Fault Tolerant Coloring of the Asynchronous Cycle

Pierre Fraigniaud
Université Paris Cité, CNRS, IRIF, F-75013, Paris, France

Patrick Lambein-Monette[1]
Université Paris Cité, CNRS, IRIF, F-75013, Paris, France

Mikaël Rabie
Université Paris Cité, CNRS, IRIF, F-75013, Paris, France

Abstract

We present a wait-free algorithm for proper coloring the n nodes of the asynchronous cycle C_n, where each crash-prone node starts with its (unique) identifier as input. The algorithm is independent of $n \geqslant 3$, and runs in $\mathrm{O}(\log^* n)$ rounds in C_n. This round-complexity is optimal thanks to a known matching lower bound, which applies even to synchronous (failure-free) executions. The range of colors used by our algorithm, namely $\{0, \ldots, 4\}$, is optimal too, thanks to a known lower bound on the minimum number of names for which renaming is solvable wait-free in shared-memory systems, whenever n is a power of a prime. Indeed, our model coincides with the shared-memory model whenever $n = 3$, and the minimum number of names for which renaming is possible in 3-process shared-memory systems is 5.

2012 ACM Subject Classification Theory of computation → Distributed algorithms; Mathematics of computing → Graph coloring; Computer systems organization → Dependable and fault-tolerant systems and networks; Theory of computation → Models of computation

Keywords and phrases graph coloring, LOCAL model, shared-memory model, immediate snapshot, renaming, wait-free algorithms

Digital Object Identifier 10.4230/LIPIcs.DISC.2022.23

Related Version *Full Version*: `https://arxiv.org/abs/2207.11198`

Funding *Pierre Fraigniaud*: additional support from the project ANR-20-CE48-0006 (DUCAT).

1 Introduction

1.1 Motivation

Two forms of coloring tasks are at the core of distributed computing. One is *vertex-coloring* [8] in the framework of synchronous distributed network computing [29]. The other is *renaming* [3] in the framework of asynchronous shared-memory distributed computing [7]. For both tasks, each process starts with its own *identifier* as input, which is supposed to be unique in the system, and must compute a *color* as output. The identifiers are supposed to be in a large range of values (typically of size $\mathrm{poly}(n)$), while the colors should lie in a restricted range of values, typically $\{0, \ldots, k-1\}$ for some $k \geqslant 1$. Depending on the context, k may be an absolute constant, or may depend on parameters of the system, like the maximum degree Δ of the network, or even the total number n of processes. In the context of network computing, the outputs must properly color the underlying graph of the network, i.e., any two neighboring nodes must output distinct colors. In the context of shared-memory computing, each process must output a color that is unique in the system, i.e., different from the color of any other process.

[1] Correponding author

36th International Symposium on Distributed Computing (DISC 2022).
Editor: Christian Scheideler; Article No. 23; pp. 23:1–23:22
Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

On the negative side, it is "hard" to color cycles of even size using only two colors in a distributed manner [26], in the sense that $\Omega(n)$ synchronous rounds of communication are required to solve this problem in the n-node cycle C_n (n even; 2-coloring an odd length cycle is impossible). A synchronous *round* consists of (1) an exchange of information between the two end-points of every edge in the network, and (2) a local computation at every node. Similarly, renaming the n processes of an asynchronous shared-memory system in a wait-free manner using a palette with fewer than $2n-1$ names (i.e., k-renaming with $k < 2n-1$) is impossible [6, 14, 24] whenever n is a power of a prime number ($n = 6$ is the smallest integer for which this bound does not hold [15]). *Wait-free* essentially means that each process terminates in a bounded number of write/read steps, independently of the asynchronous scheduling of the $n-1$ other processes, i.e., independently of the interleaving of read and write operations in the shared memory.

On the positive side, it is known that 3-coloring the n-node cycle C_n for $n \geqslant 3$ can be achieved in $\frac{1}{2}\log^* n + \mathrm{O}(1)$ synchronous rounds thanks to *deterministic coin tossing*, an efficient color-reduction technique due to Cole and Vishkin [17][2]. This bound is tight, as no algorithms can 3-color the n-node cycle in less than $\frac{1}{2}\log^* n - 1$ rounds, thanks to Linial's celebrated lower bound [26]. In shared-memory systems, while $(2n-2)$-renaming is impossible wait-free for infinitely many values of n, $(2n-1)$-renaming can be achieved wait-free for all values of $n \geqslant 2$ [3].

The above results are at the core of two separate lines of intensive research. One line studies extensions of 3-coloring the synchronous cycle, in particular $(\Delta+1)$-coloring arbitrary networks of maximum degree Δ; see, e.g., [9, 20, 23, 30] for recent contributions in this field. This line also studies variants of $(\Delta+1)$-coloring, including, for example, Δ-coloring, edge-coloring, weak-coloring, defective coloring; see, e.g., [8, 21, 22, 28]. The other line of research studies variants of renaming (e.g., long-lived [1, 5]), renaming in different shared-memory or message-passing models (e.g., [2, 16]), and the search for algorithms using fewer names whenever n is not a power of a prime [4, 15].

1.2 Objective

Our aim is to study coloring tasks in a framework relaxing two strong assumptions made in the aforementioned contexts. First, it relaxes the "all-to-all assumption" of the shared-memory model, which enables some form of global communication between the processing nodes, or processes. Second, our framework relaxes the "synchrony assumption" of the LOCAL [29] model of network computing, where the processes proceed in lock-step, in the sense that we allow *processes* to be fully asynchronous and crash-prone, while we keep reliable and instantaneous *communications* (the latter is in contrast with the classic asynchronous model known as *message-passing* [19], where, in addition, the delivery of messages is itself asynchronous). Specifically, we consider a round-based, asynchronous computing model in the n-node cycle C_n, where each *round* of a process consists of the following sequence of operations: (1) writing in its local register, (2) reading the local registers of its two neighbors in C_n, and (3) updating its local state.

The difference with the standard LOCAL model, in which vertex-coloring is typically studied, is that the rounds are asynchronous. That is, the scheduler may allow some processes to perform many rounds while other processes may perform just a few rounds, or even no

[2] For every $x > 0$, let $\log^{(0)} x := x$ and, for $k \geqslant 0$ such that $\log^{(k)} x > 0$, let $\log^{(k+1)} x := \log_2(\log^{(k)} x)$; $\log^* x$ is then defined as the smallest $k \geqslant 0$ such that $\log^{(k)} x \leqslant 1$.

rounds. Moreover, the operations performed during a round are also asynchronous, e.g., a process can write, read, and then spend a lot of time idle before changing state. In particular, the cycle may become disconnected, and processes may become isolated, due to processes that are very slow, or even crashed (a crash is a full-stop form of failure: a crashed process stops functioning, and does not recover). As a consequence, information may propagate poorly in the network due to slow or crashed processes.

The difference between our setting and typical models (e.g., shared memory) used for studying renaming [7] is that the processes do not share a single array of single-writer/multiple-reader registers. Instead, only processes sitting on adjacent nodes in C_n can read each-other's registers. Thus, instead of having processes perform snapshot operations – i.e., read the registers of all processes at once – or even immediate snapshots – i.e., write a value *and* read everything all at once – each process is restricted to *local* (immediate) snapshots, i.e., that only read the registers of its neighbors in the cycle.

We seek to address a few basic questions about this model. Is wait-free proper vertex-coloring at all possible in C_n? That is, can the n processes of the asynchronous cycle pick colors distinct from those of their neighbors in a bounded number of computational steps? If yes, what is the smallest range of colors that make it possible to color the asynchronous cycle C_n? And what is the smallest number of asynchronous rounds that a process may have to perform in order to achieve this task?

Note that it is a priori unclear whether wait-free proper vertex-coloring is at all possible in the asynchronous cycle, even if allowing a large range of colors (but less than the number n of processes). Indeed, there are very similar problems which are not solvable in this framework. An example is *maximal independent set* (MIS). MIS and 3-coloring are reducible one to another in the cycle under the synchronous failure-free setting [26]; in contrast, MIS is *not* solvable wait-free in the asynchronous crash-prone version of the LOCAL model considered in this paper (Property 1). Indeed, as we detail further down, a wait-free algorithm for MIS could be simulated in the asynchronous shared-memory model for solving *strong symmetry-breaking* wait-free, which was proved impossible in [6].

1.3 Our Results

We describe a wait-free algorithm for proper coloring the n processes of the asynchronous crash-prone cycle C_n. So, wait-free proper vertex-coloring is possible in C_n, as opposed to, e.g., MIS. Our algorithm is independent of $n \geqslant 3$, and each process performs $\mathrm{O}(\log^* n)$ asynchronous rounds in C_n. The round complexity of our algorithm is therefore asymptotically optimal, thanks to Linial's lower bound [26], which holds for the executions of our model that are synchronous and failure-free.

The range of colors used by our algorithm, namely $\{0, \ldots, 4\}$, is optimal too for the class of all cycles, thanks to the aforementioned minimum number $2n-1$ of names for which renaming is solvable wait-free in shared-memory systems, whenever n is a power of a prime. Indeed, in the specific case of the cycle C_3, our model coincides with the shared-memory model with $n = 3$ processes, which implies that proper coloring C_3 with less than five colors is impossible.

To our knowledge, our algorithm is the first distributed coloring algorithm designed for a framework combining the following two sources of difficulties: on the one hand, the possibility of crash failures in a fully asynchronous setting, and, on the other hand, a network limiting direct communications between processes.

Our Technique. Our main algorithm, given in Algorithm 3, has two components.

The first component of Algorithm 3 is introduced standalone in Algorithm 2. It bears some resemblance to the rank-based $(2n-1)$-renaming algorithm (see [7, Algorithm 55], and [3, Step 4 in Algorithm A]). It is a wait-free 5-coloring algorithm for C_n, i.e., in each of its executions over a cycle of length $n \geqslant 3$, the processes that perform enough computational steps output a color in the set $\{0,\dots,4\}$, and no two neighboring processes output the same color. However, Algorithm 2 is slow, in the sense that its running time may be as large as the longest sub-path of the cycle along which process identifiers are increasing, which can be as large as $\Theta(n)$.

The second component of Algorithm 3 uses and modifies the identifiers, in parallel to the first component. This quickly shortens such increasing sub-paths, until their length less than some constant $L \leqslant 10$, in a manner directly inspired from Cole and Vishkin's method [17]. Each process starts with its input identifiers, and successively tries to adopt new ones taken from increasingly smaller ranges of identifiers, by performing $\mathrm{O}(\log^* n)$ identifier-reductions. As this reduction process goes on, the identifiers might not remain unique in the cycle, but we ensure that they nonetheless maintain a proper coloring, i.e., adjacent processes always hold distinct identifiers. This invariant is difficult to enforce in an asynchronous environment, and we resort to a synchronization mechanism by which a process awaits a "green light" from both of its neighbors each time it seeks to change its identifier.

