Optimal Collision/Conflict-free Distance-2 Coloring in Synchronous Broadcast/Receive Tree Networks

Davide Frey, Hicham Lakhlef, Michel Raynal

Optimal Collision/Conflict-free Distance-2 Coloring in Synchronous Broadcast/Receive Tree Networks

Davide Frey[†] Hicham Lakhlef[‡] Michel Raynal[‡,⋆]

[†] INRIA Bretagne Atlantique, Rennes, France
[‡] IRISA, Université de Rennes, France
[⋆] Institut Universitaire de France
hicham.lakhlef@irisa.fr raynal@irisa.fr

Tech Report # 2030 (19 pages), December 2015

IRISA, University of Rennes 1 (France)

Abstract

This article is on message-passing systems where communication is (a) synchronous and (b) based on the "broadcast/receive" pair of communication operations. "Synchronous" means that time is discrete and appears as a sequence of time slots (or rounds) such that each message is received in the very same round in which it is sent. "Broadcast/receive" means that during a round a process can either broadcast a message to its neighbors or receive a message from one of them. In such a communication model, no two neighbors of the same process, nor a process and any of its neighbors, must be allowed to broadcast during the same time slot (thereby preventing message collisions in the first case, and message conflicts in the second case). From a graph theory point of view, the allocation of slots to processes is know as the distance-2 coloring problem: a color must be associated with each process (defining the time slots in which it will be allowed to broadcast) in such a way that any two processes at distance at most 2 obtain different colors, while the total number of colors is "as small as possible".

The paper presents a parallel message-passing distance-2 coloring algorithm suited to trees, whose roots are dynamically defined. This algorithm, which is itself collision-free and conflict-free, uses $\Delta + 1$ colors where Δ is the maximal degree of the graph (hence the algorithm is color-optimal). It does not require all processes to have different initial identities, and its time complexity is $O(d\Delta)$, where d is the depth of the tree. As far as we know, this is the first distributed distance-2 coloring algorithm designed for the broadcast/receive round-based communication model, which owns all the previous properties.

Keywords: Broadcast/receive communication, Collision, Conflict, Distance-2 graph coloring, Message-passing, Network traversal, Synchronous system, Time slot assignment, Tree network, Wireless network.

1 Introduction

Graph coloring is an important problem related to (optimal) resource allocation, mainly used to establish an optimal order in which resources have to be allocated to processes [13, 24]. The distance-k coloring problem consists in assigning colors to the vertices of the graph such that no two vertices at distance at most k have the same color. Let us remember that minimum distance-1 vertex coloring is an NP-complete problem [16].

Coloring is (with leader election [1] and renaming [2, 10]) one of the most important *symmetry breaking* problems encountered in distributed computing [20]. Solving such problems requires a pre-existing initial asymmetry from which the problem can be solved. In a lot of cases, this initial asymmetry is given by the assumption that no two processes have the same identity.[1]

Distributed distance-1 coloring in the classical point-to-point synchronous message-passing model Let us consider a distributed computing setting, where the processes constitute the vertices of a graph, and the communication channels its edges. The distributed distance-1 vertex coloring problem consists in associating a color with each process such that (a) no two neighbors have the same color, and (b) the total number of colors is as small as possible. This process-coloring problem has essentially been investigated in reliable synchronous networks where any two neighboring processes are connected by a bi-directional channel on which each of them can send and receive messages (e.g. [26, 30, 31, 32]). The main results are described in the monograph [5].

In these reliable point-to-point synchronous systems, processes proceed in synchronized steps, usually called rounds. Each round consists of three phases: during the first phase, each process sends messages to its neighbors; during the second phase, each process receives messages; and during the last phase, each process executes local computation. The fundamental synchrony property is that a message is received in the very same round in which it is sent. Hence, when solving a problem in this synchronous computation model, a crucial attribute of a problem is the minimal number or rounds needed to solve it. As far as the distance-1 coloring problem is concerned, it has been shown that, if the communication graph can be logically oriented such that each process has only one predecessor (e.g., a tree or a ring), $O(\log^* n)$ rounds are necessary and sufficient to color the processes with three colors [12, 23] (n being the total number of processes[2]). Other d1-coloring algorithms are described in several articles (e.g. [4, 6, 18, 22]). They differ in the number of rounds they need and in the number of colors they use to implement d1-coloring. Both the algorithms in [4, 6] color the vertices with $(\Delta+1)$ colors; the first one requires $O(\Delta+\log^* n)$ rounds, while the second one uses $O(\log \Delta)$ rounds. An algorithm described in [18] is for tree graphs or graphs where each vertex has two neighbors; it uses three colors and $O(\log^* n)$ rounds. Another algorithm presented in the same paper addresses constant-degree graphs; it uses $(\Delta + 1)$ colors and $O(\log^* n)$ rounds. The algorithm presented in [22] requires $O(\Delta \log \Delta + \log^* n)$ rounds. These algorithms assume that the processes (vertices) have distinct identities, which are their initial colors. They proceed iteratively, each round reducing the total number of colors.

Distance-2 and distance-3 coloring in shared memory and message-passing models A first class of algorithms addresses the distance-2 vertex coloring problems in systems where communication is through

[1] Let us notice that one of the oldest symmetry-breaking problems is mutual exclusion, where the problem has not to be solved once for all, but repeatedly in a fair way [29]. See also the monograph [3] for impossibility results in distributed computing due to symmetry/indisguishability arguments.

[2] $\log^* n$ is the number of times the function $\log$ needs to be iteratively applied in $\log(\log(\log(...(\log n))))$ to obtain a value ≤ 2. As an example, if n is the number of atoms in the universe, $\log^* n \backsimeq 5$.

shared memory, message-passing, or a mix of the two [8, 9, 17]. These algorithms find their motivation in the application of distance-2 coloring to scientific computing. As a result, they do not fit well the characteristics of wireless networks.

On the other hand, wireless protocols apply or require distance-2 and distance-3 coloring algorithms to prevent packet collisions [11]. To this end, they build a TDMA (Time Division Multiple Access) schedule [28] from the result of the coloring process. TDMA allows processes to share the same frequency channel by dividing the signal into different time slots, one per color. Hence, protocols based on distance-2 coloring guarantee that nodes can transmit messages undisturbed during their time slots. Those based on distance-3 coloring additionally allow receiving nodes to acknowledge unicast messages in the sender's slot.

With this motivation, some authors have proposed self-stabilizing algorithms to solve the distance-2 problem [7, 15] For example, [7] presents a self-stabilizing distance-2 algorithm that uses a constant number of variables on each node and that stabilizes in $O(\Delta^2 m)$ moves and uses at most Δ^2 colors, where m is the number of edges. But these algorithms do not take the broadcast nature of the wireless medium into account, and their operation may thus results in significant packet collisions.

Differently, CADCA [21] is a distributed distance-2 coloring algorithm that takes into account the risk of collisions during the coloring process. To limit this risk, CADCA organizes the nodes to be colored in concentric layers around a sink node. The coloring process proceeds in three phases: the first colors layers 1, 4, 7, ...; the second colors layers 2, 5, 8, ..., and the third colors layers 3, 6, 9. Each such phase uses a specific color palette to avoid conflicts between nodes in different layers and exploits five stages in which the nodes in a layer select colors and resolve the conflicts that arise during the process. However, the algorithm requires nodes to know their position with respect to the sink node, and most importantly it does not entirely eliminate packet collisions, it only reduces their number.

With respect to the distance-3 version of a problem, [19] proposes a self-stabilizing algorithm. Processes compute a maximal independent set and then use it to assign themselves colors. The self-stabilizing part of the algorithm gathers information about each process's 3-hop neighborhood, and does generate collisions. However, the protocol uses a special TDMA slot to continuously run the self-stabilizing protocol without interfering with colored slots. Serena [25] uses a similar approach and also presents a distance-3 coloring algorithm in a broadcast-receive model. However, it does not use a special time-slot to limit the impact of the collisions occurring during the coloring process. The protocols we present in this paper, on the other hand, do not lead to any conflict or collision and do not need any special time slots.

