which uses one of the second-level systems to recharge the entire first level (and does not influence the relay chain). Similarly, whenever we would need such a second-level branch but all of them has been used, a third-level branch is added to recharge all second-level systems, and the required second-level branch is only visited after traversing this third-level branch.

Following the recursive pattern, we obtain a construction that allows us to traverse the main relay chain $\Theta(m^{1/2}) \cdot \Theta(m^{1/4}) \cdot \Theta(m^{1/8}) \cdot ...$ times altogether. If the number of levels go to infinity with m increasing, then for any $\epsilon > 0$, there is an m large enough that the number of relay chain traversals is at least $\Theta(m^{1-\epsilon})$. Since the relay chain consists of $\Theta(m)$ nodes, this leads to a stabilization time of $\Theta(m^{2-\epsilon})$.

If we have $\Theta(m^{1/(2^i)})$ recharging systems on the i^{th} level, this setting allows us to add $\Theta(\log\log m)$ levels until the number of systems on a level decreases to a constant value.

Now let us analyze the number of nodes in the graph. On each level, the systems contain $\Theta(m)$ nodes altogether, so the number of nodes in recharging systems adds up to $\Theta(m \log\log m)$ over all levels. One can easily show that the size of the graph is dominated by these nodes. The number of branches controlling first-level systems is $\Theta(m^{1/2} \cdot m^{1/4} \cdot m^{1/8} \cdot ...) = O(m)$, the number of branches controlling second-level systems is only $\Theta(m^{1/4} \cdot m^{1/8} \cdot ...) = O(m^{1/2})$, and so on, the number of i^{th}-level branches is $O(m^{1/2^{i-1}})$. Summing these up, the number of branches altogether is still $O(m)$. Apart from recharging systems, each branch contains constantly many nodes only (in the form of simple relays, AND gates, and the corresponding parts of the fork and join gadgets). This shows that the number of nodes outside of the recharging system is only $O(m)$ altogether, thus the number of nodes in the entire graph is indeed $\Theta(m \log\log m)$.

This allows for a choice of $m = \Theta(\frac{n}{\log\log n})$, leading to a stabilization time of $\Omega(\frac{n^{2-\epsilon}}{(\log\log n)^{2-\epsilon}})$. Since this bound holds for any $\epsilon > 0$, we can easily remove the logarithmic factors: a lower bound of $\Omega(n^{2-\epsilon})$ follows from the same construction for any $\widehat{\epsilon} < \epsilon$. Thus the construction shows that the number of steps is $\Omega(n^{2-\epsilon})$.

Similarly to the non-recursive case, this lower bound holds in all of our models, since propagations over the relay chain are still only possible sequentially.

▶ **Theorem 4.** *For any $\epsilon > 0$, there exists a graph construction with $\Omega(n^{2-\epsilon})$ stabilization time in models B–G.*