The second component of our algorithm is thus not wait-free by itself, since processes are constantly waiting for "green lights" from their neighbors. However, it offers *starvation free* progress [25]: termination is guaranteed whenever all processes perform infinitely many computational steps. Our core result is that the interaction between the two components, i.e., between the (wait-free) first component and the (starvation-free) second component, remains itself wait-free, and has a running time $\mathrm{O}(\log^* n)$.

We can decompose the description of the first component further, into a starvation-free subcomponent that looks for a color a_p for every process p, which does not collide with the colors of the neighbors of p with greater identifiers, and in another subcomponent that looks for a (potentially different) color b_p for process p, which doesn't collide with the colors of *any of* p's neighbors. The latter subcomponent offers *obstruction-free* progress [25]: termination is guaranteed whenever processes are scheduled to take multiple consecutive steps alone. As obstruction-free progress and starvation-free progress are both strictly weaker than wait-free progress, it is of independent interest that we are able to bootstrap a wait-free algorithm from subcomponents that aren't themselves wait-free.

1.4 Related Work

The closest recent contributions related to the current work are [13], and the follow-up work [18], which consider a related model, albeit a distinct one. The former provides a distributed algorithm for 3-coloring the ring, while the latter provides a distributed algorithm for $(\Delta+1)$-coloring graphs with maximum degree Δ. The two papers assume n asynchronous crash-prone processes occupying the n nodes of a reliable *and synchronous* network. That is, the communications remain synchronous, and a message emitted by a node u at round r reaches all nodes at distance d from u at round $r+d$. Moreover, no messages are lost, in the sense that a late-waking process will find all messages that passed through the node it occupies. Because it "decouples" the *computing layer* from the *communication layer*, this model is called DECOUPLED in [13].

The DECOUPLED model is stronger than the fully asynchronous model considered in this paper. In fact, [18] shows that, for every task (e.g., vertex-coloring, edge-coloring, maximal independent set, etc.), if there exists an algorithm for solving that task in the

LOCAL model in $t = \mathrm{O}(\text{polylog } n)$ rounds, then there exists an algorithm for solving the task in the DECOUPLED model in $\mathrm{O}(t)$-round. In contrast, some tasks that are trivial in the LOCAL and the DECOUPLED model become impossible in our fully asynchronous model, like 3-coloring C_3 (Property 3), or computing a maximal independent set (Property 1).

The model considered in this paper bears similarities with some models used in the context of *self-stabilization*. Many papers (see, e.g., [9, 10, 11, 12]) have addressed the design of self-stabilizing algorithms for 3-coloring the cycles, or for $(\Delta + 1)$-coloring graphs with maximum degree Δ. Self-stabilization assumes that the processes start in an arbitrarily bad state (all variables can be corrupted). The objective is to design algorithms which, starting from an arbitrary initial configuration, eventually compute a legal configuration (e.g., a configuration in which the colors assigned to the nodes form a proper coloring) whenever no failures occur during a sufficiently long period. In contrast, we assume an initial configuration in which variables are correctly set. However, we do not assume that the system will be failure-free during the execution of the algorithm, and the presence of crash-failures should not prevent the correct processing nodes from computing a solution. While 3-coloring the cycle C_n is possible in a self-stabilizing manner for all $n \geqslant 3$, k-coloring C_3 is impossible in our fully asynchronous model for $k < 5$ (Property 3).

2 Model and Observations

In this section, we first describe an asynchronous variant of the (synchronous) LOCAL model, which we will call the *partial immediate snapshot* model for reasons that will soon become apparent. The model can be viewed as a sort of asynchronous message-passing on a graph with a local broadcast communication primitive and instantaneous message delivery. Equivalently, it can be viewed as a shared-memory system where access to the shared memory is mediated by a graph; we adopt the latter approach in our description. We define what is a *round* in this model, what is the *round complexity* of an algorithm, what it means to be *wait-free*, and then we provide lower bounds on the round-complexity and on the range of colors for the problem of wait-free vertex-coloring the cycle.

2.1 Operational Model

The model is described for the cycle, but it can directly be extended to any network. Specifically, we consider asynchronous wait-free computing in the n-node cycle C_n, where the processes attached to each node exchange information between neighbors using single-writer/multiple-reader registers. Each process is a deterministic (infinite) state machine. All n processes are initially asleep; they may wake up at any time, and not all processes need to wake up, or to take enough steps to terminate correctly (i.e., processes are prone to fail-stop faults). Awakened processes proceed asynchronously, each with the objective of computing a color in $\{0, \ldots, 4\}$. We focus on wait-free tasks, i.e., where a process that takes enough steps is guaranteed to terminate, regardless of the scheduling of the other processes, so as to prevent deadlocks resulting from a process waiting for an event which will never occur because another process has crashed.

Just like for the standard coloring and renaming tasks, the only input given to a process p is its identifier X_p, which is an integer in the range $[0, \text{poly}(n)]$ that is unique in the system. We do not assume that the processes are aware of the length n of the cycle, nor even of an upper bound on n. Every process proceeds with a sequence of exchanges of information with its neighbors until some condition is satisfied by its local state, at which point it terminates and outputs a color obtained by applying some function to this local state.

Immediate snapshots. Let us first recall how communication works using a standard *immediate snapshot* communication primitive. In this model, the n processes $p_1, \ldots, p_n$ communicate through n single-writer/multiple-readers registers $R_1, \ldots, R_n$, initialized with an initial value $\perp$. Every process can read all registers, but each process p_i is the single writer in register R_i, $i \in \{1, \ldots, n\}$. Each process p_i goes through a (possibly infinite) sequence of write-read-update steps, where in each step it: (1) writes a value in register R_i, (2) reads the content of all registers, and (3) performs a private computation. Taken together, these three steps constitute an asynchronous *round* of process p_i.

Each of the rounds is instantaneous, but the time elapsed between two of p_i's rounds may be arbitrarily long. For example, process p_i may perform many rounds while p_j performs none, in which case p_i will read the same value in register R_j every time, possibly $\perp$ if p_j hasn't awakened yet. Conversely, in-between two consecutive rounds of p_i's, there may be faster processes that performed many writes in their registers.

The value read by a process in a register R_j is the one written by p_j in its most recent round. Multiple processes may perform a round at the same time. In this case, the system behaves as if each of these processes first wrote a value in its own register, then all processes read all registers, and, finally, they all performed their private computation. Note that distinct processes may be at distinct rounds of their execution. For example, one process may be just starting, i.e., in its first round, while another may already have been running for some time, and so be at a later round.

Local immediate shapshots. Our model simply adds a graph to the above, which mediates which registers a process is able to read. For example, in the cycle, a process only reads three registers: its own register, and the register of each of its two neighbors. We do not assume a coherent notion of left and right, i.e., each node assigns an arbitrary order to the registers of its neighbors.

In this paper, we do not assume that the registers are bounded. Nevertheless, our algorithms only manipulate a constant number of variables using $\mathrm{O}(\log n)$ bits each.

2.2 Schedules and complexity

In our model, an execution is entirely characterized by the code of each process, the graph (here, the cycle C_n), the input identifiers of each process, as well as the activation patterns of each process. The latter is captured by the collection of n increasing sequences $t_p^{(1)}, t_p^{(2)}, \ldots$ of positive integers, one for each process $p \in [n]$, where $t_p^{(i)}$ denotes the time in which process p performs its i-th round.

As multiple processes may be performing rounds simultaneously, let us introduce, for $t \geqslant 1$, the set $\sigma(t)$ of activated processes at time t. We set: $p \in \sigma(t) \iff \exists i \geqslant 1 : t_p^{(i)} = t$. The *schedule* of an execution is the infinite sequence $\sigma = \sigma(1), \sigma(2), \ldots$ An execution of a given algorithm on the cycle C_n is thus determined by the schedule σ and the input identifiers $(X_p)_{p\in[n]}$.

We will say that a process $p \in \sigma(t)$ is *working* if the stopping condition of p has not been fulfilled before time t. This leads us to define, for any schedule σ, the restricted schedule $\overline{\sigma}$ of working processes:

$$\overline{\sigma}(t) \coloneqq \{ p \in \sigma(t) \mid p \text{ has not fulfilled the stopping condition at time} \leqslant t-1 \}.$$

An execution *terminates* if there exists some time t^* such that $\overline{\sigma}(t) = \varnothing$ for all subsequent times $t \geqslant t^*$, i.e., if eventually all processes stop working. Note that a process stops working according to two possible scenarios: it may have been activated sufficiently many times for allowing it to fulfill the stopping condition, or it was not activated after some time t, before it fulfilled the stopping condition. The latter scenario models the crash of a process (at time t, or earlier in the execution). The *round complexity* of a terminating execution is then defined as

$$\max\{\, i \in \mathbb{N} \mid \exists p \in [n] : p \in \overline{\sigma}(t_p^{(i)}) \,\}.$$

The running time of an algorithm over the cycle C_n is then the supremum of the round complexity *for all possible executions*, i.e., all possible identifier assignments and schedules. Informally, the running time corresponds to the maximal number of times a process can be activated before it is guaranteed to terminate. An algorithm is then *wait-free* if its running time is finite.

2.3 Lower Bounds and Impossibility Results

We complete this section by a couple of observations on the round complexity, and on the range of colors used by wait-free vertex-coloring algorithms for the cycle. Before that, we formalize the fact that, as claimed in the introduction, the maximal independent set (MIS) problem cannot be solved in the asynchronous cycle. Solving the MIS problem requires that, at the end of every execution, (1) every node that terminates and outputs 0 is neighbor of at least one node that terminates and output 1, and (2) no two neighboring nodes that terminate output 1.

▶ **Property 1.** *For every $n \geqslant 3$, MIS in the n-node cycle C_n, cannot be solved wait-free in our model.*

Proof. The proof is by reduction from the *strong symmetry-breaking* (SSB) problem, which cannot be solved wait-free in the asynchronous shared-memory model (see [6, Theorem 11]). We show that if there were an algorithm solving MIS in the n-node cycle, then there would exist an algorithm for SSB in the n-node shared-memory system. Recall that SSB requires that (1) if all processes terminate, then at least one processes outputs 0, and at least one process outputs 1, and (2) in every execution, at least one process outputs 1. By way of contradiction, let $\mathcal{A}$ be an algorithm solving MIS in C_n. The n processes of shared-memory system can simulate the algorithm $\mathcal{A}$ as follows. Process p_i, $i = 0, \ldots, n-1$, simulates the execution of the algorithm $\mathcal{A}$ at the node of C_n with identifier i, and with neighbors the nodes with identifiers $i \pm 1 \bmod n$, which are simulated by processes $p_{i\pm1 \bmod n}$, respectively. Since the algorithm $\mathcal{A}$ solves MIS, it guarantees that, if all processes terminate, then at least one outputs 0, and at least one outputs 1. Moreover, in every execution of the algorithm $\mathcal{A}$, a node that terminates and is isolated (none of its neighbors terminated) must output 1, and a node that terminates and has a neighbor that terminates is such that either itself outputs 1, or at least one of its neighbors outputs 1. This guarantees that, in every execution, at least one process output 1. The two conditions for solving SSB are therefore fulfilled by simulating the algorithm $\mathcal{A}$, and thus $\mathcal{A}$ cannot exist. ◀

We now show that the round-complexity of our vertex-coloring algorithm is optimal.