Content of the paper Differently from the previous articles, we propose a collision- and conflict-free algorithm that solves the distance-2 (d2) coloring problem in synchronous networks where (a) processes communicate by broadcasting and receiving messages, and (b) collisions and conflicts are not prevented by the communication model. A collision occurs when a process receives messages from two or more neighbors in the same round. A conflict occurs when, during the same round, two neighbors send a message to each other.

In this broadcast/receive communication model (which covers practical system deployments), there is not a dedicated communication medium for each pair of processes, but a single shared communication medium for each pair composed of a process and all its neighbors. Examples of such communication media are encountered in wireless networks such as sensor networks. In such networks, collision-freedom and conflict-freedom do not come for free, and the algorithms built on top of them must be collision/conflict-free to ensure the consistency of the messages that are exchanged, and consequently the progress of upper-layer applications.

This paper is on collision/conflict-free d2-coloring for the synchronous broadcast/receive communica-

tion model, where the processes are connected by a tree network. Considering such a context, it presents an algorithm which uses $(\Delta+1)$ colors, and is consequently optimal with respect to the number of colors. This algorithm relies on two assumptions to break symmetry: (a) it assumes that a process (not predetermined in advance) receives an external message that defines it as the root of the tree; (b) it assumes that any two processes at distance less than or equal to 2 have distinct identities[3] (hence, depending on the structure of the tree, lots of processes can have the same identity). Its round complexity is $O(d\Delta)$ where d is the depth of the tree. Moreover, no process needs to know n, Δ, or the depth of the tree. Hence a process has no information on the global structure of the tree. Its initial knowledge is purely local: it is restricted to its identity, and the identities of its neighbors.

Roadmap The paper consists of 6 sections. Section 2 presents the synchronous broadcast/receive model, and Section 3 introduces the distance-2 coloring problem. Then, two distributed (message-passing) distance-2 coloring algorithms suited to trees are presented. The presentation is incremental. Section 4 presents first a simple distributed distance-2 coloring algorithm which exploits a sequential tree traversal algorithm as a skeleton, on which are appropriately grafted statements implementing distance-2 the coloring. Then, Section 5 presents a distributed distance-2 coloring algorithm based on a parallel traversal of the tree. This second algorithm extends the basic coloring principles introduced in the first algorithm.

2 Synchronous Broadcast/Receive Model

Processes, initial knowledge, and the communication graph The system model consists of n sequential processes denoted p_1, ..., p_n, connected by a tree communication network.

Each process p_i has an identity id_i, which is known only by itself and its neighbors (processes at distance 1 from it). The constant $neighbors_i$ is a local set, known only by p_i, including the identities of its neighbors (and only them). As noticed in the Introduction, in order for a process p_i not to confuse its neighbors, it is assumed that no two processes at distance less than or equal to 2 have distinct identities. Hence, any two processes at distance greater than 2 may have the same identity. When computing bit complexities, we will assume that any process identity is encoded in $\log_2 n$ bits.

Let Δ_i denote the degree of a process p_i (i.e. $|neighbors_i|$) and let Δ denote the maximal degree of the process graph ($\max\{\Delta_1, \cdots, \Delta_n\}$). While each process p_i knows Δ_i, no process knows Δ (a process p_x such that $\Delta_x = \Delta$ does not know that Δ_x is Δ).

When considering a process p_i, $1 \leq i \leq n$, the integer i is called its index. Indexes are not known by the processes. They are only a notation convenience used as a subscript to distinguish processes and their local variables.

Timing model We assume that processing durations are equal to 0. This is justified by the following observations: (a) the duration of the local computations of a process is negligible with respect to message transfer delays, and (b) the processing duration of a message may be considered as a part of its transfer delay.

Communication is synchronous in the sense that there is an upper bound D on message transfer delays, and this bound is known by all the processes (global knowledge). From an algorithm design point of view, we consider that there is a global clock, denoted $CLOCK$, which is increased by 1, after each period of D physical time units. Each value of $CLOCK$ defines what is usually called a *time slot* or a *round*.

[3] Let us notice that this assumption states nothing more than the fact that a process is able to distinguish its neighbors based on their identities.

Communication operations The processes are provided with two operations denoted broadcast() and receive(). A process p_i invokes broadcast TAG(m) to send the message m, whose type is TAG, to all its neighbors. It is assumed that a process invokes broadcast() only at a beginning of a time slot. When a message TAG(m) arrives at a process p_i, this process is immediately warned of it, which triggers the execution of operation receive() to obtain the message. Hence, a message is always received and processed during the time slot –round– in which it was broadcast.

From a linguistic point of of view, we use the two following **when** notations when writing algorithms, where predicate is a predicate involving $CLOCK$ and possibly local variables of the concerned process.

> **when** TAG(m) **is received do** processing of the message.
> **when** predicate **do** code entailing at most one broadcast() invocation.

Message collision and message conflict Traditional wired round-based synchronous systems assume a dedicated a communication medium for each pair of processes (i.e., this medium is not accessible to the other processes). Hence, in these systems a process p_i obeys the following sequential pattern during each round: (a) first p_i sends a message to all or a subset of its neighbors, (b) then p_i receives the messages sent to it by its neighbors during the current round, and (c) finally executes a local computation which depends on its local state at the beginning of the round and the messages it has received during the current round.

The situation is different in systems such as wireless networks (e.g., sensor networks), which lack a dedicated communication medium per pair of processes. A process p_i shares a single communication medium with all its neighbors, and "message clash" problems can occur, each message corrupting the other ones, and being corrupted by them. Consider a process p_i, these problems are the following.

- If two neighbors of p_i invoke the operation broadcast() during the same time slot (round), a message *collision* occurs.
- If p_i and one of its neighbors invoke broadcast() during the same time slot (round), a message *conflict* occurs.

As already indicated, this paper considers this broadcast/receive communication model. This implies that protocols must prevent collisions and conflicts to ensure both message consistency and computation progress.

3 The Distance-2 Tree Coloring Problem

Solving the collision/conflict problem To prevent collisions and conflicts involving a process p_i, only a single process in the set $\{p_i\} \cup neighbors_i$ can obtain the right to communicate during a given round. To this end, we associate each process with time slots (rounds) in which it can broadcast a message, while none of its 2-hop neighbors can broadcast during these time slots. When considering the whole set of processes, this assignment must be optimal in terms of numbers of colors (ideally allowing as many processes as possible to broadcast during the same round).

This problem is a well-known graph coloring problem called *distance-2* coloring. The aim is to design distributed algorithms associating a color with each process (which will define the time-slots during which it will be allowed to broadcast) such that the following properties are satisfied.

Definition

- Validity: The final color of each process belongs to $\{0, ..., \Delta\}$.
- Consistency: No two processes at distance ≤ 2 have the same color.

- Termination: Each process obtains a color and one process knows that this occurred.

Let us observe that, as at least one process has Δ neighbors, $\Delta + 1$ different colors are necessary. The Validity property states that we are looking for *distributed* algorithms which ensure that $\Delta + 1$ is also a tight upper bound. As we will see such algorithms exist for tree networks.

Using the colors to define the time slots The colors obtained by the processes are used as follows, where $color_i$ is the color obtained by process p_i. The time slots (rounds) during which p_i is allowed to broadcast a message to its neighbors correspond to the values of $CLOCK$ such that $\big((CLOCK \bmod (\Delta{+}1)\big) = color_i)$. As we will see, these time slots are different from the time slots used during the (sequential and parallel) distributed distance-2 algorithms which are presented below. It follows that these algorithms must provide each process with the (initially unknown) value of Δ.

4 Sequential Distance-2 Coloring of a Tree

This section presents a distributed distance-2 coloring algorithm in which there are neither message collisions, nor message conflicts. This algorithm is sequential in the sense that its skeleton is a depth-first tree traversal in which the control flow (implemented by appropriate messages) moves sequentially from a process to another one.

4.1 A sequential algorithm

Algorithm 1 assumes that a single process receives a message, START(), which defines it as the root of the tree. (As noticed in the Introduction, this introduces the initial asymmetry needed to solve the symmetry-breaking problem we are interested in.) This external message causes the receiving process, p_r, to simulate the reception of a fictitious message, COLOR$(id_r, id_r, -1, \emptyset)$. This message initiates a depth-first traversal of the tree.