References

1. Cristina Bazgan, Zsolt Tuza, and Daniel Vanderpooten. Satisfactory graph partition, variants, and generalizations. *European Journal of Operational Research*, 206(2):271–280, 2010.
2. Olivier Bodini, Thomas Fernique, and Damien Regnault. Crystallization by stochastic flips. In *Journal of Physics: Conference Series*, volume 226, page 012022. IOP Publishing, 2010.
3. Zhigang Cao and Xiaoguang Yang. The fashion game: Network extension of matching pennies. *Theoretical Computer Science*, 540:169–181, 2014.
4. Jacques Demongeot, Julio Aracena, Florence Thuderoz, Thierry-Pascal Baum, and Olivier Cohen. Genetic regulation networks: circuits, regulons and attractors. *Comptes Rendus Biologies*, 326(2):171–188, 2003.
5. Silvio Frischknecht, Barbara Keller, and Roger Wattenhofer. Convergence in (social) influence networks. In *International Symposium on Distributed Computing*, pages 433–446. Springer, 2013.
6. Bernd Gärtner and Ahad N Zehmakan. Color war: Cellular automata with majority-rule. In *International Conference on Language and Automata Theory and Applications*, pages 393–404. Springer, 2017.
7. Bernd Gärtner and Ahad N Zehmakan. Majority model on random regular graphs. In *Latin American Symposium on Theoretical Informatics*, pages 572–583. Springer, 2018.
8. Eric Goles and Jorge Olivos. Periodic behaviour of generalized threshold functions. *Discrete Mathematics*, 30(2):187–189, 1980.
9. Sandra M Hedetniemi, Stephen T Hedetniemi, KE Kennedy, and Alice A Mcrae. Self-stabilizing algorithms for unfriendly partitions into two disjoint dominating sets. *Parallel Processing Letters*, 23(01):1350001, 2013.
10. Dominik Kaaser, Frederik Mallmann-Trenn, and Emanuele Natale. Brief announcement: On the voting time of the deterministic majority process. *Distributed*, page 647, 2015.
11. Barbara Keller, David Peleg, and Roger Wattenhofer. How even tiny influence can have a big impact! In *International Conference on Fun with Algorithms*, pages 252–263. Springer, 2014.
12. Jeremy Kun, Brian Powers, and Lev Reyzin. Anti-coordination games and stable graph colorings. In *International Symposium on Algorithmic Game Theory*, pages 122–133. Springer, 2013.
13. Pál András Papp and Roger Wattenhofer. Stabilization time in weighted minority processes. In *36th International Symposium on Theoretical Aspects of Computer Science (STACS 2019)*. Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik, 2019.
14. Damien Regnault, Nicolas Schabanel, and Éric Thierry. Progresses in the analysis of stochastic 2d cellular automata: A study of asynchronous 2d minority. In Luděk Kučera and Antonín Kučera, editors, *Mathematical Foundations of Computer Science 2007*, pages 320–332. Springer Berlin Heidelberg, 2007.
15. Damien Regnault, Nicolas Schabanel, and Éric Thierry. On the analysis of "simple" 2d stochastic cellular automata. In *International Conference on Language and Automata Theory and Applications*, pages 452–463. Springer, 2008.
16. Jean-Baptiste Rouquier, Damien Regnault, and Éric Thierry. Stochastic minority on graphs. *Theoretical Computer Science*, 412(30):3947–3963, 2011.
17. Saharon Shelah and Eric C Milner. Graphs with no unfriendly partitions. *A tribute to Paul Erdös*, pages 373–384, 1990.

Appendices

A Further discussion of the recursive construction

While the main idea of the recursive construction has been outlined above, there are some details worth discussing for completeness.

Since a second-level system can only be used to recharge nodes of the same color, every time we recharge all the first-level systems, we in fact need two second-level recharging systems, one of each color.

Recall that in addition to the group M, the balance of node U also has to be reset between two uses of a recharging system; however, we did not point this out when calculating the necessary size of systems, since besides M, a single extra node does not affect the magnitude. Earlier, we have noted that the recharging systems on a certain level consist of two classes of systems of different color; observe that the next level systems that recharge the groups M in one class can simultaneously be used to also recharge the nodes U in the other class. Alternatively (for simpler analysis), we can add an extra recharging system (of the same size) on each branch in order to separately recharge the nodes U on the level below.

Note that the number of total relay chain traversals, which is $\Theta(m^{1/2} \cdot m^{1/4} \cdot m^{1/8} \cdot ...)$, is in fact only guaranteed to be at least $\Theta(m^{1-\epsilon})$ if the coefficients in these factors are sufficiently large. With an analysis of the constants in the process, one can show that the coefficient in each factor can indeed be chosen as 1, and thus these constant do not add up to dividing logarithmic factors when taking the product. However, this is in fact unnecessary, as any such logarithmic factor could also be removed simply by a smaller choice of ϵ.

Finally, note that in this recursive setting, recharging systems are slightly modified in the sense that they have multiple input nodes from multiple different branches, each connected to node U. However, this does not modify the behavior of U as long as its initial balance is readjusted to 1. This also requires a minor modification in the simple relays that are used as input nodes, since relays generally assume that their output node never switches before the relays themselves are triggered. This can be resolved by using a modified relay where the base node has an initial balance of 3, and thus it is enabled by two distinct simple relays on the branch.

