

# Vertex Coloring with Communication and Local Memory Constraints in Synchronous Broadcast Networks

Hicham Lakhlef, Michel Raynal, François Taïani

► **To cite this version:**

Hicham Lakhlef, Michel Raynal, François Taïani. Vertex Coloring with Communication and Local Memory Constraints in Synchronous Broadcast Networks. [Research Report] 2035, IRISA; Université de Rennes 1. 2016, pp.23. <hal-01300095>

**HAL Id: hal-01300095**

**<https://hal.inria.fr/hal-01300095>**

Submitted on 8 Apr 2016

**HAL** is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

# Vertex Coloring with Communication and Local Memory Constraints in Synchronous Broadcast Networks

Hicham Lakhlef<sup>‡</sup> Michel Raynal<sup>‡,\*</sup> François Taïani<sup>‡</sup>

<sup>‡</sup> IRISA, Université de Rennes, France

\* Institut Universitaire de France

hicham.lakhlef@irisa.fr raynal@irisa.fr francois.taiani@irisa.fr

Tech Report #2035, 23 pages, April 2016  
IRISA, University of Rennes 1, France

## Abstract

The vertex coloring problem has received a lot of attention in the context of synchronous round-based systems where, at each round, a process can send a message to all its neighbors, and receive a message from each of them. Hence, this communication model is particularly suited to point-to-point communication channels. Several vertex coloring algorithms suited to these systems have been proposed. They differ mainly in the number of rounds they require and the number of colors they use.

This paper considers a broadcast/receive communication model in which message collisions and message conflicts can occur (a collision occurs when, during the same round, messages are sent to the same process by too many neighbors; a conflict occurs when a process and one of its neighbors broadcast during the same round). This communication model is suited to systems where processes share communication bandwidths. More precisely, the paper considers the case where, during a round, a process may either broadcast a message to its neighbors or receive a message from at most  $m$  of them. This captures communication-related constraints or a local memory constraint stating that, whatever the number of neighbors of a process, its local memory allows it to receive and store at most  $m$  messages during each round. The paper defines first the corresponding generic vertex multi-coloring problem (a vertex can have several colors). It focuses then on tree networks, for which it presents a lower bound on the number of colors  $K$  that are necessary (namely,  $K = \lceil \frac{\Delta}{m} \rceil + 1$ , where  $\Delta$  is the maximal degree of the communication graph), and an associated coloring algorithm, which is optimal with respect to  $K$ .

**Keywords:** Broadcast/receive communication, Bounded local memory, Collision-freedom, Conflict-freedom, Distributed algorithm, Message-passing, Multi-coloring, Network traversal, Scalability, Synchronous system, Tree network, Vertex coloring.

# 1 Introduction

**Distributed message-passing synchronous systems** From a structural point of view, a message-passing system can be represented by a graph, whose vertices are the processes, and whose edges are the communication channels. It is naturally assumed that the graph is connected.

Differently from asynchronous systems, where there is no notion of global time accessible to the processes, synchronous message-passing systems are characterized by upper bounds on message transfer delays and processing times. Algorithms for such systems are usually designed according to the round-based programming paradigm. The processes execute a sequence of synchronous rounds, such that, at every round, each process first sends a message to its neighbors, then receives messages from them, and finally executes a local computation, which depends on its local state and the messages it has received. The fundamental synchrony property of this model is that every message is received in the round in which it was sent. The progress from one round to the next is a built-in mechanism provided by the model. Algorithms suited to reliable synchronous systems can be found in several textbooks (e.g., [19, 21])<sup>1</sup>. When considering reliable synchronous systems, an important issue is the notion of local algorithm. Those are the algorithms whose time complexity (measured by the number of rounds) is smaller than the graph diameter [1, 17].

**Distributed graph coloring in point-to-point synchronous systems** One of the most studied graph problems in the context of an  $n$ -process reliable synchronous system is the vertex coloring problem, namely any process must obtain a color, such that neighbor processes must have different colors (distance-1 coloring), and the total number of colors is reasonably “small”. More generally, the distance- $k$  coloring problem requires that no two processes at distance less or equal to  $k$ , have the same color. When considering sequential computing, the optimal distance-1 coloring problem is NP-complete [12].

When considering the distance-1 coloring problem in an  $n$ -process reliable synchronous system, 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 at most three colors [10, 17]<sup>2</sup>. Other distance-1 coloring algorithms are described in several articles (e.g. [3, 5, 14, 16]). They differ in the number of rounds they need and in the number of colors they use to implement distance-1 coloring. Let  $\Delta$  be the maximal degree of the graph (the degree of a vertex is the number of its neighbors). Both algorithms in [3, 5] 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 is described in [14] for trees, which uses three colors and  $O(\log^* n)$  rounds. Another algorithm presented in the same paper addresses constant-degree graphs, and uses  $(\Delta + 1)$  colors and  $O(\log^* n)$  rounds. The algorithm presented in [16] requires  $O(\Delta \log \Delta + \log^* n)$  rounds. These algorithms assume that the processes have distinct identities<sup>3</sup>, which define their initial colors. They proceed iteratively, each round reducing the total number of colors. Distributed distance-2 and distance-3 coloring algorithms, suited to various synchronous models, are presented in [6, 8, 9, 11, 13, 15].

**Motivation and content of the paper** The previous reliable synchronous system model assumes that there is a dedicated (e.g., wired) bi-directional communication channel between each pair of neighbor processes. By contrast, this paper considers a broadcast/receive communication model in which there is no dedicated communication medium between each pair of neighbor processes. This covers practical system deployments, such as wireless networks and sensor networks. In such networks, the prevention

---

<sup>1</sup>The case where processes may exhibit faulty behaviors (such as crashes or Byzantine failures) is addressed in several books (e.g., [2, 18, 19, 20]).

<sup>2</sup> $\log^* n$  is the number of times the function  $\log$  needs to be iteratively applied in  $\log(\log(\log(\dots(\log n))))$  to obtain a value  $\leq 2$ . As an example, if  $n$  is the number of atoms in the universe,  $\log^* n \simeq 5$ .

<sup>3</sup>Some initial asymmetry is necessary to solve *breaking symmetry problems* with a deterministic algorithm.

of collisions (several neighbors of the same process broadcast during the same round), or conflicts (a process and one of its neighbors issue a broadcast during the same round), does not come for free. In particular, round-based algorithms that seek to provide deterministic communication guarantees in these systems must be collision and conflict-free (C2-free in short).

We are interested in this paper to offer a programming model in which, at any round, a process can either broadcast a message to its neighbors (conflict-freedom), or receive messages from at most  $m$  of its neighbors ( $m$ -collision-freedom). This means that we want to give users a round-based programming abstraction guaranteeing conflict-freedom and a weakened form of collision-freedom, that we encapsulate under the name C2 $m$ -freedom (if  $m = 1$ , we have basic C2-freedom).

The ability to simultaneously receive messages from multiple neighbors can be realized in practice by exploiting multiple frequency channels<sup>4</sup>. The parameter  $m \geq 1$  is motivated by the following observations. While a process (e.g., a sensor) may have many neighbors, it can have constraints on the number of its reception channels, or constraints on its local memory, that, at each round, allow it to receive and store messages from only a bounded subset of its neighbors, namely  $m$  of them ( $m = 1$ , gives the classic C2-free model, while  $m \geq \Delta$  assumes no collision can occur as in the classic broadcast/receive model presented previously). This “bounded memory” system parameter can be seen as a scalability parameter, which allows the degree of a process (number of its neighbors) to be decoupled from its local memory size.

C2 $m$ -freedom can be easily translated as a coloring problem, where any two neighbors must have different colors (conflict-freedom), and any process has at most  $m$  neighbors with the same color ( $m$ -collision-freedom). Once such a coloring is realized, message consistency is ensured by restricting the processes to broadcast messages only at the rounds associated with their color. While it is correct, such a solution can be improved, to allow more communication to occur during each round. More precisely, while guaranteeing C2 $m$ -freedom, it is possible to allow processes to broadcast at additional rounds, by allocating multiple colors to processes. From a graph coloring point of view, this means that, instead of only one color, a set of colors can be associated with each process, while obeying the following two constraints: (a) for any two neighbor processes, the intersection of their color sets must remain empty; and (b) given any process, no color must appear in the color sets of more than  $m$  of its neighbors.