Messages The algorithm uses two types of messages: COLOR() and TERM(). As each message is broadcast by its sender and received by all its neighbors, it carries the identity of its destination process. Hence, when a process receives a message m, it discards m if it is not the destination of m (predicate $dest \neq id_i$ at line 04 and line 19).

These messages implement a depth-first traversal of the tree network [31]. Each carries the identity of its destination $(dest)$, the identity of its sender $(sender)$, and the color of its sender $(sender\_cl)$. A message COLOR() additionally carries the colors of the already colored neighbors of the sender $(d1colors)$.

Local variables Each process p_i manages the following local variables.

- $state_i$ (initialized to 0) is used by p_i to manage the progress of the tree traversal. Each process traverses five different states during the execution of the algorithm. States 1 and 3 are active: a process in state 1 sends a COLOR () message to a child, while a process in state 3 sends a TERM () message to its parent. States 0 and 2 are waiting states. Nodes listen on the broadcast channel but cannot send any message. Finally, state 4 identifies local termination.
- $parent_i$ saves the identity of the process p_j from which p_i received the message COLOR$(id_i, -, -, -)$; p_i receives exactly one such message. This process, p_j, defines the parent of p_i in the tree. The root

p_r of the tree, defined by the reception of the external message START(), is the only process such that $parent_r = id_r$.

- $sender\_cl_i$ records the color of the parent of p_i. p_i receives this information in the parent's COLOR message.
- $d1colors_i$ is a set containing the colors of the neighbors of p_i, that have already obtained their color.
- to_color_i (initialized to $neighbors_i$) is a set containing the identities of the neighbors of p_i not yet colored.
- $color_i$ contains the color of p_i.

Initialization: $state_i \leftarrow 0$; $to\_color_i \leftarrow neighbors_i$.

(01) **when** START() **is received do** % a single process p_i receives this external message %
(02) p_i executes the lines 04-09 as if it received the message COLOR$(id_i, id_i, -1, \emptyset)$.

(03) **when** COLOR$(dest, sender, sender_cl, d1colors)$ **is received do**
(04) **if** $(dest \neq id_i)$ **then** discard the message (do not execute lines 05-09) **end if**;
(05) $parent_i \leftarrow sender$; $sender\_cl_i \leftarrow sender\_cl$; $d1colors_i \leftarrow \{sender\_cl\}$;
(06) $palette \leftarrow$ sequence $0, 1, 2, ...$ without the colors in $\{sender\_cl_i\} \cup d1colors$;
(07) $color_i \leftarrow$ first color in $palette$;
(08) $to\_color_i \leftarrow neighbors_i \setminus \{parent_i\}$;
(09) **if** $(to\_color_i \neq \emptyset)$ **then** $state_i \leftarrow 1$ **else** $state_i \leftarrow 3$ **end if**.

(10) **when** $((CLOCK$ increases$) \wedge (state_i \in \{1, 3\}))$ **do**
(11) **if** $(state_i = 1)$
(12) **then** $next \leftarrow$ any $id_k \in to\_color_i$;
(13) broadcast COLOR$(next, id_i, color_i, d1colors_i)$; $state_i \leftarrow 2$
(14) **else if** $(parent_i = id_i)$ **then** the root p_i claims termination
(15) **else** broadcast TERM$(parent_i, id_i, color_i)$
(16) **end if**; $state_i \leftarrow 4$
(17) **end if**.

(18) **when** TERM$(dest, id, sender_cl)$ **is received do**
% the tree rooted at process id is properly colored %
(19) **if** $(dest \neq id_i)$ **then** discard the message (do not execute lines 20-21) **end if**;
(20) $to\_color_i \leftarrow to\_color_i \setminus \{id\}$; $d1colors_i \leftarrow d1colors_i \cup \{sender\_cl\}$;
(21) **if** $(to\_color_i \neq \emptyset)$ **then** $state_i \leftarrow 1$ **else** $state_i \leftarrow 3$ **end if**.

Algorithm 1: Distributed depth-first-based distance-2 coloring of a tree (code for p_i)

Description of the algorithm Nodes start the algorithm in state 0, waiting for a COLOR$(id_i, -, -, -)$ message. A process, p_i, receives such a message exactly once. When it receives it, it is visited for the first time by the depth-first tree traversal. It consequently assigns the values of the message parameters to its local variables $parent_i$, $sender\_cl_i$ and $d1colors_i$ (line 05). Then, it computes its color, which is different from the color of the sender of the message and from the colors of the sender's already colored neighbors (lines 06-07). Finally, p_i updates to_color_i, and transitions to a new state: 1 if it has any children, or 3 if it is a leaf node. This prepares the progress of the tree traversal, which will take place at the next time slot (round) (lines 08-09).

When p_i enters the new time slot with $state_i \in \{1, 3\}$ (line 10), it operates as follows to ensure the progress of the tree traversal.

- If $state_i = 1$, it means that p_i has neighbors that have not yet been colored. In this case, p_i selects one of them, and makes the tree traversal progress by broadcasting a message, COLOR$(next, id_i, color_i, d1colors_i)$, which will be processed only by $next$ (line 13). Then, p_i moves into $state_i = 2$ and waits until it receive a TERM message, TERM$(id_i, next, -)$.
- If $state_i = 3$, it means that all the neighbors of p_i have been colored. In this case, if p_i is the root, the distance-2 coloring has terminated (line 14). Otherwise, if p_i is not the root, it broadcasts the message TERM$(parent_i, id_i, color_i)$ to inform its parent that the sub-tree of which it is the root has been colored (line 15). In both cases, p_i transitions to $state_i = 4$, thereby indicating that the algorithm is terminated as far as p_i is concerned (local termination).

Finally, when p_i receives the message TERM$(id_i, id, sender_cl)$, it first updates its local variables to_color_i and $d1colors_i$ according to the received values (line 20). Then, it updates $state_i$ according to the value of to_color_i (indicating whether it has colored all its neighbors, line 21). In the first case, it moves to state 1 and continues traversing another sub-tree. Otherwise it moves to state 3, which will then evolve into state 4 (signaling local termination) as indicated above.

How a process learns the value of Δ A process maintains a local variable max_d_i initialized to Δ_i, and each message TERM() now carries this value. When a process p_i receives a message TERM$(max_d, -, -, -)$ it executes the update statement $max\_d_i \leftarrow \max(max\_d_i, max\_d)$. Finally, when the root process p_r claims termination, it launches a second traversal of the tree to inform the other processes. We do not describe such a propagation of the value of Δ here. This will be done in Section 5.3 in the context of a parallel tree traversal.

4.2 Proof and cost of the algorithm

Lemma 1. *Algorithm* 1 *is both collision-free and conflict-free.*

Proof Let us first observe that, due to the assignment of the variable $state_i$ at line 09 (in the processing of a message COLOR()), or line 21 (in the processing of a message TERM()), a process p_i is allowed to broadcast one and only one message when it executes line 13 or 15. It follows from the associated assignment of the control value 2 or 4 to $state_i$ (line 13 or 15), that p_i cannot broadcast other messages COLOR() or TERM() before executing line 21 (i.e., before it receives a message TERM()).

Let us now observe that, due to line 04, a message COLOR() broadcast by a process at line 13 is processed by a single destination process. Hence, the control flow generated by these messages remains sequential, moving sequentially from a parent process to a child process. Similarly, the control flow generated by the messages TERM() (broadcast at line 15) moves sequentially from a child process p_i to its parent process (whose identity is saved in $parent_i$).

The collision-freedom and conflict-freedom properties of the algorithm follow directly from the sequentiality of the control flow realized by the messages COLOR() and TERM(). $\square_{Lemma\ 1}$

Lemma 2. *Algorithm* 1 *satisfies the Validity, Consistency, and Termination properties.*

Proof Let us first prove the following claim.
Claim. For any p_i and at any time, $|d1colors_i| < \Delta$.
Proof of the claim. Let us consider the local variable $d1colors_i$ of a process p_i. This variable is initialized at line 05, when p_i receives a message COLOR() for the first time. It then contains the color of its parent in

the tree. When the algorithm progresses, the color of a child p_j of p_i is added to $d1colors_i$ (line 20) when p_i receives from p_j a message TERM() carrying p_j's color. It follows that, when p_i issues its last broadcast of COLOR$(child, -, -, d1colors_i)$, $child$ is the identity of its only child without a color, and $d1colors_i$ contains the colors of all the other neighbors of p_i. hence, $|d1colors_i| < \Delta$. End of the proof of the claim.