▶ **Property 2.** *For every $k \geqslant 2$, the round-complexity of any wait-free algorithm for k-coloring the vertices of the n-node cycles C_n, $n \geqslant 3$, requires $\Omega(\log^* n)$ rounds in the state model.*

Proof. This directly follows from [26], which proved that, in synchronous and failure-free executions, i.e., $\sigma(t) = \{\, 1, \ldots, n \,\}$ for all $t \geqslant 1$, k-coloring the vertices of the n-node cycles C_n, requires $\Omega(\log^* n)$ rounds. ◀

Finally, we show that the range of colors used by our algorithm is optimal.

▶ **Property 3.** *If a wait-free algorithm k-colors all asynchronous cycles* $\mathcal{C} = \{ C_n \mid n \geqslant 3 \}$*, then* $k \geqslant 5$*.*

Proof. The partial shared-memory model in the cycle coincides with the standard shared-memory model when $n = 3$, since the cycle C_3 is complete. The result thus directly follows from the impossibility for $n = 3$ asynchronous processes to solve renaming wait-free using fewer than five names in an immediate snapshot shared-memory model [6, 14]. ◀

Note that Property 3 leaves open the possibility that, for specific values of n, fewer colors could be used to color the cycle C_n wait-free, the same way the lower bound $2n - 1$ on the number of names for renaming only holds when n is a power of a prime. However, a generic algorithm capable of proper coloring every cycle C_n, for all $n \geqslant 3$, must use at least 5 colors, as our algorithm does. Nevertheless, the shared-memory model with immediate snapshots does not coincide with our model when $n > 3$, and thus it may well be the case that fewer than 5 colors could be used for some specific values of $n > 3$, although we conjecture that this is not the case.

3 Asynchronously coloring the cycle in linear time

Here we develop asynchronous coloring algorithms, and show that **a)** they guarantee wait-free progress – i.e., a process will terminate in all executions, provided that it is activated sufficiently many times – and **b)** they are correct – i.e., the graph induced by the terminating processes is properly colored by the output colors of these processes. These algorithms have a poor runtime complexity of $\mathrm{O}(n)$ steps when compared to state-of-the-art algorithms in the LOCAL model, which terminate in $\mathrm{O}(\log^* n)$ synchronous rounds. We will achieve a similar runtime complexity in the next section by augmenting our wait-free algorithms with a mechanism that speeds up termination.

We first present an algorithm that uses a 6-color palette. Although it uses one extra color when compared to the theoretical minimum of 5 colors required to color the cycle C_3, this allows us to illustrate some of our main algorithmic ingredients. We then present another wait-free algorithm that colors any cycle using a 5-color palette. Some of the longer proofs of this section can be found in Appendix B.

3.1 Warm-up: using a palette of 6 colors

In Algorithm 1, we present a simple algorithm for wait-free coloring any cycle C_n ($n \geqslant 3$), using the six colors in the set $\{ (a, b) \in \mathbb{N} \times \mathbb{N} \mid a + b \leqslant 2 \}$. Given a process p, we denote by X_p its identifier, and by q and q' its two neighboring process in the cycle. We denote by $u \sim v$ the fact that processes u and v are neighbors in C_n. A process p, with neighbors q and q', is said to be *locally extremal* (with respect to the identifiers) if either $X_p > \max\{ X_q, X_{q'} \}$ or $X_p < \min\{ X_q, X_{q'} \}$.

Intuitively, Algorithm 1 guarantees that locally extremal processes quickly terminate, by sticking to one of the two components a_p or b_p of their color $c_p = (a_p, b_p)$ (Lemma 7). Termination then propagates throughout the cycle, due to the wait-free nature of the algorithm (Lemmas 6 and 7). Given an initial coloring of C_n provided by the nodes' identifiers, we will show that the worst-case convergence time of a process is determined by its distance to its nearest local extrema, which is bounded by $\mathrm{O}(\min\{ n, \max_p X_p - \min_q X_q \})$, which yields a linear convergence time.

Algorithm 1 6-coloring algorithm, code for process p with neighbors q and q'.

Input : $X_p \in \mathbb{N}$
Initially:
  $c_p = (a_p, b_p) \leftarrow (0, 0) \in \mathbb{N} \times \mathbb{N}$
Forever:
  write(X_p, c_p) and **read**$((X_q, c_q), (X_{q'}, c_{q'}))$ ▷ `local immediate snapshot`
  if $c_p \notin \{ c_q, c_{q'} \}$ **then** `return`(c_p)
  else
    $a_p \leftarrow \min \mathbb{N} \setminus \{ a_u \mid (u \sim p) \wedge (X_u > X_p) \}$
    $b_p \leftarrow \min \mathbb{N} \setminus \{ b_u \mid (u \sim p) \wedge (X_u < X_p) \}$

▶ **Theorem 4.** *In any execution of Algorithm 1 over the cycle C_n with a proper coloring provided by the values $(X_p)_{p \in [n]}$ given to the processes as input, we have:*
Termination: *every process terminates after having been activated at most $\lfloor 3n/2 \rfloor + 4$ times;*
6-color palette: *every process that terminates outputs a color in the set $\{ (a, b) \mid a + b \leqslant 2 \}$;*
Correctness: *the outputs properly color the graph induced by the terminating processes in C_n.*

The rest of the subsection is dedicated to the proof of Theorem 4. Recall that, in a schedule σ, a process $p \in \sigma(t)$ is *working* in t if it has not returned before t. Once a working process returns, it no longer partakes in the execution.

Notation. We will adopt the following notation for all algorithms throughout the paper. If x_p is a variable used by process p, we use $x_p(t)$ to denote the value of x_p *in p's memory*, at the *end* of time t, and we use $\widehat{x}_p(t)$ to denote the value of x_p *visible to p's neighbors* at the end of time t. Let $x_p(0)$ be given by the initialization of the algorithm, and let $\widehat{x}_p(0) = \perp$. By definition, we have

$$\widehat{x}_p(t) = \begin{cases} x_p(t-1) & p \in \overline{\sigma}(t) \\ \widehat{x}_p(t-1) & p \notin \overline{\sigma}(t) \end{cases} \tag{1}$$

▶ **Lemma 5.** *Let $t \geqslant 0$, and let $p \in \overline{\sigma}(t)$. We have $c_p(t) \notin \{ \widehat{c}_q(t) \mid q \sim p \}$, and process p returns at time t if and only if $c_p(t) = c_p(t-1)$.*

Proof. Process p does not update c_p when it returns, and so $c_p(t) = c_p(t-1)$ whenever p returns at time t. Let us then assume that $p \in \overline{\sigma}(t)$ does *not* return at time t, and let q be one of p's neighbors. If q has not yet been activated then $\widehat{c}_q(t) = \perp \neq \widehat{c}_p(t)$. If q has been already activated then, since the inputs form an initial proper coloring, we either have $X_p > X_q$ or $X_p < X_q$. In the former case, we have $a_p(t) \neq \widehat{a}_q(t)$, and in the latter case, we have $b_p(t) \neq \widehat{b}_q(t)$. Either way, we have $c_p(t) \neq \widehat{c}_q(t)$, and so $c_p(t) \neq c_p(t-1)$, since $c_p(t-1) = \widehat{c}_p(t) \in \{ \widehat{c}_q(t), \widehat{c}_{q'}(t) \}$ ◀

Lemma 5 provides us with an effective characterization of $\overline{\sigma}$: for every $t \geqslant 0$ and every $p \in [n]$,

$$p \in \overline{\sigma}(t) \iff \forall t' < t : \big(p \in \sigma(t') \implies c_p(t') \neq c_p(t'-1)\big). \tag{2}$$

The next lemma formalize the intuition that a process terminates fast, unless the execution is "very interleaved".

▶ **Lemma 6.** *Let p be a process that is working at times t_1 and $t_2 > t_1$, but is not activated at any time $t \in [t_1 + 1, t_2]$. If neither of p's neighbors is working in the time interval (t_1, t_2), then process p returns at time t_2.*

Proof. The result directly follows from Lemma 5, using the fact that $c_p(t_1) \notin \{\, \widehat{c}_q(t_1) \mid q \sim p \,\}$ and $\widehat{c}_p(t_2) = c_p(t_1)$. ◀

As the next lemma shows, a process cannot be prevented from returning by only one of its neighbors.

▶ **Lemma 7.** *Let process p be activated at times $t_1 < t_2 < t_3 < t_4$, but not at any other time $t \in (t_1, t_4)$. If $a_p(t_1) = a_p(t_2) = a_p(t_3) = a_p(t_4)$, and X_p is not a local minimum, then p returns at time at most t_4. The same holds if $b_p(t_1) = b_p(t_2) = b_p(t_3) = b_p(t_4)$ and X_p is not a local maximum.*

Note that, even though $X_p(t)$ remains constant throughout the execution, the public value $\widehat{X}_p(t)$ doesn't, as initially its value is $\perp$. To analyze executions of Algorithm 1, let us introduce the sets

$$N_p^+(t) \coloneqq \{\, q \sim p \mid \widehat{X}_q(t) > \widehat{X}_p(t) \,\} \text{ and } N_p^-(t) \coloneqq \{\, q \sim p \mid \widehat{X}_q(t) < \widehat{X}_p(t) \,\}.$$

We furthermore define the sets

$$A_p(t) \coloneqq \begin{cases} \bigcup_{q \in N_p^+(t)} \left(\widehat{A}_q(t) \cup \{\, \widehat{X}_q(t) \,\} \right) & p \in \overline{\sigma}(t) \\ A_p(t-1) & p \notin \overline{\sigma}(t) \end{cases} \tag{3}$$

and

$$B_p(t) \coloneqq \begin{cases} \bigcup_{q \in N_p^-(t)} \left(\widehat{B}_q(t) \cup \{\, \widehat{X}_q(t) \,\} \right) & p \in \overline{\sigma}(t) \\ B_p(t-1) & p \notin \overline{\sigma}(t) \end{cases} \tag{4}$$

where $A_p(0) = B_p(0) = \varnothing$, and where the sets $\widehat{A}_p(t), \widehat{B}_p(t)$ are defined according to Equation (1). The set $A_p(t)$ contains all processes that p has heard of at time t, and that are linked to p through a subpath of C_n where process identifiers are increasing. Symmetrically, the set $B_p(t)$ contains processes that p has heard of, and that are linked to p through a subpath where identifiers are decreasing.