We call *Coloring with Communication/Memory Constraints* (CCMC) the coloring problem described above. More precisely, this problem is denoted CCMC( $n, m, K, \geq 1$ ), where  $n$  is the number of processes (vertices),  $m$  is the bound on each local memory (bound on the number of simultaneous communication from a reception point of view), and  $K$  the maximal number of colors that are allowed. “ $\geq 1$ ” means that there is no constraint on the number of colors that that can be assigned to a process. CCMC( $n, m, K, 1$ ) denotes the problem instance where each process is assigned exactly one color. From a technical point of view, the paper focuses on tree networks. It presents a lower bound on the value of  $K$  for these communication graphs, and an algorithm, optimal with respect to  $K$ , which solves both instances of CCMC.

**Roadmap** The paper is made up of 7 sections. Section 2 presents the underlying system model. Section 3 formally defines the CCMC problem. Then, considering tree networks, whose roots are dynamically defined, Section 4 presents a lower bound on  $K$  for CCMC( $n, m, K, 1$ ) and CCMC( $n, m, K, \geq 1$ ) to be solved. Section 5 presents then a  $K$ -optimal algorithm solving CCMC( $n, m, K, \geq 1$ ). (from which a solution to CCMC( $n, m, K, 1$ ) can be easily obtained.) Section 6 presents a proof of the algorithm. Finally, Section 7 concludes the paper.

---

<sup>4</sup>Depending on the underlying hardware (e.g., multi-frequency bandwidth, duplexer, diplexer), variants of this broadcast/receive communication pattern can be envisaged. The algorithms presented in this paper can be modified to take them into account.

## 2 Synchronous Broadcast/Receive Model

**Processes, initial knowledge, and the communication graph** The system model consists of  $n$  sequential processes denoted  $p_1, \dots, p_n$ , connected by a connected communication graph. 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 to distinguish processes and their local variables.

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). 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 the same identity. Hence, any two processes at distance greater than 2 can have the very same identity.

$\Delta_i$  denotes the degree of process  $p_i$  (i.e.  $|neighbors_i|$ ) and  $\Delta$  denotes the maximal degree of the graph ( $\max\{\Delta_1, \dots, \Delta_n\}$ ). While each process  $p_i$  knows  $\Delta_i$ , no process knows  $\Delta$  (a process  $p_x$  such that  $\Delta_x = \Delta$  does not know that  $\Delta_x$  is  $\Delta$ ).

**Timing model** Processing durations are assumed equal to 0. This is justified by the following observations: (a) the duration of local computations 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*.

**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 its neighbors. It is assumed that a process invokes `broadcast()` only at a beginning of a time slot (round). When a message `TAG(m)` arrives at a process  $p_i$ , this process is immediately warned of it, which triggers the execution of the operation `receive()` to obtain and process 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 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** communication-free processing of the message.
- when** predicate **do** code entailing at most one `broadcast()` invocation.

**Message collision and message conflict in the  $m$ -bounded memory model** As announced in the Introduction, there is no dedicated communication medium for each pair of communicating processes, and each process has local communication and memory constraints such that, at every round, it cannot receive messages from more than  $m$  of its neighbors. If communication is not controlled, “message clash” problems can occur, messages corrupting each other. Consider a process  $p_i$  these problems are the following.

- If more than  $m$  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 indicated in the introduction, an aim of coloring is to prevent message clashes from occurring, i.e., in our case, ensures  $C2m$ -freedom. Let us observe that a coloring algorithm must itself be  $C2m$ -free.

### 3 The Coloring with Communication/Memory Constraints Problem

**Definition of the CCMC problem** Let  $\{p_1, \dots, p_n\}$  be the  $n$  vertices of a connected undirected graph. As already indicated,  $neighbors_i$  denotes the set of the neighbors of  $p_i$ . Let the color domain be the set of non-negative integers, and  $m$  and  $K$  be two positive integers. The aim is to associate a set of colors, denoted  $colors_i$ , with each vertex  $p_i$ , such that the following properties are satisfied.

- Conflict-freedom.  $\forall i, j : (p_i \text{ and } p_j \text{ are neighbors}) \Rightarrow colors_i \cap colors_j = \emptyset$ .
- $m$ -Collision-freedom.  $\forall i, \forall c : |\{j : p_j \in neighbors_i \wedge c \in colors_j\}| \leq m$ .
- Efficiency.  $|\cup_{1 \leq i \leq n} colors_i| \leq K$ .

The first property states the fundamental property of vertex coloring, namely, any two neighbors are assigned distinct colors sets. The second property states the  $m$ -constraint coloring on the neighbors of every process, while the third property states an upper bound on the total number of colors that can be used.

As indicated in the Introduction, this problem is denoted  $CCMC(n, m, K, 1)$  if each color set is constrained to be a singleton, and  $CCMC(n, m, K, \geq 1)$  if there is no such restriction.

**Example** An example of such a multi-coloring of a 21-process network, where  $\Delta = 10$ , and with the constraint  $m = 3$ , is given in Figure 1. Notice that  $K = \lceil \frac{\Delta}{m} \rceil + 1 = 5$  (the color set is  $\{0, 1, 2, 3, 4\}$ ).

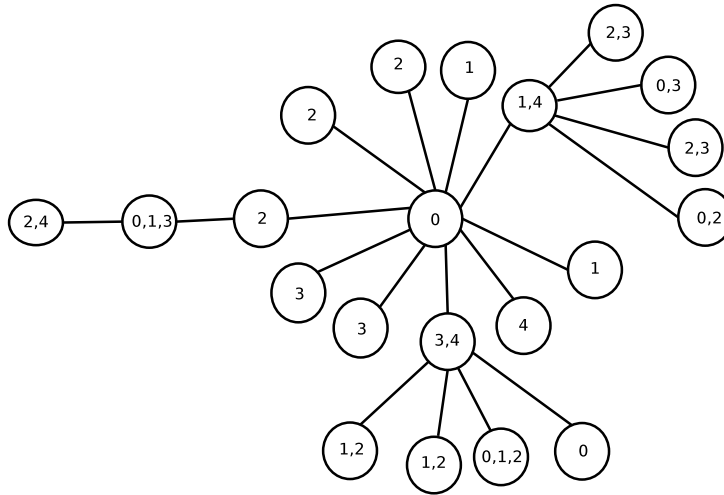


Figure 1: Multi-coloring of a 21-process 10-degree tree with the constraint  $m = 3$  (5 colors)

**Particular instances** The problem instance  $CCMC(n, \infty, K, 1)$  is nothing other than the classical vertex coloring problem, where at most  $K$  different colors are allowed ( $m = \infty$  states that no process imposes a constraint on the colors of its neighbors, except that they must be different from its own color). The problem instance  $CCMC(n, 1, K, 1)$  is nothing other than the classical distance-2 coloring problem (vertices at distance  $\leq 2$  have different colors).

**Using the colors** The reader can easily see that  $CCMC(n, m, K, \geq 1)$  captures the general coloring problem informally stated in the introduction. Once a process  $p_i$  has been assigned a set of colors  $colors_i$ , at the application programming level, it is allowed to broadcast a message to neighbors at the rounds (time slots) corresponding to the values of  $CLOCK$  such that  $(CLOCK \bmod K) \in colors_i$ .



## 4 CCMC( $n, m, K, \geq 1$ ) in a Tree Network: Lower Bounds

### 4.1 An impossibility result

Considering tree networks, this section presents a lower bound on  $K$ : neither  $\text{CCMC}(n, m, K, 1)$ , nor  $\text{CCMC}(n, m, K, \geq 1)$ , can be solved for  $K \leq \lceil \frac{\Delta}{m} \rceil$ . The next sections will present an algorithm solving  $\text{CCMC}(n, m, K, \geq 1)$  in the synchronous model described in Section 2, and a proof of it. As shown next, this algorithm is such  $K = \lceil \frac{\Delta}{m} \rceil + 1$ , and is consequently optimal with respect to the total number of colors.