The Validity property follows from the following observation. When a process p_i selects its color, it follows from the previous claim and lines 06-07 that $|\{parent\_cl\} \cup d1colors| \leq \Delta$. Consequently, there is at least one free color in the set $\{0, \cdots, \Delta\}$.

To prove the Proper-Coloring property let us first observe that, due to (a) the initialization of $d1colors_j$ done when a process p_j receives a message COLOR() from its parent (line 05), and (b) the updates that follow when it receives messages TERM() from its children (line 20), it follows that $d1colors_j$ contains the colors of the already colored neighbors of p_j. Hence, when a process p_i selects a color (lines 06-07), the colors of the already colored processes at distance 2 from it are in the set $d1colors$ carried by the message COLOR() entailing p_i's coloring. Due to line 06, p_i does not select any of these colors.

The Termination property follows from the termination of the sequential traversal, at the end of which the root learns the algorithm has terminated. $\square_{Lemma\ 2}$

The following theorem is an immediate consequence of the previous lemmas.

Theorem 1. *Algorithm* 1 *is a collision-free and conflict-free distance-2 coloring algorithm for trees.*

Cost of the algorithm (Let us recall that a process identity can be encoded with $O(\log n)$ bits.) There are two message types. A message TERM() carries two process identities and a color. A message COLOR() carries two process identities, a color, and set of at most $(\Delta - 1)$ colors. It follows that a message carries at most $2 \log n + \Delta \log \Delta$ bits.

A tree of maximal degree Δ has at least $x \geq \Delta - 1$ leaves. Let us first count the number of broadcasts of a message COLOR(). The root issues at most Δ broadcasts; a process, which is neither the root nor a leaf, issues at most $(\Delta - 1)$ broadcasts; and a leaf issues no broadcast of such a message. It follows that there are $\Delta + (n - 1 - x)(\Delta - 1)$ broadcasts of a message COLOR(). As $x \geq \Delta - 1$, the number of broadcasts of a message COLOR() is upper bounded by $\Delta + (n - \Delta)(\Delta - 1)$.

Each process which is not the root of the tree issues exactly one broadcast of a message TERM(). It follows that there are $(n - 1)$ broadcasts of such a message.

5 Parallel Distance-2 Coloring of a Tree

This section presents a distributed distance-2 coloring for trees, which based on a parallel traversal of the tree network, with feedback (i.e., the dynamically defined root process that launches the network traversal learns of the end of the traversal). This algorithm can be seen as improvement of the previous algorithm in terms of time efficiency.

5.1 A parallel algorithm

Underlying principle Let us first observe that any two children of a process must be prevented from broadcasting simultaneously a message to their neighbors. This is because, being issued by processes at distance at most two, such simultaneous broadcasts will create a collision at least at the parent process.

The idea to prevent this vicinity/concurrency problem is first to direct a parent process to compute the colors of its children, and then, as soon as a process has obtained a color, to allow it to broadcast a message

(COLOR() or TERM()) only during the time slots (rounds) associated with its color. To this end, each process uses the values provided by the global clock ($CLOCK$).

Messages and local variables The message types implementing the parallel tree traversal are the same as in Algorithm 1. Similarly, the local variables $parent_i$, to_color_i, $color_i$, $sender\_cl_i$, and the constant $neighbors_i$, have the same meaning as in Algorithm 1.

Each process p_i manages an additional variable $nb\_cl\_parent_i$, initially 0, which will contain the number of colors needed to color its parent p_j and its neighbors (processes of $neighbors_j$). This value is known to p_i when it receives its first message COLOR() (which defines its sender p_j as p_i's parent). Each process has also a constant $nb\_cl_i = \Delta_i + 1$ which represents the number of colors needed to color itself and its neighbors.

Initialization: $nb\_cl_i = \Delta_i + 1$; $state_i \leftarrow 0$; $max\_nb\_cl_i \leftarrow nb\_cl_i$.

(01) **when** START() **is received do** % a single process p_i receives this external message %
(02) p_i executes lines 04-08 as if it received the message COLOR($\{\langle id_i, cl\rangle\}, id_i, -1, nb\_cl_i$) where $cl = (CLOCK + 1) \bmod nb\_cl_i$.

(03) **when** COLOR($pairs, sender, sender\_cl, nb\_cl\_parent$) **is received do**
(04) **if** (COLOR() already received) **then** discard the message, do not execute lines 05-08) **end if**;
(05) $parent_i \leftarrow sender$; $to\_color_i \leftarrow neighbors_i \setminus \{sender\}$; $sender\_cl_i \leftarrow sender\_cl$;
(06) $color_i \leftarrow cl$ such that $\langle id_i, cl\rangle \in pairs$;
(07) $nb\_cl\_parent_i \leftarrow nb\_cl\_parent$;
(08) **if** $to\_color_i \neq \emptyset$ **then** $state_i \leftarrow 1$ **else** $state_i \leftarrow 3$ **end if**.

(09) **when** $\big((CLOCK \bmod nb\_cl\_parent_i) = color_i) \wedge (state_i \in \{1, 3\})\big)$ **do**
(10) **if** $(state_i = 1)$
(11) **then** $pairs\_for\_children_i \leftarrow$ empty set of pairs;
(12) $palette \leftarrow$ sequence $0, 1, ...$ without the colors $color_i$ and $sender\_cl_i$;
(13) **for each** $k \in to\_color_i$ **do**
(14) $cl \leftarrow$ first color in $palette$;
(15) suppress cl from $palette$ and add $\langle k, cl\rangle$ to $pairs\_for\_children_i$
(16) **end for**;
(17) broadcast COLOR($pairs\_for\_children_i, id_i, color_i, nb\_cl_i$); $state_i \leftarrow 2$
(18) **else** broadcast TERM($parent_i, id_i$); $state_i \leftarrow 4$ % p_i and its neighbors are colored %
(19) **end if**.

(20) **when** TERM($dest, id$) **is received do**
(21) **if** $(dest \neq id_i)$ **then** discard the message (do not execute lines 22-25) **end if**;
(22) $to\_color_i \leftarrow to\_color_i \setminus \{id\}$;
(23) **if** $(to\_color_i = \emptyset)$
(24) **then if** $(parent_i = id_i)$ **then** the root p_i claims termination **else** $state_i \leftarrow 3$ **end if**
(25) **end if**.

Algorithm 2: Parallel distributed distance-2 coloring of a tree (code for p_i)

Description of the algorithm To simplify the presentation, we assume that the initial value of $CLOCK$ is -1. As in the sequential version, a single process p_r receives the external message START(), that defines it as the root of the tree. Moreover, this reception entails the fictitious sending of the message COLOR($\{\langle id_r, cl\rangle\}, id_r, -1, nb\_cl_r$) where $cl = (CLOCK + 1) \bmod nb\_cl_r$ (line 02). Similarly to Al-

gorithm 1, according to the values carried by the message COLOR(), p_r initializes its local variables and obtains the color $cl = 1$ (lines 05-07). It finally updates $state_r$.

Then, when $CLOCK = 1$, p_r executes lines 10-19. More generally, these lines are executed by any process p_i as soon as, after it received its first message COLOR(), the color it obtained ($color_i$) is such that $color_i = CLOCK$ mod $nb\_cl\_parent_i$ (line 09). Hence, as the algorithm (a) allows a process to broadcast only at a round corresponding to its color, (b) allows only colored processes to broadcast COLOR() or TERM() messages, and (c) ensures the distance-2 coloring property (see the proof) without any message collision or message conflict during its execution.

Let us consider a process p_i such that the predicate of line 09 is satisfied. There are two cases.