▶ **Lemma 8.** *Let $t \in \mathbb{N}$, and let $p \in [n]$ be a process. For every $x \in A_p(t)$, we have $\widehat{X}_p(t) < x$, and, for every $x \in B_p(t)$, we have $\widehat{X}_p(t) > x$.*

▶ Remark 9. This will be used in the next section, where we present a procedure for speeding up Algorithm 2 by reducing the space of colors initially provided to the nodes thanks to their identifiers. On the other hand, the claim $\widehat{X}_p(t) > \max B_p(t)$ doesn't generalize under the same weaker condition.

In the case where X_p does not change, we can notice that $A_p(t)$ and $B_p(t)$ are increasing, inclusion-wise, with time. Moreover, the elements of $A_p(t)$ correspond to increasing identifiers X_q following a path from p (decreasing in the case of $B_p(t)$). Hence, $|A_p(t)|$ has a size bounded by the length of the longest path of increasing identifiers from p.

If a process $p \in \overline{\sigma}(t)$ fails to return in time t, the sets $A_p(t)$ and $B_p(t)$ help us compute its next color $c_p(t)$.

▶ **Lemma 10.** *For any time $t \geqslant 1$, if a process $p \in \overline{\sigma}(t)$ fails to return at time t, then:*
1. *if $\left|N_p^+(t)\right| \leqslant 1$, then $a_p(t) \equiv |A_p(t)| \bmod 2$;*
2. *if $\left|N_p^-(t)\right| \leqslant 1$, then $b_p(t) \equiv |B_p(t)| \bmod 2$.*

As a direct consequence of Lemma 10, we get the following.

▶ **Lemma 11.** *Let $t \geqslant 0$, and let $p \in [n]$ be non-extremal a process. If $p \in \overline{\sigma}(t)$, but p fails to return at time t, then we have $A_p(t) \neq A_p(t-1)$ or $B_p(t) \neq B_p(t-1)$.*

Proof. Using Lemma 10, if $A_p(t) = A_p(t-1)$ and $B_p(t) = B_p(t-1)$ then $c_p(t) = c_p(t-1)$, and so by Lemma 5 process p returns, a contradiction. ◀

This leads us to the following complexity bound for processes that are not local extrema. It relies on the distance of a process to its closest local extrema along monotone paths. Let q_i, $i = 0, \ldots, k+1$, be a set of distinct processes, excepted possibly $q_{k+1} = q_0$. Let us assume that these processes form a subpath of C_n, or possibly the entire cycle C_n if $q_{k+1} = q_0$. That is, $q_0 \sim q_1 \sim q_2 \cdots \sim q_k \sim q_{k+1}$. Let us assume that $X_{q_0} < X_{q_1}$ and $X_{q_k} < X_{q_{k+1}}$, but $X_{q_1} > X_{q_2} > \cdots > X_{q_k}$, i.e., process q_1 is locally maximal, process q_k is locally maximal, and for $i \in \{1, \ldots, k\}$, process q_i is at monotone distance $i-1$ from its closest local maximum q_1, and at monotone distance $k-i$ from its closest local minimum q_k.

▶ **Lemma 12.** *Let $p \in [n]$ be a non-extremal process, and let ℓ and ℓ' be the monotone distances from p to its closest extremal processes. Process p returns after at most $\min\{3\ell, 3\ell', \ell+\ell'\}+4$ activations.*

Proof. We know from Remark 9 that $A_p(t)$ is increasing with time, and that its size is bounded by ℓ. Thanks to Lemma 10, we have that $a_p(t)$ is determined by the size of $A_p(t)$. It follows that $a_p(t)$ changes at most $\ell+1$ times. Symmetrically, $b_p(t)$ changes at most ℓ' times. By Lemma 7, we get that a process p cannot be activated more than 3 times while keeping the same value for $a_p(t)$. It follows that process p can be activated at most $3\ell+4$ times before it returns. Symmetrically, p can be activated at most $3\ell'+4$ times before it returns. Finally, from Lemma 11, we get that p can be activated at most $\ell+\ell'+1$ times before it returns. ◀

This last results allows us to conclude.

Proof of Theorem 4. As a direct corollary of Lemma 7, that local extrema return after at most 4 steps: a maximum will maintain $a(t) = 0$, and a minimum, $b(t) = 0$. For the other nodes, Lemma 12 gives us the complexity, knowing that $\min\{\ell, \ell'\}$ is bounded by $\lfloor 3n/2 \rfloor$. ◀

▶ Remark 13. Lemma 12 states that the complexity of Algorithm 1 is linear in the length of the longest chain of processes $p_1 \sim p_2 \sim \cdots$ that is monotone for the identifiers, i.e., $X_{p_1} > X_{p_2} > \cdots$. Throughout this section, we have assumed that the processes start with their identifiers as input, and that each identifier is unique in the network, i.e., $X_p \neq X_q$ whenever $p \neq q$. Note however that Theorem 4 only requires that identifiers form a *proper coloring*, i.e., $X_p \neq X_q$ whenever $p \sim q$. In this case, the length of a monotone chain is bounded by the number of initial colors, and so is the convergence of Algorithm 1. In the Section 4, we exploit this property to dramatically accelerate our algorithms by dynamically adjusting the "identifiers" X_p themselves, using a modification of Cole and Vishkin's classic algorithm [17], initially designed for the PRAM model, but easily adapted to the LOCAL model. As we shall see, its adaptation to the asynchronous setting is more subtle.

3.2 Saving one color: wait-free 5-coloring the cycle

Here we present, in Algorithm 2, another wait-free coloring algorithm for the cycle, which only uses a palette of five colors. As already noted, when the graph is a clique, asynchronous coloring is identical to the renaming problem using an immediate snapshot communication primitive, which implies that asynchronously coloring the cycle C_3 *requires* at least a five-colors palette. Our algorithm is thus optimal in terms of colors for the class $\mathcal{C} = \{ C_n \mid n \geqslant 3 \}$ of all cycles.

Algorithm 2 5-coloring algorithm, code for process p with neighbors q and q'.

```
Input: X_p ∈ ℕ
Initially:
    a_p, b_p ← 0 ∈ ℕ
Forever:
    write(X_p, a_p, b_p) and read((X_q, a_q, b_q), (X_q', a_q', b_q'))  ▷ local imm.  snap.
    P⁺ ← { u ∈ { q, q' } | X_u > X_p }
    C⁺ ← { a_u | u ∈ P⁺ } ∪ { b_u | u ∈ P⁺ }
    C ← { a_q, b_q, a_q', b_q' }
    if a_p ∉ C then return(a_p)
    else if b_p ∉ C then return(b_p)
    else
        a_p ← min ℕ ∖ C⁺
        b_p ← min ℕ ∖ C
```

▶ **Theorem 14.** *In any execution of Algorithm 2 over the cycle C_n with a proper coloring provided by the values $(X_p)_{p\in[n]}$ given to the processes as input, we have:*
Termination: *every process terminates after having been activated at most $\mathrm{O}(n)$ times;*
5-color palette: *every process that terminates outputs a color in the set $\{ 0, \dots, 4 \}$;*
Correctness: *the outputs properly color the graph induced by the terminating processes in C_n.*

From the algorithm, we immediately deduce the following characterization of when a process returns a value.

▶ **Lemma 15.** *Let $t \geqslant 1$, and let $p \in \overline{\sigma}(t)$ be a process with neighbors q and q'. Let $C := \{ \widehat{a}_q(t), \widehat{b}_q(t), \widehat{a}_{q'}(t), \widehat{b}_{q'}(t) \}$. We have $b_p(t) \notin C$, and process p returns at time t if and only if $a_p(t-1) \notin C$ or $b_p(t-1) \notin C$.*

Note that, as a consequence of the previous lemma, $b_p(t) \neq b_p(t-1)$ unless $p \in \overline{\sigma}(t)$ returns at time t, and so Lemma 6 continues to hold for Algorithm 2.

Defining the sets $A_p(t)$ as we did for Algorithm 1, we get the following sufficient condition for a process to terminate.

▶ **Lemma 16.** *Suppose that process $p \in [n]$ is* not *a local minimum for the identifiers. If p is activated at times $t_1 < t_2 < t_3 < t_4$, and $A_p(t_1) = A_p(t_2) = A_p(t_3) = A_p(t_4)$, then p returns at time at most t_4.*

▶ **Lemma 17.** *Let $p \in [n]$ be a process that is* not *a local minimum for the identifiers, and let ℓ denote the monotone distance from p to the closest maximal process. Process p returns after at most $3\,\ell + 4$ activations.*

Proof. This is a direct consequence of the previous lemma: for p to keep working, its set $A_p(t)$ must increase at least every 4 activations. The claim follows. ◀

Proof of Theorem 14. Thanks to Lemma 17, processes that are *not* local minima return after a number of steps that is at most $\lfloor 3n/2 \rfloor + 4$. Local minima terminate at most one step after their two neighbors have terminated, i.e., in at most $3n + 8$ rounds. The proper coloring is an immediate consequence of Lemma 15. ◀

4 From Linear Time to Almost Constant Time

Here, we augment Algorithm 2 with a mechanism designed to reduce X_p, initially set to the identifier of the process. As the identifiers[3] will now be evolving through time, we will say that a process p, with neighbors q, q', is a local extremum at time $t \geqslant 1$ if $\widehat{X}_p(t) > \widehat{X}_q(t), \widehat{X}_{q'}(t)$. The resulting algorithm, displayed as Algorithm 3, 5-colors the cycle C_n in $\mathrm{O}(\log^* n)$ steps. Some of the longer proofs of this section can be found in Appendix B.