**Theorem 1.** *Neither  $\text{CCMC}(n, m, K, 1)$ , nor  $\text{CCMC}(n, m, K, \geq 1)$  can be solved when  $K \leq \lceil \frac{\Delta}{m} \rceil$ .*

**Proof** Let us first show that there is no algorithm solving  $\text{CCMC}(n, m, K, 1)$  when  $K \leq \lceil \frac{\Delta}{m} \rceil$ . To this end, let us consider a process  $p_\ell$ , which has  $\Delta$  neighbors (by the very definition of  $\Delta$ , there is a such process). Let  $\Delta = m \times x + y$ , where  $0 \leq y < m$ . Hence,  $x = \frac{\Delta - y}{m} = \lfloor \frac{\Delta}{m} \rfloor$  colors are needed to color  $\Delta - y = m \times x$  processes. Moreover, if  $y \neq 0$ , one more color is needed to color the  $y < m$  remaining processes. It follows that  $\lceil \frac{\Delta}{m} \rceil$  is a lower bound to color the neighbors of  $p_\ell$ . As  $p_\ell$  cannot have the same color as any of its neighbors, it follows that at least  $\lceil \frac{\Delta}{m} \rceil + 1$  are necessary to color  $\{p_i\} \cup \text{neighbors}_i$ , which proves the theorem for  $\text{CCMC}(n, m, K, \geq 1)$ .

Let us observe that an algorithm solving  $\text{CCMC}(n, m, K, 1)$  can be obtained from an algorithm solving  $\text{CCMC}(n, m, K, \geq 1)$  by associating with each  $p_i$  a single color of its set  $\text{colors}_i$ . Hence, any algorithm solving  $\text{CCMC}(n, m, \lceil \frac{\Delta}{m} \rceil, \geq 1)$  can be used to solve  $\text{CCMC}(n, m, \lceil \frac{\Delta}{m} \rceil, 1)$ . As  $\text{CCMC}(n, m, \lceil \frac{\Delta}{m} \rceil, 1)$  is impossible to solve, it follows that  $\text{CCMC}(n, m, \lceil \frac{\Delta}{m} \rceil, \geq 1)$  is also impossible to solve.  $\square_{\text{Theorem 1}}$

### 4.2 A necessary and sufficient condition for multicoloring

Let  $\text{CCMC}(n, m, \lceil \frac{\Delta}{m} \rceil + 1, > 1)$  denote the problem  $\text{CCMC}(n, m, \lceil \frac{\Delta}{m} \rceil + 1, \geq 1)$  where at least one node obtains more than one color.

**Theorem 2.**  *$\text{CCMC}(n, m, \lceil \frac{\Delta}{m} \rceil + 1, > 1)$  can be solved on a tree of maximal degree  $\Delta$ , if and only if*

$$\exists i : \lceil \frac{\Delta}{m} \rceil + 1 > \max \left( \left\{ \left\lceil \frac{\Delta_i}{m} \right\rceil \right\} \cup \left\{ \left\lfloor \frac{\Delta_j}{m} \right\rfloor \mid p_j \in \text{neighbors}_i \right\} \right) + 1.$$

The proof of this theorem appears in Appendix A.

## 5 CCMC( $n, m, K, \geq 1$ ) in a Tree Network: Algorithm

The algorithm presented in this section use as a skeleton a parallel traversal of a tree [21]. Such a traversal is implemented by control messages that visit all the processes, followed by a control flow that returns at the process that launched the tree traversal.

Algorithm 1 is a  $C2m$ -free algorithm that solves the  $\text{CCMC}(n, m, \lceil \frac{\Delta}{m} \rceil, \geq 1)$  problem. It assumes that a single process initially receives an external message  $\text{START}()$ , which dynamically defines it as the root of the tree. This message and the fact that processes at distance smaller or equal to 2 do not have the same identity provide the initial asymmetry from which a deterministic coloring algorithm can be built. The reception of the message  $\text{START}()$  causes the receiving process (say  $p_r$ ) to simulate the reception of a fictitious message  $\text{COLOR}()$ , which initiates the sequential traversal.

**Messages** The algorithm uses two types of messages, denoted  $\text{COLOR}()$  and  $\text{TERM}()$ .

- The messages  $\text{COLOR}()$  implement a control flow visiting in parallel the processes of the tree from the root to the leaves. Each of them carries three values, denoted  $\text{sender}$ ,  $\text{cl\_map}$ , and  $\text{max\_cl}$ .

- *sender* is the identity of the sender of the message. If it is the first message `COLOR()` received by a process  $p_i$ , *sender* defines the parent of  $p_i$  in the tree.
  - *cl\_map* is a dictionary data structure with one entry for each element in  $neighbors_x \cup \{id_x\}$ , where  $p_x$  is the sender of the message `COLOR()`.  $cl\_map[id_x]$  is the set of colors currently assigned to the sender and, for each  $id_j \in neighbors_x$ ,  $cl\_map[id_j]$  is the set of colors that  $p_x$  proposes for  $p_j$ .
  - *max\_cl* is an integer defining the color domain used by the sender, namely the color set  $\{0, 1, \dots, (max\_cl - 1)\}$ . Each child  $p_i$  of the message sender will use the color domain defined by  $\max(max\_cl, \sigma_i)$  to propose colors to its own children ( $\sigma_i$  is defined below). Moreover, all the children of the sender will use the same slot span  $\{0, 1, \dots, (max\_cl - 1)\}$  to broadcast their messages. This ensures that their message broadcasts will be collision-free<sup>5</sup>.
- The messages `TERM()` propagate the return of the control flow from the leaves to the root. Each message `TERM()` carries two values: the identity of the destination process (as this message is broadcast, this allows any receiver to know if the message is for it), and the identity of the sender.

**Local variables** Each process  $p_i$  manages the following local variables. The constant  $\Delta_i = |neighbors_i|$  is the degree of  $p_i$ , while the constant  $\sigma_i = \lceil \frac{\Delta_i}{m} \rceil + 1$  is the number of colors needed to color the star graph made up of  $p_i$  and its neighbors.

- *state<sub>i</sub>* (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 states: a process in state 1 broadcasts a `COLOR()` message for its neighbors, while a process in state 3 broadcasts a message `TERM()` which has a meaning only for its parent. States 0 and 2 are waiting states in which a process listens on the broadcast channels but cannot send any message. Finally, state 4 identifies local termination.
- *parent<sub>i</sub>* stores the identity of the process  $p_j$  from which  $p_i$  receives a message `COLOR()` for the first time (hence  $p_j$  is 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  $parent_r = id_r$ .
- *colored<sub>i</sub>* is a set containing the identities of the neighbors of  $p_i$  that have been colored.
- *to\_color<sub>i</sub>* is the set of neighbors to which  $p_i$  must propagate the coloring (network traversal).
- *color\_map<sub>i</sub>* $[neighbors_i \cup \{id_i\}]$  is a dictionary data structure where  $p_i$  stores colors of its neighbors in  $color\_map_i[neighbors_i]$ , and its own colors in  $color\_map_i[id_i]$ ; *colors<sub>i</sub>* is used as a synonym of  $color\_map_i[id_i]$ .
- *max\_cl<sub>i</sub>* defines both the color domain from which  $p_i$  can color its children, and the time slots (rounds) at which its children will be allowed to broadcast.
- *slot\_span<sub>i</sub>* is set to the value *max\_cl* carried by the message `COLOR()` received by  $p_i$  from its parent. As this value is the same for all the children of its parent, they will use the same slot span to define the slots during which each child will be allowed to broadcast messages.

**Initial state** In its initial state ( $state_i = 0$ ), a process  $p_i$  waits for a message `COLOR()`. As already indicated, a single process receives the external message `START()`, which defines it at the root process. It is assumed that  $CLOCK = 0$  when a process receives this message. When it receives it, the corresponding process  $p_i$  simulates the reception of the message `COLOR( $id_i, cl\_map, \sigma_i$ )` where  $cl\_map[id_i]$  defines its color, namely,  $(CLOCK + 1) \bmod \sigma_i$  (lines 01-02). Hence, at round number 1, the root will send a message `COLOR()` to its children (lines 19-20).