- If $state_i = 1$, the neighbors of p_i are not colored. Process p_i computes then a color for each of its children (lines 11-16), and broadcasts a message COLOR() to inform them of it (line 17). It is easy to see that the messages COLOR() implements a parallel traversal of the tree from the root to the leaves. Then, p_i enters a waiting period by setting $state_i$ to 2.
- If $state_i = 3$, all neighbors of p_i (and all processes in their sub-trees) are colored. In this case, p_i forwards this information to its parent. As its local participation to the algorithm is terminated, p_i sets $state_i$ to 4.

Finally, when a process p_i receives a message TERM() from one of its children, it first updates to_color_i accordingly (line 22). If this set is empty, it claims termination if it is the root. Otherwise, it assigns 3 to $state_i$, so that it will inform its parent of its local termination at the first time slot (round) at which its color satisfies the predicate of line 09.

5.2 Proof and cost of the algorithm

Lemma 3. *Algorithm* 2 *is collision-free and conflict free.*

Proof Let us observe that a process p_i is allowed to broadcast a message COLOR() (line 16) only if $\big((CLOCK \text{ mod } cl\_bound) = color_i) \wedge (state_i = 1)\big)$. As such a sending entails the assignment $(state_i \leftarrow 2$ (and $state_i$ never decreases), it follows that p_i issues at most one such broadcast. The same occurs for the broadcast of a message TERM().

Due to the initialization of its local variable $state_i$, no process p_i can broadcast a message until one of them receives the message $start()$. Let us assume that (without loss of generality) that $CLOCK = 0$ when a process a process p_i receives this message. It follows from the text of lines 01-08 that, before $CLOCK$ increases, we have $color_i = 1$ and $state_i = 1$ (if the graph is not a singleton), or $color_i = 1$ and $state_i = 3$ (if the graph is a singleton).

Hence, when $CLOCK = 1$, p_i executes lines 09-19 and broadcasts a message COLOR() (line 17), or a message TERM() (line 18), while no other process can broadcast a message. Moreover, due to the color assignment done by p_i at lines 12-16, its neighbors obtain different colors when they receive the message COLOR() from p_i at time $CLOCK = 1$.

Due to the color computation done by p_i at line 12-16 (when $CLOCK = 1$), no two of its neighbors have the same color, and none of them has the same color as p_i. It follows from the time predicate of line 09 that no two processes of the set $neighbors_i \cup id_i$ can broadcast during the same time slot.

Let p_j be a neighbor of the root process p_i. It follows from line12 that, when p_j selects colors for its neighbors, it assigns them colors which are different among themselves and different from its own color and the color of p_i. Hence, due the time predicate of line 09, it follows that no two processes of the set

$neighbors_j \cup \{id_j\}$ can broadcast during the same time slot. The same reasoning applies to the neighbors of p_j, etc., which completes the proof of the lemma. $\square_{Lemma\ 3}$

Lemma 4. *Let p_i and p_j be two processes such that such $parent_i = id_j$. The color of p_i belong to the set $\{0, ..., \Delta_j\}$.*

Proof Let p_i be any children pf p_j. The color of p_i is defined by p_j when it executes the lines 12-16. The local palette of p_j is then the sequence $0, 1, ...$, from which its own color ($color_j$) and the color of its neighbor which is parent ($sender\_cl$) are suppressed (line 12). Hence, at most two integers (one in case of the root) are suppressed from the first $\Delta_j + 1$ non-negative integers from which the palette is built. Hence, p_j can color its neighbors with up to $\Delta_j - 1$ (Δ_j if p_j is the root) colors with values less than $\Delta_j + 1$ and different from that of its parent. $\square_{Lemma\ 4}$

Lemma 5. *Algorithm 2 uses at most $\Delta+1$ different colors to the processes, and no two processes at distance ≤ 2 have the same color.*

Proof Let us consider a process p_i. It has at most Δ neighbors. It follows from lines 12-16, that all its neighbors (including the process from which it received the message COLOR()) are assigned different colors, and those are different from its color $color_i$. Hence no two processes at distance ≤ 2 have the same color. As $\Delta = \max(\Delta_1, \cdots, \Delta_n)$, it follows from Lemma 4 that at most $(\Delta + 1)$ colors are used by the algorithm. $\square_{Lemma\ 5}$

Lemma 6. *Any process is colored, and (only after all processes are colored) this is known by the root process.*

Proof Once a process received an external message START(), it becomes the root of the tree, it takes a color, and broadcasts a message COLOR() to its neighbors at line 17 as soon as the predicate of line 09 becomes satisfied. Then, each of these neighbor processes broadcasts a message COLOR() to their neighbors, etc.

As soon as its neighbors are colored, a process p_i is such that $state_i = 3$, and consequently it broadcasts a message TERM() at line 18. Let p_j be the parent process that broadcasts the message COLOR() received by p_i. When p_j has received message TERM() from all its children, it will proceed to $state_j = 3$ (line 24) and will broadcast TERM() at line 18 as soon as the time predicate of line 09 becomes satisfied.

It follows from the previous observations that the process that received the message START() eventually claims termination at line 24. $\square_{Lemma\ 6}$

The following theorem is an immediate consequence of the previous lemmas.

Theorem 2. *Algorithm 2 is a collision-free and conflict-free distance-2 coloring parallel algorithm for trees.*

Cost of the algorithm Each process which is not a leaf issues one broadcast of a message COLOR(), and each process which is not the root issues one broadcast of a message TERM(). Let x be the number of leaves. There are consequently $(2n - (x + 1))$ broadcasts. As $x \geq \Delta - 1$, the number of broadcasts is upper bounded by $2n - \Delta$.

As far as time complexity is concerned, we have the following. Let d be the depth of the tree, and assume $CLOCK = 0$ when a process p_r receives the message START(), which defines it as the root of the tree. It follows from their color assignment (lines 12-16) that p_r's children start one after the other at times 1, 2, ..., Δ_r. Let p_x be the child of p_r that obtains the color Δ_r. It broadcasts a message COLOR() to its

own children at time $\Delta_r + 1$, and its child with the highest color will do the same at time $\Delta_r + \Delta_i$. Etc. As this worst pattern can repeat along a path of the tree, it follows than process does not receive a message COLOR() before time $d\Delta$.

An analogous reasoning applies to the "return" messages TERM() starting from the leaves to the root. Hence, the time complexity is $O(d\Delta)$. (Let us remind that $d = O(\log_\Delta n)$ when the tree is well-balanced.

5.3 Informing processes of termination

Global vs local termination Algorithm 2 implements a parallel distance-2 coloring, but only the root learns that the algorithm has terminated (global termination). A non-root process p_i knows only that the sub-tree of which it is the root has terminated (local termination). This section, enriches this algorithm so that any process learns about global termination.

The extended algorithm This extended algorithm is made up of Algorithm 2 where line 18 is modified, and the statement "**when** TERM(id) **is received do** ..." (lines 20-24) is replaced by the statements described in Algorithm 3. Moreover, each process p_i manages an additional local variable denoted $max\_nb\_cl_i$, which is initialized to $\Delta_i + 1$.

When considering base Algorithm 2 and its extension Algorithm 3, the lines with the same number are the same in both algorithms, the lines suffixed by a "prime" are modified lines, and the lines N1-N7 are new lines.

```
(20')  when TERM(dest, id, max_nb_cl) is received do
(21)     if (dest ≠ id_i) then discard the message (do not execute lines 22-25) end if;
(22)     to_color_i ← to_color_i \ {id};
(N1)     max_nb_cl_i ← max(max_nb_cl_i, max_nb_cl);
(23)     if (to_color_i = ∅)
(24')       then if (parent_i = id_i) then state_i ← 5 else state_i ← 3 end if
(25)     end if.

(N2)   when ((CLOCK mod max_nb_cl_i) = color_i) ∧ (state_i = 5)) do
(N3)     if (|neighbors_i| ≠ 1) then broadcast END(id_i, max_nb_cl_i) end if; state_i ← 6.

(N4)   when END(parent, max_cl) is received do
(N5)     if (parent = parent_i) ∧ (state_i = 4)
(N6)       then max_nb_cl_i ← max(max_nb_cl_i, max_cl); state_i ← 5
(N7)     end if.
```

Algorithm 3: Parallel distance-2 coloring of a tree: propagation of the termination

The message TERM(id_i) broadcast by a process p_i at line 18 must now carry the current value of $max\_nb\_cl_i$, i.e., p_i broadcasts TERM($parent_i, id_i, max\_nb\_cl_i$). The local variable $max\_nb\_cl_i$ of a process p_i is updated at line N1. In this way, starting from the leaves, these local variables allow the root to know the value $(\Delta + 1)$ (upper bound on the number of colors needed by a process to color itself and its neighbors).