The intuition for Algorithm 3 is as follows. Every process p essentially runs Algorithm 2 unchanged, and stops whenever this algorithm terminates. However, in parallel, every process p updates its identifier X_p, initially equal to the identifier of p, *à la* Cole and Vishkin using a reduction function f defined hereafter. This helps to shorten long monotone chains of identifiers down to a constant length, speeding up the convergence of Algorithm 2. This addition to the algorithm is blocking, as, to maintain a proper coloring of the identifiers X_p (which is crucial for the wait-free coloration algorithm), every process p must wait for the approval of both its neighbors each time p wants to update its identifier, through the use of a local counter r_p which tracks the number of times process p tried to pick a smaller identifier. If all processes advance "almost synchronously", then they quickly (in $\mathrm{O}(\log^* n)$ steps) reach a stage where the remaining monotone chains of identifiers are all shorter than a constant $L \leqslant 10$. From then on, the algorithm behaves as Algorithm 2, and all processes terminate in $\mathrm{O}(L)$ steps, that is, in constant time. The crux of the proof is therefore to show that slow processes cannot delay the convergence of fast processes too much. Indeed, a slow process may delay other processes, but if it blocks them during too many iterations (with respect to the reduction of the identifiers X_p), then the system starts behaving as Algorithm 2, and neighboring processes actually quickly terminate. On the other hand, if a process is only "moderately slow", and allows its neighbors to make some progress on the reduction of their identifiers X_p, then other processes use this property for breaking symmetry, and they stop waiting for the slow process.

4.1 Reducing identifiers with deterministic coin-tossing

The considerable speedup achieved in comparison to Algorithm 2 relies on an identifier-reduction function $f : \mathbb{N} \times \mathbb{N} \to \mathbb{N}$, adapted from Cole and Vishkin's algorithm [17], defined as follows. For any natural number Z, we denote its binary decomposition by $Z = \sum_{k \in \mathbb{N}} Z_k 2^k$, and its length by $|Z| \coloneqq \lceil \log_2(Z + 1) \rceil$. Given two natural numbers X and Y, we then set

$$f(X, Y) = 2i + X_i \quad \text{where } i \coloneqq \min\{\, |X|, |Y| \,\} \cup \{\, k \in \mathbb{N} \mid X_k \neq Y_k \,\} \tag{5}$$

As $f(x, y) \leqslant 2|x| + 1 = \mathrm{O}(\log(x))$, one reaches a constant fixed point after $\mathrm{O}(\log^* n)$ iterate calls to f, which gives the following. Recall that, for $k \in \mathbb{N}$, the k-th iterate of a function $F : A \to A$ is recursively defined as $F^{(0)}(x) = x$ and, for $k \geqslant 1$, $F^{(k)} = F \circ F^{(k-1)}$.

[3] For simplicity, we continue to refer to $X_p(t)$ as process p's "identifier", even though it is now possible that $X_q(t) = X_p(t)$ for some other process $q \nsim p$.

▶ **Lemma 18.** *Let $F : [1, +\infty) \to [1, +\infty)$ be the function $x \mapsto F(x) = 2\lceil \log(x+1) \rceil + 1$. There exists $\alpha > 0$ such that, for every $x \geqslant 1$, there exists $t \leqslant \alpha \log^* x$ such that $F^{(t)}(x) < 10$.*

▶ **Lemma 19.** *Let $x, y \in \mathbb{N}$. If $x > y \geqslant 10$, then $f(x, y) < y$.*

Proof. Let $\ell = |y|$. By assumption, we have $\ell \geqslant 4$. If $\ell = 4$, then $f(x, y) \leqslant 2\ell + 1 = 9 < y$. If $\ell \geqslant 5$, then we have $y \geqslant 2^{\ell-1}$, and so $y - f(x, y) \geqslant 2^{\ell-1} - 2\ell - 1 > 0$, where the last inequality is because $2^z > 4z + 2$ whenever $z \geqslant 5$. ◀

The proper coloring maintained by the function f relies on the following Cole and Vishkin-like property.

▶ **Lemma 20.** *Let $x, y, z \in \mathbb{N}$. If $x > y > z$, then $f(x, y) \neq f(y, z)$.*

Proof. Let $f(x, y) = 2i^* + x_{i^*}$. For all $i < i^*$, $x_i = y_i$, and if $i^* < |y|$ then $x_i \neq y_i$. Suppose that $f(y, z) = f(x, y)$. Then $y_{i^*} = x_{i^*}$, and by the above $i^* \geqslant |y| \geqslant |z|$. In this case, $y_i = z_i$ for all $i < |y|$, and thus $y = z$, contradicting our assumption $y > z$. ◀

4.2 5-coloring the cycle in near-constant time

Algorithm 3 Fast 5-coloring algorithm, code for process p with neighbors q and q'.

```
Input : X_p ∈ ℕ
Initially:
    a_p, b_p, r_p ← 0 ∈ ℕ
Forever:
    write(X_p, r_p, a_p, b_p) and read((X_q, r_q, a_q, b_q), (X_q', r_q', a_q', b_q'))
    if a_p ∉ { a_q, b_q, a_q', b_q' } then return(a_p)
    else if b_p ∉ { a_q, b_q, a_q', b_q' } then return(b_p)
    else
        a_p ← min ℕ ∖ { a_u, b_u | (u ∼ p) ∧ (X_u > X_p) }
        b_p ← min ℕ ∖ { a_q, b_q, a_q', b_q' }
        if (r_p < ∞) ∧ (r_p ⩽ min{ r_q, r_q' }) then
            if min{ X_q, X_q' } < X_p < max{ X_q, X_q' } then
                r_p ← r_p + 1
                Y ← f(X_p, min{ X_q, X_q' })              ▷ f given in Equation (5)
                if Y < min{ X_q, X_q' } then X_p ← Y
            else
                r_p ← ∞
                if X_p < min{ X_q, X_q' } then
                    X_p ← min{ X_p, min(ℕ ∖ { f(X_q, X_p), f(X_q', X_p) }) }
```

▶ **Theorem 21.** *In any execution of Algorithm 3 over the cycle C_n with a proper coloring provided by the values $(X_p)_{p \in [n]}$ given to the processes as input:*

Termination: *every process terminates after having been activated at most $\mathrm{O}(\log^* n)$ times;*

5-color palette: *every process that terminates outputs a color in the set $\{0, \ldots, 4\}$;*

Correctness: *the outputs properly color the graph induced by the terminating processes in C_n.*

A crucial ingredient in the proof of correctness is to establish that the coloring provided by the evolving values of the local variables X_p, $p \in [n]$, is always proper throughout any execution.

▶ **Lemma 22.** *Let $p, q \in [n]$ be neighboring processes. For every $t \in \mathbb{N}$, if $\widehat{X}_p(t) \neq \bot$ then $\widehat{X}_p(t) \neq \widehat{X}_q(t)$.*

When discussing executions of Algorithm 3, we say that a process p is *blocked* at time t if p has not yet returned at time t and $r_p(t) = \widehat{r}_p(t) < \infty$. Since the value of X_p changes only if r_p increases, we have $X_p(t) = \widehat{X}_p(t)$ whenever process p is blocked at time t. A process p that *is not* blocked at time t, will write a new value for $\widehat{r}_p(t)$ at its next activation. Moreover, p writes a new value for $\widehat{X}_p(t)$ as well, unless p satisfies specific properties: X_p is a local maximum, X_p is a local minimum, or p has a neighbor q with $\widehat{X}_q < 10$. Note that, before its first activation, every process p is unblocked, as $r_p(0) = 0 \neq \widehat{r}_p(0) = \bot$.

Every process that takes sufficiently many non-blocked steps, namely $\Omega(\log^* n)$ steps, quickly reduces its identifier X_p until either X_p, or the identifier X_q of one of its neighbors q becomes smaller than 10. At this stage of the execution, monotone chains of identifiers will cease to evolve after an additional constant number of steps. Once the monotone chains of identifiers cease to evolve, the analysis developed in the previous section shows that processes terminate in a number of steps that is not larger than the length of monotone chains of identifiers, which is itself bounded by a constant $L \leqslant 10$. In other words, when all processes take $\Omega(\log^* n)$ non-blocked steps, they terminate in an additional $\mathrm{O}(1)$ steps.

In the following, we then focus on the case where the identifiers of the processes are still greater than 10, and we will show fast convergence is guaranteed even in the presence of blocked processes. Indeed, the main difficulty in proving Theorem 21 is to deal with blocked processes. Mainly, we show that a process quickly terminates whenever it is not blocked at *too many* steps.

▶ **Lemma 23.** *Let $p \in [n]$ be a process. For all $t \geqslant 1$, if $p \in \overline{\sigma}(t)$ and $\widehat{X}_p(t)$ is a local maximum in some time t, then $\widehat{X}_p(t')$ is a local maximum for all $t' \geqslant t$.*

Proof. Local maxima never update their identifiers; other processes might, but only to decrease them. The claim follows. ◀

▶ **Lemma 24.** *Let $p \in [n]$ be a process, and let q, q' be its two neighboring processes in the cycle. Let us assume that processes p and q are blocked at some time t_0, with $\widehat{r}_q(t_0) < \widehat{r}_p(t_0)$, $\widehat{X}_q(t_0) > \widehat{X}_p(t_0) > \widehat{X}_{q'}(t_0)$, and $\widehat{X}_p(t_0) \geqslant 10$. Additionally, let us assume that process q remains blocked throughout the whole time interval $[t_0, t_1)$ for some $t_1 > t_0$, but becomes activated and unblocked at time t_1. Then, one of the following claims holds:*

- *X_q is a local maximum at time t_1.*
- *If process q is activated again at some time $t_2 > t_1$, then X_p will become a local maximum as soon as p is activated at a time $t \geqslant t_2$.*

▶ **Lemma 25.** *Let $k \geqslant 1$, and let $q_0 \sim q_1 \sim \cdots \sim q_k$ be a sequence of $k+1$ distinct processes in the cycle. Let $t_0 \in \mathbb{N}$, and let $t_1 \in \mathbb{N} \cup \{\infty\}$ with $t_1 > t_0$. Let us assume that (1) for every $t \in [t_0, t_1)$, $q_0 \notin \overline{\sigma}(t)$, (2) processes $q_1, \ldots, q_k$ are blocked at time t_0, with $\widehat{r}_{q_0}(t_0) < \widehat{r}_{q_1}(t_0) < \cdots < \widehat{r}_{q_k}(t_0) < \infty$, and (3) $\widehat{X}_{q_k}(t_0) \geqslant 10$. Then, for every $i \in \{1, \ldots, k\}$, process q_i terminates after having been activated at most $3i+1$ times in the time interval $[t_0, t_1)$.*

▶ **Lemma 26.** *Let $p, q \in [n]$ be two neighboring processes. If X_q is a local maximum at time $t_0 \in \mathbb{N}$, and if $\widehat{r}_q(t_0) = \infty$, then p terminates after having been activated $\mathrm{O}(\log^* n)$ times during the time interval $[t_0, \infty)$.*

▶ **Lemma 27.** *Let $p \in [n]$ be a process. If p is blocked at every time $t \in [t_0, t_1)$, and if p takes 4 steps during that interval, then p takes up to $\mathrm{O}(\log^* n)$ steps in $[t_0, \infty)$ before terminating.*

We are now ready to show Theorem 21.