---

<sup>5</sup>As we will see, conflicts are prevented by the message exchange pattern imposed by the algorithm.



```

Initialization:  $\sigma_i = \lceil \frac{\Delta_i}{m} \rceil + 1$ ;  $state_i \leftarrow 0$ ;  $colors_i \leftarrow \emptyset$ ;  $colors_i$  is a synonym of  $color\_map_i[id_i]$ .

(01) when START() is received do % a single process  $p_i$  receives this external message %
(02)  $p_i$  executes lines 04-25 as if it received the message COLOR( $id_i, cl\_map, \sigma_i$ )
    where  $cl\_map[id_i] = \{(CLOCK + 1) \bmod \sigma_i\}$ .

(03) when COLOR( $sender, cl\_map, max\_cl$ ) is received do
(04) if (first message COLOR() received)
(05) then  $parent_i \leftarrow sender$ ;  $color\_map_i[parent_i] \leftarrow cl\_map[sender]$ ;
(06)  $colored_i \leftarrow \{sender\}$ ;  $to\_color_i \leftarrow neighbors_i \setminus \{sender\}$ ;
(07)  $color\_map_i[id_i] \leftarrow cl\_map[id_i]$ ; % Synonym of  $colors_i$  %
(08)  $max\_cl_i \leftarrow \max(max\_cl, \sigma_i)$ ;  $slot\_span_i \leftarrow max\_cl$ ;
(09) if ( $to\_color_i \neq \emptyset$ ) % next lines:  $tokens_i$  is a multiset %
(10) then  $tokens_i \leftarrow \{m \text{ tokens with color } x, \text{ for each } x \in ([0..(max\_cl_i - 1)] \setminus colors_i)\}$ 
    \ { one token with color  $z$ , for each  $z \in color\_map_i[parent_i]\}$ ;
(11) while ( $|tokens_i| < |to\_color_i|$ ) do
(12) if ( $|colors_i| > 1$ ) then let  $cl \in colors_i$ ; suppress  $cl$  from  $colors_i$ 
(13) add  $m$  tokens colored  $cl$  to  $tokens_i$ 
(14) else let  $cl$  be the maximal color in  $color\_map_i[parent_i]$ ;
(15) add one token colored  $cl$  to  $tokens_i$ ;
(16)  $color\_map_i[parent_i] \leftarrow color\_map_i[parent_i] \setminus \{cl\}$ 
(17) end if
(18) end while;
(19) Extract  $|to\_color_i|$  non-empty non-intersecting multisets  $tk[id]$  (where  $id \in to\_color_i$ )
    from  $tokens_i$  such that no  $tk[id]$  contains several tokens with the the same color;
(20) for each  $id \in to\_color_i$  do  $color\_map_i[id] \leftarrow \{\text{colors of the tokens in } tk[id]\}$  end for;
(21)  $state_i \leftarrow 1$  %  $p_i$  has children %
(22) else  $state_i \leftarrow 3$  %  $p_i$  is a leaf %
(23) end if
(24) else  $color\_map_i[id_i] \leftarrow color\_map_i[id_i] \cap cl\_map[id_i]$ 
(25) end if.

(26) when ( $(CLOCK \bmod slot\_span_i) \in colors_i$ )  $\wedge$  ( $state_i \in \{1, 3\}$ ) do
(27) case ( $state_i = 1$ ) then broadcast COLOR( $id_i, color\_map_i, max\_cl_i$ );  $state_i \leftarrow 2$ 
(28) ( $state_i = 3$ ) then broadcast TERM( $parent_i, id_i$ );  $state_i \leftarrow 4$  %  $p_i$ 's subtree is colored %
(29) end case.

(30) when TERM( $dest, id$ ) is received do
(31) if ( $dest \neq id_i$ ) then discard the message (do not execute lines 25-28) end if;
(32)  $colored_i \leftarrow colored_i \cup \{id\}$ ;
(33) if ( $colored_i = neighbors_i$ )
(34) then if ( $parent_i = id_i$ ) then the root  $p_i$  claims termination else  $state_i \leftarrow 3$  end if
(35) end if.

```

Algorithm 1: C2 $m$ -free algorithm solving CCMC( $n, m, \lceil \frac{\Delta}{m} \rceil + 1, \geq 1$ ) in tree networks (code for  $p_i$ )

**Algorithm: reception of a message** COLOR() When a process  $p_i$  receives a message COLOR() for the first time, it is visited by the network traversal, and must consequently (a) obtain an initial color set, and (b) propagate the the network traversal, if it has children. The processing by  $p_i$  of this first message COLOR( $sender, cl\_map, max\_cl$ ) is done at lines 05-23. First,  $p_i$  saves the identity of its parent (the sender of the message) and its proposed color set (line 05), initializes  $colored_i$  to  $\{sender\}$ , and  $to\_color_i$  to its other neighbors (line 06). Then  $p_i$  obtains a color set proposal from the dictionary  $cl\_map$  carried by the message (line 07), computes the value  $max\_cl_i$  from which its color palette will be defined, and saves the value  $max\_cl$  carried by the message COLOR() in the local variable  $slot\_span_i$  (line 08). Let us remind that the value  $max\_cl_i$  allows it to know the color domain used up to now, and the rounds at which it will be able to broadcast messages (during the execution of the algorithm) in a

collision-free way.

Then, the behavior of  $p_i$  depends on the value of  $to\_color_i$ . If  $to\_color_i$  is empty,  $p_i$  is a leaf, and there is no more process to color from it. Hence,  $p_i$  proceeds to state 3 (line 22).

If  $to\_color_i$  is not empty,  $p_i$  has children. It has consequently to propose a set of colors for each of them, and save these proposals in its local dictionary  $color\_map_i[neighbors_i]$ . To this end,  $p_i$  computes first the domain of colors it can use, namely, the set  $\{0, 1, \dots, (max\_cl_i - 1)\}$ , and considers that each of these colors  $c$  is represented by  $m$  tokens colored  $c$ . Then, it computes the multiset<sup>6</sup>, denoted  $tokens_i$ , containing all the colored tokens it can use to build a color set proposal for each of its children (line 10). The multiset  $tokens_i$  is initially made up of all possible colored tokens, from which are suppressed (a) all tokens associated with the colors of  $p_i$  itself, and, (b) one colored token for each color in  $color\_map_i[parent_i]$  (this is because, from a coloring point of view, its parent was allocated one such colored token for each of its colors).

Then,  $p_i$  checks if it has enough colored tokens to allocate at least one colored token to each of its children (assigning thereby the color of the token to the corresponding child). If the predicate  $|tokens_i| \geq |to\_color_i|$  is satisfied,  $p_i$  has enough colored tokens and can proceed to assign set of colors to its children (lines 19-20). Differently, if the predicate  $|tokens_i| < |to\_color_i|$  is satisfied,  $p_i$  has more children than colored tokens. Hence, it must find more colored tokens. For that, if  $colors_i$  (i.e.,  $color\_map_i[id_i]$ ) has more than one color,  $p_i$  suppresses one color from  $colors_i$ , adds the  $m$  associated colored tokens to the multiset  $tokens_i$  (lines 12-13), and re-enters the “while” loop (line 11). If  $colors_i$  has a single color, this color cannot be suppressed from  $colors_i$ . In this case,  $p_i$  considers the color set of its parent ( $color\_map_i[parent_i]$ ), takes the maximal color of this set, suppresses it from  $color\_map_i[parent_i]$ , adds the associated colored token to the multiset  $tokens_i$ , and –as before– re-enters the “while” loop (line 15). Only one token colored  $cl$  is available because the  $(m - 1)$  other tokens colored  $cl$  were already added into the multiset  $tokens_i$  during its initialization at line 10.

As already said, when the predicate  $|tokens_i| < |to\_color_i|$  (line 11) becomes false,  $tokens_i$  contains enough colored tokens to assign to its children. This assignment is done at lines 19-20. Let  $ch = |to\_color_i|$  (number of children of  $p_i$ );  $p_i$  extracts  $ch$  pairwise disjoint and non-empty subsets of the multiset  $tokens_i$ , and assigns each of them to a different neighbor. “Non-empty non-intersecting multisets” used at line 19 means that, if each of  $z$  multisets  $tk[id_x]$  contains a token with the same color, this colored token appears at least  $z$  times in the multiset  $tokens_i$ .

If the message  $COLOR(sender, cl\_map, -)$  received by  $p_i$  is not the first one, it was sent by one of its children. In this case,  $p_i$  keeps in its color set  $color\_map_i[id_i]$  ( $colors_i$ ) only colors allowed by its child  $sender$  (line 24). Hence, when  $p_i$  has received a message  $COLOR()$  from each of its children, its color set  $colors_i$  has its final value.