When the root learns about global termination, it proceeds to $state_i = 5$ (line 24'). At this point, the value of its local variable $max\_nb\_cl_i$ is $\Delta + 1$. Moreover, its new local state $state_i$ allows it to inform its children about global termination by broadcasting the message END() carrying the value $\Delta + 1$ (lines N2-N3).

Finally, when a process p_i, which is not the root (hence $state_i = 4$) receives an END $(parent, max\_cl)$ message, from its parent, it updates $max\_nb\_cl_i$ and proceeds to $state_i = 5$ (lines N4-N7), which allows it to forward the END message to its children, if any (lines N2-N3). Let us notice that, as the children of a process p_x do not broadcast END() messages during the same time slot (round), there is no collision of these messages at their parent. But since the parent simply discards these messages, a trivial optimization may consist in relaxing the collision-freedom constraint. Specifically, the children could send their END() messages in parallel as their parent does not need to receive them.[4] Independently of the optimization, it follows from the propagation of the END() messages that, eventually, all the processes are such that $state_i = 6$. When this occurs, they learn about global termination.

Remark Let us consider the situation where, while the coloring algorithm has terminated (i.e., each process has obtained a color and knows Δ), a new process p_x wants to enter the tree and obtains a color. Let p_i be the process chosen to be p_x's parent. If $\Delta_i = \Delta$, p_i cannot accept p_x as a child (this would require an additional color). Differently, if $\Delta_i < \Delta$, it is easy to dynamically add p_x as a child of p_i. To this end, p_i takes the first color cl in 0, 1, ..., Δ, which is different from its own color and the colors of its neighbors. Then, during its next time slot, p_i broadcasts the message NEW(cl, Δ), which, when received by p_x, gives it a proper color. Let us notice that this "join" does not require p_x to have an identity. It is consequently possible for p_i to also assign to p_x an identity not belonging to $\{id_i\} \cup neighbors_i$.

5.4 Merging two trees with the same maximal degree

Considering two trees T_1 and T_2, which have the same maximal degree Δ, let p_x be a process of T_1 and p_y a process of T_2, both having a degree less than Δ.

It is easy to see that, if $\big(id_x \notin (\{id_y\} \cup neighbors_y)\big) \wedge \big(id_y \notin (\{id_x\} \cup neighbors_x)\big)$, the trees T_1 and T_2 can be "added" to compose a single tree made up of T_1 and T_2 connected by the additional edge (p_x, p_y), This composition preserves the colors previously assigned by two independent executions of the algorithm, the one which assigned colors to T_1 and the one which assigned colors to T_2.

6 Conclusion

Synchronous networks where the time is decomposed in a sequence of time slots (rounds) and the communication operations are "broadcast a message to neighbors" and "receive a message a neighbor", are prone to message collisions (which occur when two neighbors of a process send it a message during the same time slot), and message conflicts ((which occur when a process and one of its neighbors broadcast during the same time slot). Distance-2 coloring solves this problem by assigning a color to each process such that there is a matching of time slots with colors which prevents message collisions/conflicts from occurring.

This paper has presented a distributed algorithm which solves the distance-2 coloring problem in tree networks. This algorithm is based on a parallel tree traversal algorithm skeleton on which are grafted appropriate coloring assignments. It is itself collision/conflict-free. It uses only $\Delta + 1$ colors (Δ being the maximal degree of the network), which is optimal. Its time complexity is $O(d\Delta)$ (where d is the depth of the tree). This algorithm does not require a process to initially know more than its identity and the ones of its neighbors. Moreover, any two processes at distance greater than 2 are not prevented from having the

[4] A similar optimization could be applied in the basic algorithm: processes could send COLOR messages in parallel and use the designated time slots only when propagating the TERM messages upstream.

same identity (which is important for scalability issues). Let us also notice that this algorithm is relatively simple (a first-class property).

A very challenging issue is now the design of a parallel collision/conflict-free distance-2 coloring algorithm for synchronous broadcast/receive systems whose communication graph is more general than a tree. As, when considering sequential computing, distance-2 coloring is an NP-complete problem, the design of such a distributed algorithm using a "reasonable" number of colors does not seem to be a "trivial" challenge. (A sequential algorithm suited to an arbitrary graph is presented in Appendix A.)

Acknowledgments

This work has been partially supported by the Franco-Hong Kong ANR-RGC Joint Research Programme 12-IS02-004-02 CO^{2}2Dim, the Franco-German DFG-ANR Project 40300781 DISCMAT (devoted to connections between mathematics and distributed computing), the French ANR project SocioPlug (ANR-13-INFR-0003), and the Labex CominLabs excellence laboratory (ANR-10-LABX-07-01) through the DeSceNt project.

References

[1] Angluin D., Local and global properties in networks of processors. *Proc. 12th ACM Symposium on Theory of Computation (STOC'81)*, ACM Press, pp. 82–93, 1981.

[2] Attiya H., Bar-Noy A., Dolev D., Peleg D., and Reischuk R., Renaming in an asynchronous environment. *Journal of the ACM*, 37(3):524-548, 1990.

[3] Attiya H. and Ellen F., *Impossibility results in distributed computing.* Morgan & Claypool Publishers, 140 pages, 2014 (ISBN 978-1-60845-525-6).

[4] Barenboim L. and Elkin M., Deterministic distributed vertex coloring in polylogarithmic time. *Journal of the ACM*, 58(5):23, 2011.

[5] Barenboim L. and Elkin M., *Distributed graph coloring, fundamental and recent developments*, Morgan & Claypool Publishers, 155 pages, 2014.

[6] Barenboim L., Elkin M., and Kuhn F., Distributed (Delta+1)-coloring in linear (in Delta) time. *SIAM Journal of Computing*, 43(1):72-95, 2014.

[7] Blair J. and Manne F., An efficient self-stabilizing distance-2 coloring algorithm. *Proc. 16th Colloquium on Structural Information and Communication Complexity (SIROCCO'10)*, Springer LNCS 5869, pp. 237-251, 2009.

[8] Bozdag D., Çatalyürek U.V., Gebremedhin A.H., Manne F., Boman E.G., and Öuzgüner F., A Parallel distance-2 graph coloring algorithm for distributed memory computers. *Proc. Int'l Conference on High Performance Computing and Communications (HPCC'05)*, Springer LNCS 3726, pp. 796-806, 2005.

[9] Bozdag D., Gebremedhin A.S., Manne F., Boman G. and Çatalyürek U.V., A framework for scalable greedy coloring on distributed-memory parallel computers. *Journal of Parallel and Distributed Computing*, 68(4):515-535, 2008.

[10] Castañeda A., Rajsbaum S., and Raynal M., The renaming problem in shared memory systems: an introduction. *Elsevier Computer Science Review*, 5:229-251, 2011.

[11] Chipara O., Lu C., Stankovic J., and Roman. G.-C., Dynamic conflict-free transmission scheduling for sensor network queries. *IEEE Transactions on Mobile Computing*, 10(5):734-748, 2011.

[12] Cole R. and Vishkin U., Deterministic coin tossing with applications to optimal parallel list ranking. *Information and Control*, 70(1):32-53, 1986.

[13] Dolev D., Klawe M., and Rodeh M., An $O(n \log n)$ unidirectional distributed algorirthm for extrema finding in a circle. *Journal of Algorithms*, 3:245–260, 1982.

[14] Ergen S.C. and Varaiya P., PEDAMACS: Power efficient and delay aware medium access protocol for sensor networks. *IEEE Transaction on Mobile Computing*, 5(7):920-930, 2005.