B Detailed analysis of gadgets

Here we provide a more detailed description of the gadgets, and also comment on their behavior and their use in the construction.

Simple relay. The construction and behavior of the simple relay has already been described above. One thing to note is that in our construction, simple relays are always used only once: after node B switches, propagation never returns to the same part of the graph again, and thus node B will remain unswitchable for the rest of the process.

While we mostly use this original version of the gadget, we occasionally need relays with multiple output nodes instead of just one. This only requires a simple modification: besides connecting x extra (black) output nodes to node B, we also need to add x fixed (white) nodes in order to keep the initial balance of B unchanged.

Chains of simple relays are mostly used to connect more complex gadgets in our construction. Note that depending on whether the input and output nodes in these gadgets are

supposed to have the same or different initial colors, we only need a chain of length 1 or 2 for this, respectively.

Rechargeable relay. In a rechargeable relay, node B is connected to an upper node U instead of a fixed node. Node U is connected to a group C of two nodes, which is further connected to nodes R_1, R_2. Initially, C has the opposite color as B and U, and one of R_1 and R_2 is white, the other is black. Node B has the same external neighbors as a simple relay. The recharge nodes can both have any set of external neighbors as long as their initial balance is 3 with C ignored (so with C included, the initial balance of R_1 and R_2 is then 1 and 5, respectively).

Note that since R_1 and R_2 have opposite colors, this recharging process can always be executed on a used relay through either R_1 or R_2, depending on the current color of the nodes. We only need to select the recharge node that has the current color of U, and switch 3 of its neighbors (to U's current color) for the recharging step, and then switch 3 of its neighbors (to the opposite color) for the resetting step.

Recharging system. In a basic recharging system, the node U is connected to the input node v, the group M, and to $\sqrt{n}+1$ fixed white nodes. The middle level group M has a further edge to all nodes L_i, and is balanced by $\sqrt{m}$ fixed black nodes. Finally, each node L_i has $\sqrt{m}$ distinct neighbors in X, and thus each node in X is connected to exactly one lower-level node. For convenience, we assume that m is a square number.

A generalized recharging system is almost identical to this, except for the nodes L_i occasionally being connected to the same node. The connections between the lower level and X are not directly specified: we are free to choose which of the nodes L_i to connect to a specific node in X. Note, however, that the gadget design implicitly assumes that $x_j \leq \sqrt{\chi}$ for all nodes in X. This is naturally satisfied whenever we use the gadget in our constructions, since we always have $x_1 = x_2 = ... = x_m$ with $|X| > x_j$. Also note that for convenience, we assume χ to be a square number.

Nodes in the upper and lower levels are initially white, while M and the input node v are initially black. The nodes X may assume any color, and also may switch multiple times before the recharging system is activated. However, the graph construction ensures that at the time when the gadget is activated (that is, when v switches), all nodes in X are currently colored black (i.e., we indeed use the system on rechargeable relays that can currently be recharged). The gadget design ensures that U and M have an initial balance of 1, while the nodes L_i have a balance of 1 at least, depending on the current color of their neighbors in X.

AND gate. The AND gate consist of 7 nodes. The input nodes of X are connected to node A, which is further connected to all other nodes in the gadget (B_1, B_2, D and the group C on 3 nodes). Nodes B_1 and B_2 are also connected to group C, node B_1 has an edge to node D, and node D is connected to some external black node v on the output side. Furthermore, A, B_1 and B_2 have $x+1$, 4 and 3 fixed black neighbors, respectively.

One can check that each node has a positive balance as long as there exists a black node in X. Node A gets a balance of $x-1$ from the nodes within the gadget, so it is not switchable unless all nodes in X are white. Nodes B_1, B_2, C and D all have an initial balance of 1.

After the gadget reaches its final stage (see Figure 8), no node in the gadget can ever change again, regardless of the states of X or v.