**Algorithm: broadcast of a message** A process  $p_i$  is allowed to broadcast a message only at the rounds corresponding to a color it obtained (a color in  $colors_i = color\_map_i[id_i]$  computed at lines 07, 12, and 24), provided that its current local state is 1 or 3 (line 26).

If  $state_i = 1$ ,  $p_i$  received previously a message  $COLOR()$ , which entailed its initial coloring and a proposal to color its children (lines 09-21). In this case,  $p_i$  propagates the tree traversal by broadcasting a message  $COLOR()$  (line 27), which will provide each of its children with a coloring proposal. Process  $p_i$  then progresses to the local waiting state 2.

If  $state_i = 3$ , the coloring of the tree rooted at  $p_i$  is terminated. Process  $p_i$  consequently broadcasts the message  $TERM(parent_i, id_i)$  to inform its parent of it. It also progresses from state 3 to state 4, which indicates its local termination (line 28).

---

<sup>6</sup>Differently from a set, a *multiset* (also called a *a bag*), can contain several times the same element. Hence, while  $\{a, b, c\}$  and  $\{a, b, a, c, c, c\}$  are the same set, they are different multisets.

**Algorithm: reception of a message**  $\text{TERM}()$  When a process  $p_i$  receives such a message it discards it if it is not the intended destination process (line 31). If the message is for it,  $p_i$  adds the sender identity to the set  $\text{colored}_i$  (line 32). Finally, if  $\text{colored}_i = \text{neighbors}_i$ ,  $p_i$  learns that the subtree rooted at it is colored (line 33). It follows that, if  $p_i$  is the root ( $\text{parent}_i = i$ ), it learns that the algorithm terminated. Otherwise, it enters state 3, that will direct it to report to its parent the termination of the coloring of the subtree rooted at it.

**Solving CCMC( $n, m, K, 1$ ) in a tree** Algorithm 1 can be easily modified to solve  $\text{CCMC}(n, m, K, 1)$ . When a process enters state 3 (at line 22 or line 34), it reduces  $\text{color\_map}_i[\text{id}_i]$  (i.e.,  $\text{colors}_i$ ) to obtain a singleton.

## 6 CCMC( $n, m, K, \geq 1$ ) in a Tree Network: Cost and Proof

The proof assumes  $n > 1$ . Let us remember that  $\text{colors}_i$  and  $\text{color\_map}_i[\text{id}_i]$  are the same local variable of  $p_i$ , and  $p_r$  denotes the dynamically defined root process.

**Cost of the algorithm** Each non-leaf process broadcasts one message  $\text{COLOR}()$ , and each non-root process broadcasts one message  $\text{TERM}()$ . Let  $x$  be the number of leaves. There are consequently  $(2n - (x + 1))$  broadcasts. As  $\Delta \leq x + 1$  (<sup>7</sup>), the number of broadcast is upper bounded by  $2n - \Delta$ .

Given an execution whose dynamically defined root is the process  $p_r$ , let  $d$  be the height of the corresponding tree. The root computes the colors defining the slots (rounds) at which its children can broadcast the messages  $\text{COLOR}()$  and  $\text{TERM}()$ . These colors span the interval  $[0.. \lceil \frac{\Delta_r}{m} \rceil]$ , which means that the broadcasts of messages  $\text{COLOR}()$  by the processes at the first level of the tree span at most  $\lceil \frac{\Delta_r}{m} \rceil + 1$  rounds. The same broadcast pattern occurs at each level of the tree. It follows that the visit of the tree by the messages  $\text{COLOR}()$  requires at most  $d \lceil \frac{\Delta}{m} \rceil$  rounds. As the same occurs for the the messages  $\text{TERM}()$ , returning from the leaves to the root, it follows that the time complexity of the algorithm is  $O(d \lceil \frac{\Delta}{m} \rceil)$ .

**Lemma 1.** *Algorithm 1 is conflict-free.*

**Proof** The algorithm uses two types of messages:  $\text{COLOR}()$  and  $\text{TERM}()$ . We first show conflict-freedom for  $\text{COLOR}()$  messages (if a process broadcasts a message  $\text{COLOR}()$ , none of its neighbors is broadcasting any message in the same round). Let us first notice that a process  $p_i$  broadcasts at most one message  $\text{COLOR}()$ , and one message  $\text{TERM}()$  (this is due to the guard  $\text{state}_i \in \{1, 3\}$ , line 26, and the fact that the broadcast of a message makes its sender progress to the waiting state 2 or 4). Moreover, let us make the following observations.

- Observation 1: The first message sent by any node is of type  $\text{COLOR}()$  (line 27).
- Observation 2: Except for the root process, a message  $\text{COLOR}()$  is always broadcast by a process after it received a message  $\text{COLOR}()$  (which triggers the execution of lines 03-25).
- Observation 3: Except for leaf processes, a message  $\text{TERM}()$  is always broadcast by a process after it received a message  $\text{TERM}()$  from each of its children (lines 30-35 and line 28.).

Observations 1 and 2 imply that when the root process broadcasts its  $\text{COLOR}()$  message, none of its neighbors is broadcasting a message, and they all receive the root's  $\text{COLOR}()$  message without conflict. Let us now consider a process  $p_i$ , different from the root, which receives its first message  $\text{COLOR}_k()$  (from its parent  $p_k$ ). Because there is no cycle in the communication graph (a tree), all the children of  $p_i$

---

<sup>7</sup>Let  $p_i$  be the process that has  $\Delta$  as degree. If  $p_i$  is the root of the tree, the tree contains at least  $\Delta$  leaf processes. This is because each neighbor of  $p_i$  is either a leaf or the root of a subtree that has at least one leaf process. And if  $p_i$  is not the root of the tree,  $p_i$  possesses  $\Delta - 1$  children, and the number of leaf processes is at least  $\Delta - 1$  following a similar reasoning.

$(neighbors_i \setminus \{p_k\})$  are in state 0, waiting for their  $COLOR()$  message. Moreover, due to Observations 1 and 2, they will receive from  $p_i$  their message  $COLOR()$  without conflict. After sending its  $COLOR()$  message,  $p_i$ 's parent  $p_k$  remains in the waiting state 2 until it receives a  $TERM()$  message from all its children (lines 32-33), which include  $p_i$ . As a consequence,  $p_k$  is not broadcasting any message in the round in which it receives  $p_i$ 's  $COLOR()$  message, which is consequently received without conflict by all its neighbors.

As far the messages  $TERM()$  are concerned we have the following. Initially, only a leaf process can broadcast a message, and when it does it, its parent is in the waiting state 2 (since it broadcast a message  $COLOR()$  at line 27 and it must receive messages  $TERM()$  to proceed to state 3). Hence a message  $TERM()$  broadcast by a leaf cannot entail conflict. Let us now consider a non-leaf process  $p_i$ . It follows from Observation 3 that  $p_i$  can broadcast a message  $TERM()$  only when its children are in state 4 (in which they cannot broadcast), and its parent (because it has not yet received a message  $TERM()$  from each of its children) is in the waiting state 2. Hence, we conclude that the broadcast of a message  $TERM()$  by a non-leaf process is conflict-free, which concludes the proof of the lemma.  $\square_{Lemma 1}$

**Definition** A message  $COLOR(sender, cl\_map, max\_cl)$  is *well-formed* if its content satisfies the following properties. Let  $sender = id_i$ .

- M1 The keys of the dictionary data structure  $cl\_map$  are the identities in  $neighbors_i \cup \{id_i\}$ .
- M2  $\forall id \in (neighbors_i \cup \{id_i\}) : cl\_map[id] \neq \emptyset$ .
- M3  $\forall id \in neighbors_i : cl\_map[id] \cap cl\_map[id_i] = \emptyset$ .
- M4  $\forall c : |\{j : (id_j \in neighbors_i) \wedge (c \in cl\_map[id_j])\}| \leq m$ .
- M5  $1 < max\_cl \leq \lceil \frac{\Delta}{m} \rceil + 1$ .
- M6  $\forall id \in (neighbors_i \cup \{id_i\}) : cl\_map[id] \subseteq [0..max\_cl - 1]$ .