[15] Gairing M., Goddard W, Hedetniemi S. T., Kristiansen P., and McRae A. A., Distance-two information in self-stabilizing algorithms. *Parallel Processing Letters*, 14(03-04):387–398, 2004.

[16] Garey M.R. and Johnson D.S., *Computers and intractability: a guide to the theory of NP-completeness*. Freeman W.H. & Co, New York, 340 pages, 1979.

[17] Gebremedhin A.H., Manne F., and Pothen A., Parallel distance-k coloring algorithms for numerical optimization. *Proc. Euro-Par Parallel Processing*, Springer LNCS 2400, pp. 912-921, 2002.

[18] Goldberg A., Plotkin S., and Shannon G., Parallel symmetry-breaking in sparse graphs. *SIAM Journal on Discrete Mathematics*, 1(4):434-446, 1988.

[19] Herman T., Tixeuil S., A distributed TDMA slot assignment algorithm for wireless sensor networks. *Proc. Int'l Workshop on Algorithmic Aspects of Wireless Sensor Networks (ALGOSENSORS'04)*, Springer LNCS 3121, pp. 45-58, 2004.

[20] Imbs D., Rajsbaum S., and Raynal M., The universe of symmetry breaking tasks. *Proc. 18th Colloquium on Structural Information and Communication Complexity (SIROCCO'11)*, Springer LNCS 6796, pp. 66-77, 2011.

[21] Jemili I., Ghrab D., Belghith A., Derbel B., and Dhraief A., Collision aware coloring algorithm for wireless sensor networks. In *Proc. 9th Int' lWireless Communications and Mobile Computing Conference (IWCMC'13)*, pages 1546-1553, 2013.

[22] Kuhn F. and Wattenhofer R., On the complexity of distributed graph coloring. *Proc. 25th ACM Symposium Principles of Distributed Computing (PODC'06)*, ACM Press, pp. 7-15, 2006.

[23] Linial N., Locality in distributed graph algorithms. *SIAM Journal on Computing*, 21(1):193-201, 1992.

[24] Lynch N.A. Upper bounds for static resource allocation in a distributed system. *Journal of Computer and System Sciences*, 23(2):254-278, 1981.

[25] Mahfoudh S., Chalhoub G., Minet P., Misson M., and Amdouni I., Node coloring and color conflict detection in wireless sensor networks. *Future Internet*, 2(4):469, 2010.

[26] Peleg D., *Distributed computing, a locally sensitive approach*. SIAM Monographs on Discrete Mathematics and Applications, 343 pages, 2000 (ISBN 0-89871-464-8).

[27] Prabhakar B., Uysal-Biyikoglu, E., and El Gamal A., Energy-efficient packet transmission over awireless link. *IEEE/ACM Transactions on Networking*, 10(4):487499, 2002.

[28] Ramanathan S., A unified framework and algorithms for (T/F/C)DMA channel assignment in wireless networks. *Proc. 16th IEEE INFOCOM Conference*, IEEE Press, pp. 900-907, 1997.

[29] Raynal M., *Algorithms for mutual exclusion*. The MIT press, 107 pages, 1986 (ISBN 0-262-18119-3).

[30] Raynal M., *Fault-tolerant agreement in synchronous message-passing systems.* Morgan & Claypool Publishers, 165 pages, 2010 (ISBN 978-1-60845-525-6).

[31] Raynal M., *Distributed algorithms for message-passing systems*. Springer, 500 pages, 2013, ISBN 978-3-642-38122-5.

[32] Santoro N., *Design and analysis of distributed algorithms*. Wiley, 589 pages, 2007.

A A sequential distributed algorithm for an arbitrary graph

Algorithm 4 is a sequential distance-2 coloring algorithm for an arbitrary (connected) graph. The design of this algorithm is the same the one of Algorithm 1. It added complexity comes from the fact it allows the sequential control flow (implemented by the messages COLOR() and TERM() in Algorithm 1) to back track when a process discovers a coloring conflict. This backtracking is implemented by the messages CORRECT(), CORRECTED_COLOR(), and RESUME_COLORING().

Considering the network of Figure 1, An example of execution of Algorithm 4 is depicted in Table 1. Due to space restriction we abbreviate the following:

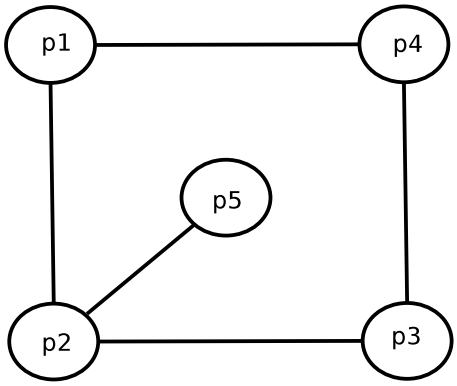

Figure 1: A 5-process arbitrary network

- br = broadcast operation,
- $d1_i$ = $d1colors_i$,
- $d2_i$ = $d2colors_i$,
- CL(p_a,p_b,c,z,S) = COLOR (p_a,p_b,c,z,S),
- CRL(p_a,p_b,c,S) = CORRECT(p_a,p_b,c,z,S),
- CR_CL(p_a,p_b,c,S) = CORRECTED_CL(p_a,p_b,c,S),
- RSM_CL(p_a,p_b) = RESUME_CL(p_a,p_b),
- $s_i = state_i$.

Process p_1 receives the message START() at round -1. It broadcasts the message COLOR (id_1, id_1, -1, 0, $\emptyset$) to itself. It receives this message at round 0. It updates its local variables $d2_1$ at line 02 (abbreviated as L2), $d1_1$, s_1 and $color_1$ at line 06 (abbreviated as L6). This appears in the first row of the table where the value of $CLOCK$ is 0. Then, when $CLOCK$ progresses to 1, p_1 has $state_1 = 2$, therefore, it broadcasts the message COLOR() with appropriate parameters, where it proposes a color that is not in $d2_1 \cup d1_1$ to each of its neighbors, and subsequently enters $state_1 = 0$ (L16). This appears in the second row of the table where the value of $CLOCK$ is 1. When p_2 and p_4 receive this message at round 2, they execute the associated processing at L2 and L6 for p_2 and at L10 for p_4 . So, in this round p_2 updates its local variables $d2_2$ at L2, $d1_2$, s_2 and $color_2$ at L6. p_4 updates its local variables $d2_4$ at L2, $d1_4$, at L10. This appears in the third row of the table where $CLOCK$ is 2. In round 3, p_2 has $state_2 = 2$, so it broadcasts the message COLOR() with appropriate parameters (L16) and gets $state_2 = 0$ (L16), Etc.

In round 6, p_4 finds that the color proposed by its neighbor p_3 (color 0 proposed in round 5) is in $d1_4$, so it "refuses" this color and gets $state_4 = 1$ (L4). In round 7, p_4 gets a color which is not in $d2_4 \cup d1_4$ (L11) and broadcasts the message CORRECT(id_3, id_4, 2, 0, $\{0\}$). In round 8, p_3 updates $d1_3$ and $d2_3$ and gets a new color (L31). Its state will allow it to broadcast the message CORRECTED_CL() (round 11). The broadcast of this message will trigger the broadcast of the message RESUME_CL() (round 15) to resume the coloring as described before.

init: $d1colors_i \leftarrow \emptyset$; $sender_i \leftarrow 0$; $d2colors_i \leftarrow \emptyset$; $state_i \leftarrow 0$; $to\_color_i \leftarrow neighbors_i$.

when START() **is received do**
(01) the reception of this message defines its receiver p_i as the root of the tree;
this process p_i simulates then the sending of COLOR$(id_i, id_i, -1, 0, \emptyset)$ to itself.