Proof of Theorem 21. For a process p, there are two possible paths, both leading to p returning:

1. Process p never gets blocked. By Lemma 18, if a process updates its identifier up to $\mathrm{O}(\log^* n)$ times, its identifier ends up in the interval $[0, 10]$. Therefore, after $\mathrm{O}(\log^* n)$ rounds, either X_p becomes a local maximum, or $X_p \leqslant 10$. In the first case, it stays a maximum by Lemma 23, its $a_p(t)$ remains constant, and p terminates after 4 rounds, thanks to Lemma 16. In the second case, it will stay at distance at most 10 from a local minimum. As the processes of this path will no longer change their X_-, Lemma 17 allows us to conclude.
2. Process p becomes blocked. This can happen after at most $\mathrm{O}(\log^* n)$ rounds (otherwise we would end up in the previous case). Lemma 27 ensures that at most $\mathrm{O}(\log^* n)$ rounds will happen before p returns.

This complete the proof that 5-coloring the asynchronous cycles C_n, $n \geqslant 3$, can be achieved in $O(\log^* n)$ rounds. ◀

5 Conclusion and future works

We have presented a wait-free distributed algorithm for proper coloring the n nodes of the asynchronous cycle C_n, for every $n \geqslant 3$. This algorithm performs in $\mathrm{O}(\log^* n)$ rounds, which is optimal, thanks to Linial's lower bound [26] that applies to the synchronous execution. The algorithm uses 5 colors to proper color any cycle C_n, $n \geqslant 3$, matches the minimum number 5 of colors required to properly color the asynchronous cycle C_3 [6, 14, 24]. Even if, for $n > 3$, the existence of a 3-coloring algorithm is not directly ruled out by [6, 14, 24], we conjecture that k-coloring the n-node cycle C_n requires $k \geqslant 5$ for every $n \geqslant 3$.

A natural extension of this work is to consider wait-free coloring arbitrary graphs. Note that, by the renaming lower bound, coloring graphs with maximum degree Δ requires a palette of at least $2\Delta + 1$ colors whenever $\Delta + 1$ is a power of a prime. This is because the shared memory model and the model in this paper coincide in the case of coloring the clique of $n = \Delta + 1$ nodes. We do not know if $2\Delta + 1$ colors suffice for properly coloring all graphs of maximum degree Δ in a wait-free manner. It is however easy to extend Algorithm 1 to graphs with maximum degree Δ, for every $\Delta \geqslant 2$, using a range of colors of size $O(\Delta^2)$ (see Appendix A). The running time of this algorithm may however be as large as the one of Algorithm 1, i.e., $O(n)$ rounds. In the synchronous setting, there is a $O(\Delta^2)$-coloring algorithm performing in $O(\log^* n)$ rounds [26] in any graph. However, the techniques used in the synchronous setting for reducing the number of colors from $O(\Delta^2)$ to $\Delta + 1$ (see [27]) seem hard to transfer to the asynchronous setting.

More generally, it would be interesting to characterize which classical graph problems studied in synchronous failure-free networks admit wait-free solutions for asynchronous networks, and which do not. And, for those solvable wait-free, what are their round-complexities? For instance, 5-coloring can be solved wait-free in the asynchronous cycle, in $O(\log^* n)$ rounds, while maximal independent set (MIS) cannot be solved at all in asynchronous cycles.

References

1 Yehuda Afek, Hagit Attiya, Arie Fouren, Gideon Stupp, and Dan Touitou. Long-lived renaming made adaptive. In *18th ACM Symposium on Principles of Distributed Computing (PODC)*, pages 91–103, 1999. doi:10.1145/301308.301335.

2 Dan Alistarh, Hagit Attiya, Rachid Guerraoui, and Corentin Travers. Early deciding synchronous renaming in $o(\log f)$ rounds or less. In *19th International Colloquium on Structural Information and Communication Complexity (SIROCCO)*, LNCS 7355, pages 195–206. Springer, 2012. doi:10.1007/978-3-642-31104-8_17.

3 Hagit Attiya, Amotz Bar-Noy, Danny Dolev, David Peleg, and Rüdiger Reischuk. Renaming in an asynchronous environment. *Journal of the ACM*, 37(3):524–548, July 1990. doi:10.1145/79147.79158.

4 Hagit Attiya, Armando Castañeda, Maurice Herlihy, and Ami Paz. Bounds on the step and namespace complexity of renaming. *SIAM Journal on Computing*, 48(1):1–32, 2019. doi:10.1137/16M1081439.

5 Hagit Attiya and Arie Fouren. Polynominal and adaptive long-lived ($2k$-1)-renaming. In *14th International Conference on Distributed Computing (DISC)*, LNCS 1914, pages 149–163. Springer, 2000. doi:10.1007/3-540-40026-5_10.

6 Hagit Attiya and Ami Paz. Counting-based impossibility proofs for set agreement and renaming. *Journal of Parallel and Distributed Computing*, 87:1–12, 2016. doi:10.1016/j.jpdc.2015.09.002.

7 Hagit Attiya and Jennifer Welch. *Distributed Computing*. Wiley Series on Parallel and Distributed Computing. John Wiley & Sons, Inc., Hoboken, NJ, USA, April 2004. doi:10.1002/0471478210.

8 Leonid Barenboim and Michael Elkin. *Distributed Graph Coloring: Fundamentals and Recent Developments*. Synthesis Lectures on Distributed Computing Theory. Morgan & Claypool Publishers, 2013. doi:10.2200/S00520ED1V01Y201307DCT011.

9 Leonid Barenboim, Michael Elkin, and Uri Goldenberg. Locally-iterative distributed $(\delta+1)$-coloring below szegedy-vishwanathan barrier, and applications to self-stabilization and to restricted-bandwidth models. In *37th ACM Symposium on Principles of Distributed Computing (PODC)*, pages 437–446, 2018. doi:10.1145/3212734.3212769.

10 Samuel Bernard, Stéphane Devismes, Maria Gradinariu Potop-Butucaru, and Sébastien Tixeuil. Optimal deterministic self-stabilizing vertex coloring in unidirectional anonymous networks. In *23rd IEEE International Symposium on Parallel and Distributed Processing (IPDPS)*, pages 1–8, 2009. doi:10.1109/IPDPS.2009.5161053.

11 Jean R. S. Blair and Fredrik Manne. An efficient self-stabilizing distance-2 coloring algorithm. *Theoretical Computer Science*, 444:28–39, 2012. doi:10.1016/j.tcs.2012.01.034.

12 Lélia Blin, Laurent Feuilloley, and Gabriel Le Bouder. Brief Announcement: Memory Lower Bounds for Self-Stabilization. In Jukka Suomela, editor, *33rd International Symposium on Distributed Computing (DISC 2019)*, volume 146 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 37:1–37:3, Dagstuhl, Germany, 2019. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik. doi:10.4230/LIPIcs.DISC.2019.37.

13 Armando Castañeda, Carole Delporte-Gallet, Hugues Fauconnier, Sergio Rajsbaum, and Michel Raynal. Making local algorithms wait-free: the case of ring coloring. *Theory of Computing Systems*, 63(2):344–365, 2019. doi:10.1007/s00224-017-9772-y.

14 Armando Castañeda and Sergio Rajsbaum. New combinatorial topology bounds for renaming: the lower bound. *Distributed Computing*, 22(5-6):287–301, 2010. doi:10.1007/s00446-010-0108-2.

15 Armando Castañeda and Sergio Rajsbaum. New combinatorial topology bounds for renaming: the upper bound. *Journal of the ACM*, 59(1):3:1–3:49, 2012. doi:10.1145/2108242.2108245.

16 Armando Castañeda, Michel Raynal, and Julien Stainer. When and how process groups can be used to reduce the renaming space. In *16th International Conference on the Principles of Distributed Systems (OPODIS)*, LNCS 7702, pages 91–105. Springer, 2012. doi:10.1007/978-3-642-35476-2_7.

17 Richard Cole and Uzi Vishkin. Deterministic coin tossing with applications to optimal parallel list ranking. *Information and Control*, 70(1):32–53, July 1986. `doi:10.1016/S0019-9958(86)80023-7`.

18 Carole Delporte-Gallet, Hugues Fauconnier, Pierre Fraigniaud, and Mikaël Rabie. Distributed computing in the asynchronous LOCAL model. In *21st International Symposium on Stabilization, Safety, and Security of Distributed Systems (SSS)*, LNCS 11914, pages 105–110. Springer, 2019. `doi:10.1007/978-3-030-34992-9_9`.

19 Michael J. Fischer, Nancy A. Lynch, and Michael S. Paterson. Impossibility of distributed consensus with one faulty process. *Journal of the ACM*, 32(2):374–382, April 1985. `doi:10.1145/3149.214121`.

20 Pierre Fraigniaud, Marc Heinrich, and Adrian Kosowski. Local conflict coloring. In *57th IEEE Symposium on Foundations of Computer Science (FOCS)*, pages 625–634, 2016. `doi:10.1109/FOCS.2016.73`.

21 Mohsen Ghaffari, Juho Hirvonen, Fabian Kuhn, and Yannic Maus. Improved distributed Δ-coloring. *Distributed Computing*, 34(4):239–258, 2021. `doi:10.1007/s00446-021-00397-4`.

22 Mohsen Ghaffari, Juho Hirvonen, Fabian Kuhn, Yannic Maus, Jukka Suomela, and Jara Uitto. Improved distributed degree splitting and edge coloring. *Distributed Computing*, 33(3-4):293–310, 2020. `doi:10.1007/s00446-018-00346-8`.

23 Magnús M. Halldórsson, Fabian Kuhn, Yannic Maus, and Tigran Tonoyan. Efficient randomized distributed coloring in CONGEST. In *53rd ACM Symposium on Theory of Computing (STOC)*, pages 1180–1193, 2021. `doi:10.1145/3406325.3451089`.

24 Maurice Herlihy and Nir Shavit. The topological structure of asynchronous computability. *Journal of the ACM*, 46(6):858–923, November 1999. `doi:10.1145/331524.331529`.

25 Maurice Herlihy and Nir Shavit. On the Nature of Progress. In Antonio Fernàndez Anta, Giuseppe Lipari, and Matthieu Roy, editors, *Principles of Distributed Systems*, pages 313–328, Berlin, Heidelberg, 2011. Springer. `doi:10.1007/978-3-642-25873-2_22`.

26 Nathan Linial. Locality in distributed graph algorithms. *SIAM Journal on Computing*, 21(1):193–201, 1992. `doi:10.1137/0221015`.

27 Yannic Maus. Distributed Graph Coloring Made Easy. In *Proceedings of the 33rd ACM Symposium on Parallelism in Algorithms and Architectures*, SPAA '21, pages 362–372, New York, NY, USA, July 2021. Association for Computing Machinery. `doi:10.1145/3409964.3461804`.