Once established in Lemma 3, not all properties M1-M6 will be explicitly used in the lemmas that follow. They are used by induction to proceed from one well-formed message to another one.

**Lemma 2.** *If a message  $COLOR(sender, cl\_map, max\_cl)$  received by a process  $p_i \neq p_r$  is well-formed and entails the execution of lines 05-23, the **while** loop (lines 11-18) terminates, and, when  $p_i$  exits the loop, the sets  $colors_i$  and  $color\_map_i[parent_i]$  are not empty, and their intersection is empty.*

**Proof** Let us consider a process  $p_i \neq p_r$  that receives a well-formed  $COLOR_j(sender, cl\_map, max\_cl)$  message from  $p_j$ . Let us assume  $COLOR()$  causes  $p_i$  to start executing the lines 05-23, i.e.,  $COLOR()$  is the first such message received by  $p_i$ . The body of the **while** loop contains two lines (lines 12 and 14) that select elements from two sets,  $colors_i$  and  $color\_map_i[parent_i]$  respectively.

Before discussing the termination of the **while** loop, we show that lines 12 and 14 are *well-defined*, i.e. the sets from which the elements are selected are non-empty. To this aim, we prove by induction that the following invariant holds in each iteration of the loop:

$$color\_map_i[parent_i] \neq \emptyset, \tag{1}$$

$$colors_i \neq \emptyset, \tag{2}$$

$$|tokens_i| = m \times max\_cl_i - m \times |colors_i| - |color\_map_i[parent_i]|. \tag{3}$$

Just before the loop (i.e., before line 11), Assertion (1) follows from the assignment to  $color\_map_i[parent_i]$  at line 05 and the property M2 of  $COLOR_j()$  ( $id_j = parent_i$ ). Assertion (2) also follows from M2 ( $colors_i$  is synonym of  $color\_map_i[id_i]$ ). Assertion (3) follows from M3, M6, and the initialization of  $max\_cl_i$  at line 08.

Let us now assume that Assertion (1) holds at the start of a loop iteration (i.e., just before lines 12). There are two cases.

- If  $|colors_i| > 1$ , lines 14-16 are not executed, and consequently  $color\_map_i[parent_i]$  is not modified. It follows from the induction assumption that Assertion (1) still holds.
- If  $|colors_i| \leq 1$ , we have the following. Because we are in the **while** loop, we have  $|tokens_i| < |to\_colors_i|$ , which, combined with Assertion (3), implies

$$|to\_colors_i| > m \times max\_cl_i - m \times |colors_i| - |color\_map_i[parent_i]|,$$

from which we derive

$$\begin{aligned} |color\_map_i[parent_i]| &> m \times max\_cl_i - m \times |colors_i| - |to\_colors_i|, \\ &> m \times max\_cl_i - m \times |colors_i| - (\Delta_i - 1) && \text{(because of line 06),} \\ &> m \times \sigma_i - m \times |colors_i| - (\Delta_i - 1) && \text{(because of line 08),} \\ &> m \times (\lceil \frac{\Delta_i}{m} \rceil + 1) - m \times |colors_i| - (\Delta_i - 1) && \text{(by definition),} \\ &> \Delta_i + m - m \times |colors_i| - (\Delta_i - 1) && \text{(arithmetic),} \\ &> m \times (1 - |colors_i|) + 1. \end{aligned}$$

Hence, because  $|colors_i| \leq 1$ , we obtain  $|color\_map_i[parent_i]| > 1$ , which means that  $p_i$ 's local variable  $color\_map_i[parent_i]$  contains at least two elements before the execution of line 12. Because only one color is removed from  $color\_map_i[parent_i]$ , this local variable remains non-empty after line 16, thus proving Assertion (1).

Let us now assume that both Assertion (2) and Assertion (3) hold at the start of a loop iteration (i.e., just before line 12). There are two cases.

- Case  $|colors_i| > 1$ . In this case we have: (i) one color is removed from  $colors_i$ , (ii)  $m$  colored tokens are added to  $tokens_i$ , and (iii)  $color\_map_i[parent_i]$  remains unchanged.  $|colors_i| > 1$  and (i) imply that Assertion (2) remains true; and (i) and (ii) mean that Assertion (3) is preserved.
- Case  $|colors_i| \leq 1$ . In this case we have: (i) one color is removed from  $color\_map_i[parent_i]$ , and one colored token added to  $tokens_i$ , and (ii)  $colors_i$  stays unchanged. (i) implies that Assertion (3) remains true, and (ii) ensures Assertion (2) by assumption.

This concludes the proof that the three assertions (1)–(3) are a loop invariant. Hence, Assertion (1) and Assertion (2) imply that lines 12 and 14 are well-defined.

Let us now observe that, in each iteration of the loop, new colored tokens are added to  $tokens_i$ , and thus  $|tokens_i|$  is strictly increasing. Because  $|to\_color_i|$  remains unchanged, the condition  $|tokens_i| < |to\_color_i|$  necessarily becomes false at some point, which proves that the loop terminates.

Just after the loop, the invariant is still true. In particular Assertion (1) and Assertion (2) show that both the sets  $colors_i$  and  $color\_map_i[parent_i]$  are not empty when  $p_i$  exits the **while** loop.

Finally, due to the fact that the message  $COLOR_j()$  is well-formed, it follows from M3 that we have  $colors_i \cap color\_map_i[parent_i] = \emptyset$  after line 07. As colors are added neither to  $colors_i$ , nor to  $color\_map_i[parent_i]$  in the loop, their intersection remains empty, which concludes the proof of the lemma.  $\square$ Lemma 2

**Lemma 3.** *All messages  $COLOR()$  broadcast at line 27 are well-formed.*

**Proof** To broadcast a message  $COLOR()$ , a process  $p_i$  must be in local state 1 (line 27). This means that  $p_i$  executed line 21, and consequently previously received a message  $COLOR(sender, cl\_map, max\_cl)$  that caused  $p_i$  to execute lines 05-23.

Let us first assume that  $COLOR()$  is well-formed. It then follows from Lemma 2 that  $p_i$  exits the while loop, and each of  $colors_i$  and  $color\_map_i[parent_i]$  is not empty (A), and they have an empty intersection (B). When considering the message  $COLOR(id_i, color\_map_i, max\_cl_i)$  broadcast by  $p_i$  we have the following.



- M1 follows from the fact that the entries of the dictionary data structure created by  $p_i$  are:  $color\_map_i[parent_i]$  (line 05),  $color\_map_i[id_i]$  (line 07), and  $color\_map_i[id]$  for each  $id \in to\_colors_i = neighbors_i \setminus \{parent_i\}$  (lines 06 and 20), and the observation that no entry is ever removed from  $color\_map_i$  is the rest of the code.
- M2 follows from (A) for  $color\_map_i[parent_i]$  and  $color\_map_i[id_i]$ , from line 19 for the identities in  $to\_colors_i = neighbors_i \setminus \{parent_i\}$  (due to  $|tokens_i| \geq |to\_colors_i|$  when line 19 is executed, and the non-intersection requirement of the  $tk[id]$  sets, no  $tk[id]$  is empty), and from the observation that  $color\_map_i$  is not modified between the end of line 20 and the broadcast of line 27. This last claim is derived from the fact that  $color\_map_i$  is only modified when messages are received, and that neither  $p_i$ 's parent nor  $p_i$ 's children are in states that allow them to send messages while  $p_i$  is transitioning from line 20 to line 27.
- Similarly M3 follows
  - for  $id = parent_i$ : from (B) and the fact that  $color\_map_i[parent_i]$  never increases,
  - for  $id \in to\_color_i = neighbors_i \setminus \{parent_i\}$ : from the fact that, due to lines 10 and 12, at line 19  $tokens_i$  contains no token whose color belongs to  $colors_i$ , from which we have  $tk[id] \cap color\_map_i[id_i] = \emptyset$  for any  $id \in to\_color_i$ .
- M4 follows from the construction of  $tokens_i$ . This construction ensures that, for any color  $c$ ,  $tokens_i$  contains at most  $m$  tokens with color  $c$  (line 10, 13, and 15).
- M5 is an immediate consequence of the assignment  $max\_cl_i \leftarrow \max(max\_cl, \sigma_i)$  at line 15.
- M6 follows from the following observations:
  - for  $id \in \{id_i, parent_i\}$ : from  $max\_cl \leq max\_cl_i$  (line 08) and the fact that the message  $COLOR()$  received by  $p_i$  is well-formed (hence  $color\_map_i[id_i] \cup color\_map_i[parent_i] \subseteq [0..(max\_cl - 1)]$ ),
  - for  $id \in to\_color_i = neighbors_i \setminus \{parent_i\}$ : from the fact that  $tokens_i$  contains only tokens whose color is in  $[0..(max\_cl_i - 1)]$  (line 10).