when COLOR$(dest, sender, sender_cl, proposed_color, d1colors)$ **is received do**
(02) $to\_color_i \leftarrow to\_color_i \setminus \{sender\}$; $d2colors_i \leftarrow d2colors_i \cup d1colors$;
(03) **if** $(dest = id_i) \wedge ((sender\_cl \in d1colors_i) \vee (proposed\_color \in (d1colors_i \cup d2colors_i)))$
(04) **then** $state_i \leftarrow 1$; $sender_i \leftarrow sender$;
(05) **else if** $(dest = id_i) \wedge (to\_color_i \neq \emptyset)$
(06) **then** $state_i \leftarrow 2$; $color_i \leftarrow proposed\_color$; $parent_i \leftarrow sender$; $d1colors_i \leftarrow d1colors_i \cup \{sender\_cl\}$;
(07) **else if** $(dest = id_i) \wedge (to\_color_i = \emptyset)$ **then** $state_i \leftarrow 3$; $parent_i \leftarrow sender$; **end if**
(08) **end if**
(09) **end if**;
(10) **if** $(dest \neq id_i)$ **then** $d2colors_i \leftarrow d2colors_i \cup \{proposed\_color\}$; $d1colors_i \leftarrow d1colors_i \cup \{sender\_cl\}$ **end if**.

when ($CLOCK$ increases)
(11) **if** $(state_i = 1)$ **then** $color_i \leftarrow$ the first color in $0, 1...$ which is not in $d1colors_i \cup d2colors_i$;
(12) broadcast CORRECT$(sender_i, id_i, color_i, d1colors_i)$; $state_i \leftarrow 0$
(13) **end if**;
(14) **if** $(state_i = 2)$ **then** $color\_for\_child \leftarrow$ the first color in $0, 1..$ which is not in $d1colors_i \cup color_i$;
(15) $next \leftarrow$ any $id_k \in to\_color_i$;
(16) broadcast COLOR$(next, id_i, color_i, color_for_child, d1colors_i)$; $state_i \leftarrow 0$
(17) **end if**;
(18) **if** $(state_i = 3)$ **then** broadcast TERM$(parent_i, color_i, id_i)$; $state_i \leftarrow 0$ **end if**;
(19) **if** $(state_i = 4)$ **then** broadcast CORRECTED_COLOR $(sender_i, parent_i, id_i, color_i)$; $state_i \leftarrow 0$ **end if**;
(20) **if** $(state_i = 5)$ **then** broadcast RESUME_COLORING $(sender_i, id_i)$; $state_i \leftarrow 0$ **end if**;
(21) **if** $(state_i = 6)$ **then** broadcast CORRECTED_COLOR $(-1, -1, id_i, corrected\_cl_i)$; $state_i \leftarrow 0$ **end if**.

when TERM$(dest, id, color)$ **is received do**
(22) **if** $(dest \neq id_i)$ **then** discard the message (do not execute lines 23-28) **end if**;
(23) $to\_color_i \leftarrow to\_color_i \setminus \{id\}$;
(24) **if** $(to\_color_i = \emptyset)$ % the neighbors of p_i are properly colored %
(25) **then if** $(parent_i = id_i)$ **then** the root claims the graph is colored **else** $state_i \leftarrow 3$ **end if**
(26) **else** $d1colors_i \leftarrow d1colors_i \cup \{color\}$; $next \leftarrow$ any $id_k \in to\_color_i$;
(27) $color\_to\_child \leftarrow$ the first color in $0, 1..$ which is not in $d1colors_i \cup color_i$; $state_i \leftarrow 2$
(28) **end if**.

when CORRECT$(dest, sender, color, d1colors)$ **is received do**
(29) **if** $(dest \neq id_i)$ **then** discard the message (do not execute lines 30-31) **end if.**
(30) $d2colors_i \leftarrow d2colors_i \cup d1colors$; $d1colors_i \leftarrow d1colors_i \cup color$;
(31) $color_i \leftarrow$ the first color in $0, 1..$ which is not in $d1colors_i \cup d2colors_i$; $sender_i \leftarrow sender$; $state_i \leftarrow 4$.

when CORRECTED_COLOR$(dest1, dest2, sender, color)$ **is received do**
(32) **if** $(dest1 \neq id_i) \wedge (dest1 \neq -1)$ **then** delete the last color added to $d1colors_i$; $d1colors_i \leftarrow d1colors_i \cup \{color\}$ **end if**;
(33) **if** $(dest1 \neq id_i) \wedge (dest1 = -1)$ **then** delete the last color added to $d2colors_i$; $d2colors_i \leftarrow d2colors_i \cup \{color\}$ **end if**;
(34) **if** $(dest2 = id_i)$ **then** $state_i \leftarrow 6$; $corrected\_cl_i \leftarrow color$ **end if**;
(35) **if** $(dest1 = -1) \wedge (dest2 = -1) \wedge (color_i = color)$ **then** $state_i \leftarrow 5$ **end if**.

when RESUME_COLORING$(dest, sender)$ **is received do**
(36) **if** $(dest \neq id_i)$ **then** discard the message (do not execute lines 37-38) **end if**;
(37) $parent_i \leftarrow sender$;
(38) **if** $(to\_color_i \neq \emptyset)$ **then** $state_i \leftarrow 2$ **else** $state_i \leftarrow 3$ **end if**.

Algorithm 4: Sequential distance-2 coloring for an arbitrary graph (code for p_i)

p_i / clock	p_1	p_2	p_3	p_4	p_5
0	$d2_1$ = {} (L2), $d1_1$ = $\{-1\}$ $s_1 = 2$, $color_1 = 0$ (L6)				
1	br CL(id_2,id_1,0,1,$\{-1\}$) $s_1 = 0$ (L16)				
2		$d2_2 = \{-1\}$ (L2),$d1_2 = \{0\}$ $s_2 = 2$, $color_2 = 1$ (L6)		$d2_4 = \{1\}$,$d1_4 = \{0\}$ (L10)	
3		br CL(id_3,id_2, 1, 2, $\{0\}$) $s_2 = 0$ (L16)			
4	$d2_1 = \{-1, 2\}$, $d1_1 = \{1\}$ (L10)		$d2_=\{0\}$ (L2),$d1_3 = \{1\}$ $s_3 = 2$, $color_3 = 2$ (L6)		$d2_5 = \{0, 2\}$ $d1_5 = \{1\}$ (L10)
5			br CL(id_4,id_3, 2, 0, $\{1\}$) $s_3 = 0$ (L16)		
6		$d2_2$ = $\{-1, 0\}$, $d1_2 = \{0, 2\}$ (L10)		$d2_4 = \{1\}$ (L2), $s_4 = 1$ (L4)	
7				$color_4 = 2$ (L11), br CR(id_3,id_4,2,$\{0\}$) $s_4 = 0$ (L12)	
8			$d2_3$ = $\{0\}$, $d1_3$ = $\{1, 2\}$ (L30) $color_3 = 3, s_3 = 4$ (L31)		
9			br CR_CL(id_4,id_2,id_3,3) $s_3 = 0$ (L19)		
10		$d1_2$ = $\{0, 3\}$ (L32), $s_2 = 6$ (L34)			
11		br CR_CL(-1,-1,p_2,3) $s_2 = 0$ (L21)			
12	$d2_1 = \{-1, 3\}$ (L33)				
13			$s_3 = 5$ (l35)		$d2_5$ = $\{0, 3\}$ (L32)
14			br RSM_CL(id_4, id_3) $s_3 = 0$ (L20)		
15				$s_4 = 3$ (L38)	
16				br TERM(id_3, id_4) $s_4 = 0$ (L18)	
17			$s_3 = 3$ (L38)		
18			br TERM(id_2, id_3) $s_3 = 0$ (L18)		
19		$s_2 = 2$ (L27) $d1_2 = \{0, 3\}$ (L26)			
20		br CL(id_5,id_2,1,2,$\{0, 3\}$) (L16) $s_2 = 0$ (L16)			
21	$d2_1$ = $\{-1, 3, 2\}$, $d1_1 = \{0, 1\}$ (L10)		$d2_3$ =$\{0, 2\}$, $d1_3 = \{2, 1\}$ (L10)		$d2_5$ = $\{0, 3\}$ $s_5 = 3$ (L07)
22					br TERM(id_2,id_5) $s_5 = 0$ (L18)
23		$s_2 = 3$ (L25)			
24		br TERM(id_1, id_2) $s_2 = 0$ (L18)			
25	end algorithm (L25)				

Table 1: An execution of Algorithm 4 on the network of Figure 1