28 Moni Naor and Larry J. Stockmeyer. What can be computed locally? *SIAM Journal on Computing*, 24(6):1259–1277, 1995. `doi:10.1137/S0097539793254571`.

29 David Peleg. *Distributed computing: a locality-sensitive approach.* Society for Industrial and Applied Mathematics, USA, 2000.

30 Václav Rozhon and Mohsen Ghaffari. Polylogarithmic-time deterministic network decomposition and distributed derandomization. In *52nd ACM Symposium on Theory of Computing (STOC)*, pages 350–363, 2020. `doi:10.1145/3357713.3384298`.

A Coloring General Graphs

It is possible to extend Algorithm 1 to connected graphs with maximum degree Δ, for every $\Delta \geqslant 2$ (see Algorithm 4). By construction, every process running Algorithm 4 returns a color taken in the set

$$\{\,(a,b) \mid a+b \leqslant \Delta\,\},$$

of cardinality $\frac{(\Delta+1)(\Delta+2)}{2} = \mathrm{O}(\Delta^2)$. The proof of correctness for Algorithm 4 uses the same arguments as for establishing the correctness of Algorithm 1. In particular, a process cannot run forever whenever its identifier becomes a local extremum among the identifiers of its active neighbors, which guarantee that every process eventually terminates.

Algorithm 4 $O(\Delta^2)$-coloring algorithm for general graphs, code for process p with neighbors $q_1, \ldots, q_k$, $k \leqslant \Delta$.

Input : $X_p \in \mathbb{N}$
Initially:
  $c_p = (a_p, b_p) \leftarrow (0, 0) \in \mathbb{N} \times \mathbb{N}$
Forever:
  write(X_p, c_p) and **read**$((X_{q_1}, c_{q_1}), \ldots, (X_{q_k}, c_{q_k}))$ ▷ `immediate snapshot`
  if $c_p \notin \{ c_{q_1}, \ldots, c_{q_k} \}$ **then** `return`(c_p)
  else
    $a_p \leftarrow \min \mathbb{N} \setminus \{ a_u \mid u \sim p, X_u > X_p \}$
    $b_p \leftarrow \min \mathbb{N} \setminus \{ b_u \mid u \sim p, X_u < X_p \}$

B Technical proofs

B.1 Proofs of Section 3

Proof of Lemma 8. We proceed by induction on $t \in \mathbb{N}$. For $t = 0$, the claim is vacuously true, as $A_p(t) = B_p(t) = \varnothing$. For the inductive step, we suppose the claim holds for $t = 0, \ldots, T$, and we show that it holds for $t = T + 1$. If $p \notin \overline{\sigma}(T + 1)$ then we have $\widehat{X}_p(T + 1) = \widehat{X}_p(T)$ and $A_p(T + 1) = A_p(T)$, Thus the claim holds by induction. Let us assume that $p \in \overline{\sigma}(T + 1)$, and let $x \in A_p(t)$. By Equation (3) and the assumption $p \in \overline{\sigma}(T + 1)$, there exists $q \in N_p^+(T + 1)$ such that either $x = \widehat{X}_q(T + 1)$ or $x \in \widehat{A}_q(T + 1)$. In the former case, we have $\widehat{X}_p(T + 1) < Y$ by the definition of N_p^+. In the latter case, there must exist some time $t' \leqslant T$, with $q \in \overline{\sigma}(t')$, for which $\widehat{X}_q(T + 1) = X_q(t')$ and $\widehat{A}_q(T + 1) = A_q(t')$. Since $\tau \leqslant T$, we get that that $\widehat{X}_q(t') < x$, thanks to the induction hypothesis. Also, since the value of $X_q(t)$ is stable throughout the execution, we have $X_q(t') = X_q(t' - 1) = \widehat{X}_q(t') < x$. Therefore $\widehat{X}_q(T + 1) < x$, and, since $q \in N_p^+(T + 1)$, we have $\widehat{X}_p(T + 1) < \widehat{X}_q(T + 1) < x$, which proves the claim.

The proof is symmetric for $x \in B_p(t)$. ◀

Proof of Lemma 7. We establish the result for the case where X_p is not a local minimum and $a_p(t_1) = a_p(t_2) = a_p(t_3) = a_p(t_4)$. The proof uses the same arguments with local maxima and $b_p(t_1) = b_p(t_2) = b_p(t_3) = b_p(t_4)$.

Suppose that process p fails to return at time t_1; we consider two cases.

If p is a local maximum, then we have $\widehat{a}_p(t) = 0$ for all t. Moreover, if some process $q \sim p$ is working in the interval $[t_1, t_3]$, then $a_q(t_3) \neq 0$. Furthermore, we have $c_p(t_3) \neq \widehat{c}_q(t_3)$ by Lemma 5. In this case, either process q works in the interval $[t_3 + 1, t_4]$, and $\widehat{a}_q(t_4) \neq 0$, or it does not work in this interval, and $\widehat{c}_q(t_4) = \widehat{c}_q(t_3)$. Either way, we get $\widehat{c}_p(t_4) \neq \widehat{c}_q(t_4)$, and thus p returns at time at most t_4. Suppose now process p has a neighbor q' that is inactive in the interval $[t_1, t_3]$. If p's other neighbor q is *not* working in the interval $[t_1 + 1, t_2]$, then p returns at time t_2 by Lemma 6; if, on the other hand, q is working in this interval, then we have $a_q(t_2) \neq 0$, and, as above, process p returns at time at most t_3.

If process p is *not* a local maximum, then it has a neighbor q' with $X_{q'} > X_p$. If we had $\widehat{a}_p(t) = \widehat{a}_{q'}(t)$, in any $t \in \{ t_2, t_3, t_4 \}$, then $a_p(t)$ would be switched, which contradicts the lemma assumptions; hence we have $\widehat{c}_p(t) \neq \widehat{c}_{q'}(t)$ for $t = t_2, t_3, t_4$. Suppose p fails to return in t_2. In this case, as before, either the other neighbor q of p is working in the interval $[t_2 + 1, t_3]$, and so $a_q(t_3) \neq \widehat{a}_p(t_4)$; or q is inactive in that interval and p returns at time t_3. Either way, process p returns at the latest at time t_4. ◀

Proof of Lemma 10. We only treat the case $\left|N_p^+(t)\right| \leqslant 1$, as the other case is symmetric. First, note that for any process q, $\widehat{X}_q(t)$ is equal to either $\perp$ or X_q. In the former case, process q is still inactive in time t, and thus $A_q(t) = \varnothing$. As a consequence, thanks to Lemma 8, we have $X_q \notin A_q(t)$ for all $t \in \mathbb{N}$.

Given $p \in \overline{\sigma}(t)$, we proceed by induction over $|A_p(t)|$ by treating two base cases $|A_p(t)| = 0$, $|A_p(t)| = 1$, and then the general case. For the base cases, as $p \in \overline{\sigma}(t)$, we have $|A_p(t)| = 0$ if and only if $N_p^+(t) = \varnothing$, which corresponds to p being a local maximum among its neighbors awaken at time t. In this case, if p fails to return, then the algorithm enforces $a_p(t) = 0$, as desired, which gives the base case of the induction. If $|A_p(t)| = 1$, then the set $N_p^+(t)$ is a singleton. Let $\{ q \} = N_p^+(t)$. We have $A_p(t) = \{ \widehat{X}_q(t) \} = \{ X_q \}$, and $\widehat{A}_q(t) \in \{ \varnothing, \{ X_q \} \}$. The set $\widehat{A}_q(t)$ is therefore empty, i.e., $\widehat{a}_q(t) = 0$, and thus the algorithm enforces $a_p(t) = 1 = |A_p(t)|$.

For the inductive case, let us assume that the claim is true for $|A_p(t)| = 0, \ldots, T$ with $T \geqslant 1$, and let us show that it still holds for $|A_p(t)| = T + 1 \geqslant 2$. Here again, the set N_p^+ has to be a singleton, say $\{ q \}$, and so we have $a_p(t) = 1 - \widehat{a}_q(t)$, and $A_p(t) = \{ X_q \} \cup \widehat{A}_q(t)$, with $X_q \notin \widehat{A}_q(t)$. Thus $\left|\widehat{A}_q(t)\right| = T$, and there was an earlier time $t' < t$ where $q \in \overline{\sigma}(t')$ failed to return, and $\widehat{A}_q(t) = A_q(t')$. Since $X_p < X_q$, $|N_q^+(t')| \neq 2$, and so $a_q(t') \equiv T \pmod 2$ by the induction hypothesis. Thus $a_p(t) = 1 - a_q(t') \equiv T + 1 \pmod 2$, which completes the proof of the claim. ◀

Proof of Lemma 16. We first show the following: if $p \in \overline{\sigma}(t)$ fails to return at time $t \geqslant 1$, then

$$a_p(t) = 0 \iff |A_p(t)| \equiv 0 \bmod 2. \tag{6}$$

We proceed by induction on $|A_p(t)|$. If $|A_p(t)| = 0$, then process p is a local maximum among its active neighbors, and so in Algorithm 2 we have $C^+ \leftarrow \varnothing$, which implies $a_p(t) = 0$. For the inductive step, suppose the result true for $|A_p(t)| = k$, and suppose that $|A_p(t)| = k + 1$. Since process p is assumed to be non-minimal, it has one neighbor q with $\widehat{X}_q(t) > \widehat{X}_p(t)$ and $|\widehat{A}_q(t)| = k$, and we have $a_p(t) = \min \mathbb{N} \smallsetminus \{ \widehat{a}_q(t), \widehat{b}_q(t) \}$.

If k is even, then by inductive assumption we have $\widehat{a}_q(t) = 0$, and so $a_p(t) \neq 0$. Otherwise, k is odd, and by inductive assumption we have $\widehat{a}_q(t) > 0$. In the code of Algorithm 2, we have $C^+ \subseteq C$, and so for any process $u \in [n]$ and time $\tau \geqslant 0$ we have $b_u(\tau) \geqslant a_u(\tau)$. Thus in particular we have $\widehat{b}_q(t) \geqslant \widehat{a}_q(t) > 0$, and therefore $a_p(t) = 0$.