The previous reasoning showed that, if a process receives a well-formed message  $COLOR()$ , executes lines 05-23 and line 27, the message  $COLOR(id_i, color\_map_i, max\_cl_i)$  it will broadcast at this line is well-formed. Hence, to show that all messages broadcast at line 27 are well-formed, it only remains to show that the message  $COLOR(id_r, color\_map_r, max\_cl_r)$  broadcast by the root  $p_r$  is well-formed. Let us remember that  $neighbors_r$  is a constant defined by the structure of the tree, and  $parent_r = id_r \notin neighbors_r$ .

Let us notice that the message  $COLOR(id, cl\_map, max\_cl)$ , that  $p_r$  sends to itself at line 02, is not well-formed. This is because,  $cl\_map[id]$  is not defined for  $id \in neighbors_i$ . When  $p_r$  receives this message we have the following after line 10:

$$|tokens_r| = m \times |\sigma_r| - m = m \times (|\sigma_r| - 1) = m \lceil \frac{\Delta_r}{m} \rceil \geq \Delta_r,$$

from which we conclude  $|tokens_r| \geq \Delta_r = |to\_colors_r| = |neighbors_i|$ . Hence,  $p_r$  does not execute the loop body, and proceeds to lines 19-20 where it defines the entries  $color\_map_r[id]$  for  $id \in to\_colors_r = neighbors_r$ . A reasoning similar to the previous one shows that the message  $COLOR(id_r, color\_map_r, max\_cl_r)$  broadcast by  $p_r$  at line 27 satisfies the properties M1-M6, and is consequently well-formed. (The difference with the previous reasoning lies in the definition of the set  $to\_colors_i$  which is equal to  $neighbors_i \setminus \{parent_i\}$  for  $p_i \neq p_r$ , and equal to  $neighbors_r$  for  $p_r$ .)

□ Lemma 3

**Lemma 4.** *If a process  $p_i$  computes a color set ( $colors_i$ ), this set is not empty.*



**Proof** Let us first observe that, if a process  $p_i \neq p_r$  receives a message  $\text{COLOR}(-, cl\_map, -)$ , the previous lemma means that this message is well-formed, and due to property M2, its field  $cl\_map[id_i]$  is not empty, from which follows that the initial assignment of a value to  $color\_map_i[id_i] \equiv colors_i$  is a non-empty set. Let us also observe, that, even if it is not well-formed the message  $\text{COLOR}(-, cl\_map, -)$  received by the root satisfies this property. Hence, any process that receives a message  $\text{COLOR}()$  assigns first a non-empty value to  $color\_map_i[id_i] \equiv colors_i$ .

Subsequently, a color can only be suppressed from  $color\_map_i[id_i] \equiv colors_i$  at line 24 when  $p_i$  receives a message  $\text{COLOR}()$  from one of its children. If  $p_i$  is a leaf, it has no children, and consequently never executes line 24. So, let us assume that  $p_i$  is not a leaf and receives a message  $\text{COLOR}(id_j, cl\_map, -)$  from one of its children  $p_j$ . In this case  $p_i$  previously broadcast at line 27 a message  $\text{COLOR}(id_i, color\_map_i, -)$  that was received by  $p_j$  and this message is well-formed (Lemma 3).

A color  $c$  that is suppressed at line 24 when  $p_i$  processes  $\text{COLOR}(id_j, cl\_map, -)$  is such that  $c \in colors_i$  and  $c \notin cl\_map[id_i]$ .  $cl\_map[id_i]$  can be traced back to the local variable  $color\_map_j[id_i]$  used by  $p_j$  to broadcast  $\text{COLOR}()$  at line 27. Tracing the control flow further back,  $color\_map_j[id_i]$  was initialized by  $p_j$  to  $color\_map_i[id_i]$  (line 05) when  $p_j$  received the well-formed message  $\text{COLOR}()$  from  $p_i$ . When processing  $\text{COLOR}()$  received from  $p_i$ , process  $p_j$  can suppress colors from  $color\_map_j[id_i]$  only at line 16, where it suppresses colors starting from the greatest remaining color. We have the following.

- If  $p_i$  is not the root, the message  $\text{COLOR}()$  it received was well-formed (Lemma 3). In this case, it follows from the proof of Lemma 2 that it always remains at least one color in  $color\_map_j[id_i]$ .
- If  $p_i = p_r$ , its set  $colors_r$  is a singleton (it “received”  $\text{COLOR}(id_r, cl\_map_r, -)$  where  $cl\_map_r$  has a single entry, namely  $cl\_map_r[id_r] = \{1\}$ ). When  $p_j$  computes  $tokens_j$  (line 10) we have  $|tokens_j| = m \times \max(|\sigma_r|, |\sigma_j|) - m = m \lceil \frac{\max(\Delta_r, \Delta_j)}{m} \rceil \geq \max(\Delta_r, \Delta_j) \geq \Delta_j = |to\_colors_j|$ , from which follows that  $|tokens_j| \geq |to\_colors_j| = |neighbors_j| - 1$ . Hence,  $p_j$  does not execute the loop, and consequently does not modify  $color\_map_j[id_r]$ .

Consequently, the smallest color of  $colors_i \equiv color\_map_i[id_i]$  is never withdrawn from  $color\_map_j[id_i]$ . It follows that, at line 24,  $p_i$  never withdraws its smallest color from the set  $color\_map_j[id_i]$ .  $\square_{\text{Lemma 4}}$

**Lemma 5.** *If  $p_i$  and  $p_j$  are neighbors  $colors_i \cap colors_j = \emptyset$ .*

**Proof** As all color sets are initialized to  $\emptyset$ , the property is initially true. We show that, if a process receives a message  $\text{COLOR}()$ , the property remains true. As  $\text{TERM}()$  messages do not modify the coloring—lines 30-35—they do not need to be considered.

Let us consider two neighbor processes  $p_i$  and  $p_j$ , which computes their color sets (if none or only one of  $p_i$  and  $p_j$  computes its color set, the lemma is trivially satisfied). As the network is a tree, one of them is the parent of the other. Let  $p_i$  be the parent of  $p_j$ .

Process  $p_i$  broadcast a message  $\text{COLOR}(-, cl\_map, -)$  at line 27 in which the set  $cl\_map[id_j]$  is  $color\_map_i[id_j]$ , as computed at line 19. If this message is received by  $p_j$ , this set will in turn be assigned to  $color\_map_i[id_j]$  at  $p_j$ . As this message is well-formed (Lemma 3), we therefore have  $color\_map_i[id_i] \cap color\_map_j[id_j] = \emptyset$  (Property M3 of a well-formed message). Then, while  $p_i$  can be directed to suppress colors from  $color\_map_i[id_i]$  at line 24, it never adds a color to this set. The same is true for  $p_j$  and  $color\_map_j[id_j]$ . It follows that the predicate  $color\_map_i[id_i] \cap color\_map_j[id_j] = \emptyset$  can never be invalidated.  $\square_{\text{Lemma 5}}$

**Lemma 6.**  $\forall i, \forall c : |\{j : j \in neighbors_i \wedge c \in colors_j\}| \leq m$ .

**Proof** The property is initially true. We show that it remains true when processes receive messages.

Let us consider a process  $p_i$  that broadcasts a message  $\text{COLOR}()$ . Due to the fact that such messages are broadcast only at line 27, it follows from Lemma 3 that the message  $\text{COLOR}(id_i, cl\_map, -)$  broadcast by  $p_i$  is well-formed. Hence it satisfies property M4. When processing this message

- A each child  $p_j$  of  $p_i$  adopts  $cl\_map[id_j]$  as its initial color set and assigns it to  $color\_map_j[id_j]$ ;
- B  $p_i$ 's parent  $p_k$  uses  $cl\_map[id_k]$  to update  $color\_map_k[id_k]$  at line 24 such that  $color\_map_k[id_k] \subseteq cl\_map[id_k]$ .