Note that for the described behavior of the gadget, we also need the fact that none of the nodes in X switch between the first and second switching of A. The switching of A only

increases their balance (temporarily), so this is guaranteed if other neighbors of nodes in X do not interfere with the process. In the construction, we only use AND gates this way: whenever a node A becomes switchable in a gate, then that is the only switchable node in the entire graph, so no other nodes will switch until the propagation reaches v.

As long as this condition is fulfilled, we can connect any number of AND gates to a given node of the graph without affecting its behavior; we only have to make sure that we also add fixed node neighbors to restore the node's balance to the original value.

Join gadget. The join gadget consists of a central node C, and of p distinct 2-group *starter gadgets* of alternating color (we assume p to be even). Each starter gadget consists of two groups A_i and B_i, both of size 2 (with $i \in \{1, 2, ..., p\}$). The two groups are connected to each other, and A_i has a further edge to the input node v_i, and two fixed nodes of the same color as its own. Finally, all B_i are connected to a central node C, which is in turn connected to an output node v. Node C also has two further fixed black connections.

Initially, A_i for odd i values, B_i for even i values, v_i for even i and node C are colored white; the remaining nodes are colored black. Nodes A_i have an initial balance of 1, nodes B_i have an initial balance of 1 or 3 (depending on parity), and C has an initial balance of 3.

Every time after v switches, the balance of C returns to its initial value of 3, so the switching of the next input node will trigger the same process through the next starter gadget.

Fork gadget. The fork gadget consists of q nodes F_1, ..., F_q of alternating color, where we assume q to be an odd number. All F_i are connected to the same input node v, and each to a distinct output node v_i. They are also linked to each other, with F_i connected to F_{i+1} for all $i \in \{1, 2, ..., q-1\}$. Also, node F_1 and F_q have a fixed neighbor colored black and white, respectively (imitating the role of the nonexistent nodes F_0 and F_{q+1}). Finally, each F_i has a further fixed neighbor of its original color. Initially, F_i is colored black for odd i and white for even i values.

The balance of F_1 and all white F_i-s is originally 1 in this setting, while the balance of black F_i-s (except for F_1) is 3. Hence when v first switches, only F_1 will become switchable (and switching it will propagate on through v_1). The next time v switches, it switches back to white; with v and F_1 both white, F_2 can now switch too. The pattern continues all the way to F_q: as F_{i-1} has already been switched before, as soon as v switches back to the color of F_i, F_i becomes switchable, too, enabling propagation on the next branch. After v_i switches (and remains that way), F_i is not switchable anymore, since v_i, F_{i-1} and its fixed neighbor all have the opposite color.

Note that since each switching F_i increases the current balance of v from 1 to 3, we need to switch two neighbors of v in each turn to make v switchable again. This is exactly what happens when v is the base node of the rightmost relay in the chain: between every consecutive switches of v, we switch both node U (by the recharging step) and node v_L (by propagation through the chain) in the relay, and thus v becomes switchable again.

Note that since it is connected to the fork gadget, the rightmost rechargeable relay in the chain is a modified one in the sense that its base node has not one, but q right-side neighbors, colored in alternating fashion. However, this fact does not change its behavior at all. The initial balance of the base node is still 1, and every time after v switches, it has one of its neighbors F_i switching in the opposite direction. That has exactly the same effect as if the right neighbor was simply a subsequent relay in the chain, triggered by v.

On the whole construction. For convenience, we assume in the construction that both m and r are even numbers.

Recharging systems and AND gates, as all other gadgets, are available in two color variants; in the overview of the construction, we did not discuss which variant is used in which case. However, the current state of each relay in each round is straightforward to calculate, so the necessary color of all recharging systems and AND gates can easily be determined.

Also, we have seen that AND gates are used to ensure that the given recharging or resetting operations have completely been executed. In order to achieve this, in case of the first systems (which recharge relays), the input edges of the gates can be connected to the upper nodes of the corresponding relays, since that is the last node to switch in the sequence. In case of the systems that reset relays, the aim is only to switch the corresponding recharge node of the relay, so we can connect the gates to the recharge nodes.