For the main claim, let $\ell \coloneqq |A_p(t_1)| = |A_p(t_2)| = |A_p(t_3)| = |A_p(t_4)|$. If ℓ is even, then $a_p(t) = 0$ for all $t \in [t_1, t_4]$. Reasoning as in Lemma 7, if p still hasn't returned by time t_4, then we have $|A_p(t_3)| = |A_q(t_3)| - 1 = |A_{q'}(t_3)| + 1$ without loss of generality. Then if neither q nor q' is activated in the interval $[t_3 + 1, t_4]$, p terminates by Lemma 6. Otherwise, using again the fact that $b_u(\tau) \geqslant a_u(\tau)$ for any process u and time τ, we have $\widehat{a}_p(t_4) = 0 < \min\{ \widehat{a}_q(t_4), \widehat{b}_q(t_4), \widehat{a}_{q'}(t_4), \widehat{b}_{q'}(t_4) \}$, and so p returns in t_4.

If ℓ is odd, we suppose without loss of generality that $X_q > X_p > X_{q'}$. We have $\widehat{a}_p(\tau) > 0$ for all $\tau \in [t_2, t_4]$, and reasoning again as in Lemma 7, by time t_4 we have $\widehat{a}_{q'}(t_4) = 0$, and p terminates if it is still active. ◀

B.2 Proofs of Section 4

Proof of Lemma 22. We show the following: for every $t \in \mathbb{N}$, $X_p(t) \notin \{ X_q(t), \widehat{X}_q(t) \}$, proceeding by induction. The case $t = 0$ results from the initial proper coloring of the identifiers.

For the induction, suppose the claim holds for $t = 0, \ldots, T$. If $p, q \notin \overline{\sigma}(T + 1)$, then nothing changed, and the claim still holds for $t = T + 1$.

Suppose $p, q \in \overline{\sigma}(T+1)$. If $r_q(T+1) = r_q(T)$, the claim immediately follows, as does it if $X_q(T+1) = X_q(T)$. Otherwise, by assumption we either have $X_q(T) > X_p(T)$, or the opposite. If the former, $X_q(T+1) = f(X_q(T), X_p(T)) < X_p(T)$, and by Lemma 20 we have $X_q(T{+}1) \notin \{ X_p(T), X_p(T{+}1) \}$, and so $X_p(T{+}1) \notin \{ \widehat{X}_q(T{+}1), X_q(T{+}1) \}$. Otherwise, $X_q(T) < X_p(T)$; if q is a local minimum in $T+1$, then $X_q(T+1) \neq f(X_p(T), X_q(T))$, and the claim follows from $X_p(T+1) \in \{ X_p(T), f(X_p(T), X_q(T)) \}$. If q is *not* a local minimum, then $X_q(T+1) = f(X_q(T), Z) < Z$ for some $Z < X_q(T)$; here again, the claim follows from Lemma 20.

Finally, suppose $p \in \overline{\sigma}(T+1)$ and $q \notin \overline{\sigma}(T+1)$. If $X_p(T+1) = X_p(T)$, then the claim still holds. Otherwise, we have $r_p(T) < r_p(T+1) \leqslant \infty$, and $X_p(T+1) < X_p(T)$. Process p is then not a local maximum in $T+1$, and the algorithm guarantees $X_p(T+1) < \widehat{X}_q(T+1)$. If $X_q(T+1) = \widehat{X}_q(T+1)$, and in particular if $r_q(T+1) = \widehat{r}_q(T+1)$, and the claim holds.

Suppose then that $r_q(T{+}1) < \widehat{r}_q(T{+}1)$, and let t_0 be the earliest time when $r_q(t_0) = r_q(T)$, such that $\widehat{r}_q(t_0) = \widehat{r}_q(T{+}1)$. Process q takes no steps in the interval $(t_0, T{+}1]$, and because r_q increases in t_0, we have $\widehat{r}_q(t_0) \leqslant \widehat{r}_p(t_0)$. Thus $\widehat{r}_q(T+1) \leqslant \widehat{r}_p(t_0) \leqslant \widehat{r}_p(T+1)$. Since r_p increases in $T+1$, we have $\widehat{r}_p(T+1) \leqslant \widehat{r}_q(T+1)$, and thus

$$\widehat{r}_q(t_0) = \widehat{r}_p(t_0) = \widehat{r}_q(T+1) = \widehat{r}_p(T+1),$$

i.e., $\widehat{r}_p(t)$ is constant for $t \in [t_0, T{+}1]$, and as a consequence, $\widehat{X}_p(T{+}1) = \widehat{X}_p(t_0)$. From here, we proceed as in the previous case: $X_q(T{+}1) = X_q(t_0)$ was computed with q seeing $\widehat{X}_p(t_0) = \widehat{X}_p(T+1)$, and, since $X_p(T+1)$ was computed with p seeing $\widehat{X}_q(T+1) = \widehat{X}_q(t_0)$, we have indeed $X_p(T+1) \notin \{ X_q(T+1), \widehat{X}_q(T+1) \}$. ◀

Proof of Lemma 24. Since $p \sim q$, and since p is blocked at time t_0, we have $\widehat{r}_p(t_0) = \widehat{r}_q(t_0) + 1$. Moreover, as $\widehat{X}_p(t_0) \geqslant 10$, the fact that $\widehat{X}_p(t_0)$ is not a local minimum means that $\widehat{r}_{q'}(t_0) \geqslant \widehat{r}_p(t_0)$. Otherwise, by Lemma 19, X_p would be smaller than $X_{q'}$. In particular, process p remains blocked as long as process q is itself blocked.

Now, suppose that $r_q(t_1) \neq \widehat{r}_q(t_1)$. As processes p, q are blocked until t_1, we have $\widehat{X}_q(t_1) = \widehat{X}_q(t_0)$ and $\widehat{X}_p(t_1) = \widehat{X}_p(t_0)$, so $\widehat{X}_q(t_1) > \widehat{X}_p(t_1)$, and q is not a local minimum in t_1. The case $r_q(t_1) = \infty$ thus corresponds to $\widehat{X}_p(t_0)$ being a local maximum at time t_1. If $r_q(t_1) < \infty$, then $r_q(t_1) = \widehat{r}_q(t_0){+}1$, and since $\widehat{X}_p(t_1) = \widehat{X}_p(t_0) \geqslant 10$, we get $X_q(t_1) = f(\widehat{X}_q(t_0), \widehat{X}_p(t_0)) < \widehat{X}_p(t_0)$ by Lemma 19.

Finally, suppose then that q is next activated at time t_2, and that p is activated at time $t \geqslant t_2$.Note that as long as q does not get activated again, p remains blocked, as it did not see the update on r_q. Moreover, as $X_q(t_1)$ is not a local maximum, $X_q(t_2) < \widehat{X}_p(t_1) = \widehat{X}_p(t_0)$. Then process p sees $\widehat{X}_p(t_0)$ to be a local maximum, and since it is no longer blocked by q we have $r_p(t) = \infty$. ◀

Proof of Lemma 25. Under the hypotheses of the lemma, processes $q_1, \ldots, q_k$ remain blocked throughout the time interval $[t_0, t_1 - 1]$, and we have $\widehat{X}_{q_0}(t) > \widehat{X}_{q_1}(t) > \cdots > \widehat{X}_{q_k}(t)$ whenever $t_0 \leqslant t < t_1$. By the same arguments as the ones used in the previous section for establishing Lemma 10, the sign of $a_{q_i}(t)$ is determined by the sign of the variables $a_{q_0}, \ldots, a_{q_{i-1}}$ throughout the time interval $[t_0, t]$. In particular, the sign of $\widehat{a}_{q_i}(t)$ stabilizes after q_i has been activated at most $3\,i$ times, and thus process i itself terminates after having been activated at most $(3i+1)$ times. ◀

Proof of Lemma 26. Let $t_1, t_2, \ldots$ be the consecutive steps taken by process p in the interval $[t_0, \infty)$, that is, for every $k \geqslant 1$, p is inactive during the whole time (t_k, t_{k+1}). Note that, since $\widehat{r}_q(t_0) = \infty$, we have $\widehat{a}_q(t) = 0$ for all $t \geqslant t_0$, and so $\widehat{a}_p(t) > 0$ for all $t \geqslant t_2$, as process q will remain a local maximum forever.

Pick $k \geqslant 2$. If p is *not* a local minimum at any point in $[t_k, t_{k+3}]$, then by Lemma 16 it terminates in t_{k+3} at the latest. Conversely, if p *is* a local minimum throughout the same interval, then we have repeatedly $\widehat{a}_p(t_i) \in \{\, \widehat{a}_{q'}(t_i), \widehat{b}_{q'}(t_i) \,\}$, $i = k, \ldots, k+3$. This implies that $\widehat{a}_{q'}$ is positive during that interval, otherwise p and q' would eventually stop having conflicts. By the same argument, process q' terminates in a constant number of activations, and so do process p. Therefore, for process p *not* to terminate the relative order of $\widehat{X}_p$ and $\widehat{X}_{q'}$ must switch every $\mathrm{O}(1)$ steps. Thus, every time p takes $\mathrm{O}(1)$ steps and fails to return, it must be the case that $\widehat{X}_p$ has decreased. As argued before, this can happen at most $\mathrm{O}(\log^* n)$ times before either $X_p \leqslant 10$ or $X_{q'} \leqslant 10$, at which point convergence happens in a bounded number of steps. ◀

Proof of Lemma 27. Since process p is blocked, a direct induction shows that p lies somewhere within a monotone chain of identifiers, as described in Lemma 25. That is, there is a chain of distinct adjacent processes

$$q_{-k-1} \sim q_{-k} \sim \cdots \sim q_{-1} \sim q_0 \sim q_1 \sim \cdots \sim q_\ell,$$

with $q_0 = p$, and $k, \ell \geqslant 0$, where, for every $i \in [-k, \ell]$, $\widehat{r}_{q_i}(t_0) = \widehat{r}_p(t_0) + i$, and either $\widehat{r}_{q_{-k-1}}(t_0) = \perp$ (in which case $k = \widehat{r}_p(t_0)$) or $\widehat{r}_{q_{-k-1}}(t_0) = R - k - 1$ (in which case $k < \widehat{r}_p(t_0)$. Moreover, all processes $q_{-k}, \ldots, q_\ell$ are blocked at time t_0, and process q_{-k-1} is not blocked at time t_0.

We now distinguish two cases, depending on whether $k = 0$ or not. If $k = 0$, then, by Lemma 25, process p terminates after taking 4 steps within the time interval $[t_0, t_1)$. If $k > 0$, then process q_{-1} is blocked; if process q_{-1} remains blocked while process p takes $3k+1$ steps, then, by Lemma 25, p terminates. The same holds if q_{-1} takes a single non-blocked step. If process q_{-1} ever becomes unblocked, and takes another step, then we meet the assumptions of Lemma 24, and either of X_p or $X_{q_{-1}}$ become a local maximum. If X_p becomes a local maximum, then it terminates in $\mathrm{O}(1)$ steps. If $X_{q_{-1}}$ becomes a local maximum, then, by Lemma 26, process p terminates in $\mathrm{O}(\log^* n)$ steps. ◀