However, as each AND gate belongs to a certain branch of the construction, we also have to ensure that the AND gate is only activated when the propagation reaches this branch, and stays inactive as long as previous branches are being processed. Therefore, besides the specified nodes in the relays, the final input node of the AND gate is the node which was used to enable the recharging system in question (node v of Figure 5). This way, the gates ensure that *after* the recharging system is activated, propagation only continues if all the resulting switches were executed.

Generalization to $\omega(1)$ colors

One can observe that in the construction of Section 5, except for nodes A in the AND gates, all nodes in the graph have a degree of $O(\sqrt{n})$. We can slightly modify the construction and replace each of these AND gates with two levels of such gates, with $\Theta(\sqrt{n})$ distinct gates on the first level (each with $\Theta(\sqrt{n})$ input nodes), and a final gate that connects the outputs of these first-level gates. This gives us a construction with the same properties, but a maximum degree of $O(\sqrt{n})$.

This allows us to generalize the lower bound of $\Omega(n^{\frac{3}{2}})$ to the case of not only $O(1)$, but up to $O(\sqrt{n})$ colors. The technique for this is the same as in the case of $O(1)$ colors: we add a multipartite graph colored with the additional colors, and connect each of its nodes to each original node. With $\Delta = O(\sqrt{n})$ established, it suffices to have $\Theta(\sqrt{n})$ nodes in each of the color classes. Therefore, using only $\Theta(n)$ additional nodes, we can extend the graph by a multipartite graph on $\Theta(\sqrt{n})$ color classes, each consisting of only $\Theta(\sqrt{n})$ nodes.

C Notes on simulations

Due to its complexity, we have also verified the correctness of the non-recursive construction of Section 5 through implementing it and running a simulation of the minority process. Note that in general, it is difficult to simulate a minority process in a benevolent model, since all possible switching sequences would have to be examined to find the one with the smallest number of steps.

Fortunately, the task is significantly simpler in our case, due to the properties of the construction. The key observation in our graph is that whenever propagation is split into multiple parallel threads (that is, when there are multiple switchable nodes at the same time), then propagation on any of these threads does not influence propagation on other threads at all. Specifically, the nodes on separate threads do not have common neighbors except for the beginning and end of such threads; i.e. when a switching node splits the propagation to multiple threads, or when threads are joined in an AND-*like* fashion, meaning

that a common neighbor only becomes switchable when propagation has been finished in all of the threads. This implies that throughout the process, these threads can be handled completely independently from each other, and the order in which they are processed is irrelevant. Note that this is also the property of the construction which ensures that the set of switchable nodes is an independent set in any state.

If we exploit this property, the process can be simulated easily by always choosing an arbitrary one of the switchable nodes in the graph, knowing that the choice of nodes will not influence the outcome. To verify correctness in such a simulation, we only have to check that in each step of the process, the set of nodes that become switchable is exactly the set of nodes determined by the analysis. Note that the opposite does not happen in our construction: the switching of a node never makes another switchable node unswitchable (this would also contradict the property that switchable nodes form an independent step in any state).

When examining concrete instances of our construction, we used the parameter r as the input to determine the size of the instance. For a given input value of r (always an even number), we have chosen $m = 2 \cdot (r-1)^2$, which fits our preconditions on both magnitudes and parity. All other details of the construction are already determined above; the only additional thing to note is that whenever different gadgets are connected through a chain of simple relays, we always use the smallest possible such chain in the implementation.

The simulations verified that the analysis of the construction is correct, and thus stabilization time is indeed $\Omega(n^{3/2})$ in model B. Table 1 illustrates the number of steps for some choices r, along with the resulting number of nodes in the construction. One can observe that the number of steps indeed grows superlinearly in n.

Input (r)	Nodes (n)	Steps
2	99	112
4	469	772
8	1 929	5 884
16	7 729	47 404
24	17 369	161 372
30	27 119	316 568
40	48 169	754 108
60	108 269	2 559 188
80	192 369	6 084 268
100	300 469	11 905 348
120	432 569	20 598 428

Table 1 Number of steps on some specific graphs