(A), (B), and M4 imply that just after  $p_i$ 's neighbors have processed  $p_i$ 's message, the lemma holds. As already seen in the proof of other lemmas,  $color\_map_j[id_j]$  may subsequently decrease, but never increases: colors can be suppressed from  $color\_map_j[id_j]$  (line 24) but never added to it. And the same is true at  $p_i$  for its set of colors  $color\_map_i[id_i]$ , and at its parent  $p_k$  for  $color\_map_k[id_k]$ . It then follows that  $|\{j : j \in neighbors_i \wedge c \in colors_j\}| \leq m$  throughout the execution of the algorithm, which concludes the proof of the lemma.  $\square$  Lemma 6

**Lemma 7.** *Algorithm 1 is collision-free.*

**Proof** We have to show that no process can have more than  $m$  of its neighbors that broadcast during the same round. Initially, all processes are in state 0. Let us consider a process  $p_i$  and assume that one of its neighbors  $p_j$  is broadcasting a message. Let us further assume that this message is of type COLOR().

- If  $p_j$  is  $p_i$ 's parent,  $p_j$ 's COLOR() message is the first message received by  $p_i$ , and both  $p_i$  and its children ( $p_i$ 's remaining neighbors) are in state 0, and hence silent. There is no collision at  $p_i$ .
- If  $p_j$  is one of  $p_i$ 's children, the value  $slot\_span_j$  used by  $p_j$  at line 26 is equal to  $max\_cl$  contained in the message COLOR( $-, -, max\_cl$ ) first received by  $p_j$  from  $p_i$ . Because of Lemma 3, this message is well-formed, and consequently satisfies property M6. Any other child  $p_\ell$  of  $p_i$  broadcasting during this round will have received the same first message, and will therefore be using the same  $slot\_span_\ell = max\_cl$  value. It follows from Property M6, the assignment of line 07 executed by any child  $p_\ell$  (of  $p_i$ ) that received the message, and the fact that its set  $colors_\ell$  can only decrease after being first assigned, that  $colors_\ell \subseteq [0..slot\_span_\ell - 1]$  for any child  $p_\ell$  of  $p_i$  (C).

Lemma 6, Property (C), and the CLOCK-based predicate defining the rounds at which a process is allowed to broadcast (line 26), imply that at most  $m$  children of  $p_i$  can broadcast during the same round. If  $p_i$  has a parent  $p_k$  (i.e.  $p_i$  is not the root), both  $p_i$  and  $p_k$  are in state 2, and hence  $p_k$  is silent, proving the lemma. If  $p_i$  is the root, all its neighbors are its children, and the lemma also holds.

The same reasoning applies to the messages TERM() broadcast by the children of  $p_i$  and its parent.

$\square$  Lemma 7

**Lemma 8.** *Each process computes a set of colors, and the root process knows when coloring is terminated.*

**Proof** Let us first observe that, due to Lemmas 2 and 3, no process  $p_i \neq p_r$  can loop forever inside the **while** loop (lines 11-18), when it receives its first message COLOR(). The same was proved for the root  $p_r$  at the end of the proof of Lemma 4. Moreover, a process cannot block at line 24 when it receives other messages COLOR() (one from each of its children). Hence, no reception of a message COLOR() can prevent processes from terminating the processing of the message. The same is trivially true for the processing of a message TERM().

Let us first show that each process obtains a non-empty set of colors. To this end, we show that each non-leaf process broadcasts a message COLOR().

- When the root process  $p_r$  receives the external message START(), it “simulates the sending to itself” of the message COLOR( $id_r, color\_map_r, \sigma_r$ ), where the dictionary data structure  $color\_map_r$  has a single element, namely,  $color\_map_r[id_r] = \{1\}$ . The root  $p_r$  executes consequently the

lines 25-23, during which it obtains a color ( $color\_map_r[id_r] = \{1\}$ , line 07), and computes a set of proposed colors  $color\_map_r[id_j]$  for each of its children  $p_j$  (lines 19-20). It then progresses to the local non-waiting state 1 (line 21). Hence, during the first round, it broadcasts to its neighbors the message  $COLOR(id_r, color\_map_r, \sigma_r)$ . Because the algorithm is conflict- and collision-free (Lemmas 1 and 5), this message is received by all the root's neighbors.

- Let us now consider a process  $p_i$  that receives a message  $COLOR(sender, color\_map, max\_cl)$  for the first time. It follows from Lemma 4 that  $p_i$  starts computing a non-empty set  $colors_i$  and enters the waiting state 1 (line 21). Finally, as  $colors_i \subseteq [0..slot\_span_i - 1]$ , and  $CLOCK$  never stops increasing, the predicate of line 26 is eventually satisfied. It follows  $p_i$  broadcasts the message  $COLOR(id_i, color\_map_i, max\_cl_i)$ . As above all of  $p_i$ 's neighbors will receive this message.

It follows that  $COLOR()$  messages flood the tree from the root to the leaves.

Moreover, when a process  $p_i$  has received a message  $COLOR()$  from each of its neighbors (children and parent), it has obtained the final value of its color set  $color\_map_i[id_i] = colors_i$ . Due to lemma 4, this set is not empty, which concludes the first part of the proof.

Let us now show that the root learns coloring termination. This relies on the messages  $TERM()$ . As previously, due to Lemma 1 and Lemma 7, these messages entail neither message conflicts nor message collisions.

Let us observe that each leaf process enters the non-waiting state 3. When the predicate of line 26 is satisfied at a leaf  $p_\ell$  (this inevitably occurs), this process broadcasts the message  $TERM()$  to its parent  $p_i$ . Then, when  $p_i$  has received a message  $TERM()$  from each of its children, it broadcasts  $TERM()$  to its own parent. This sequence repeats itself on each path from a leaf to the root. When the root has received a message  $TERM()$  from each of its children, it learns termination (line 34), which concludes the proof of the lemma.  $\square_{Lemma\ 8}$

**Lemma 9.**  $|\bigcup_{1 \leq i \leq n} colors_i| = \lceil \frac{\Delta}{m} \rceil + 1$ .

**Proof** Let  $p_r, p_a, \dots, p_\ell$  be a path in the tree starting at the root  $p_r$  and ending at a leaf  $p_\ell$ . It follows from

- the content of the parameter  $max\_cl$  of the messages  $COLOR(sender, cl\_map, max\_cl)$  broadcast along this path of the tree (broadcast at line 27 and received at line 03), and
- the assignment of  $\max(max\_cl, \sigma_i)$  to  $max\_cl_i$  at line 08,

that  $max\_cl_\ell = \max(\sigma_r, \sigma_a, \dots, \sigma_\ell)$ . Let  $p_{\ell_1}, \dots, p_{\ell_x}$  be the set of leaves of the tree. It follows that  $\max(max\_cl_{\ell_1}, \dots, max\_cl_{\ell_x}) = \max(\sigma_1, \dots, \sigma_n)$ , i.e., the value  $max\_cl$  carried by any message is  $\leq \lceil \frac{\Delta}{m} \rceil + 1$ .

The fact that a process  $p_i$  uses only colors in  $[0..(max\_cl_i - 1)]$ , combined with Theorem 1 implies the lemma. The algorithm is consequently optimal with respect to the number of colors.  $\square_{Lemma\ 9}$

**Theorem 3.** Let  $K = \lceil \frac{\Delta}{m} \rceil + 1$ . Algorithm 1 is a  $C2m$ -free algorithm, which solves  $CCMC(n, m, K, \geq 1)$  in tree networks. Moreover, it is optimal with respect to the value of  $K$ .

**Proof** The proof that Algorithm 1 is  $C2m$ -free follows from Lemma 1 and Lemma 7. The proof that it satisfies the Conflict-freedom, Collision-freedom, and Efficiency properties defining the  $CCMC(n, m, K, \geq 1)$  problem follows from Lemmas 2-6, and Lemma 8. The proof of its optimality with respect to  $K$  follows from Lemma 9.  $\square_{Theorem\ 3}$

## 7 Conclusion

The paper first introduced a new vertex coloring problem (called CCMC), in which a process may be assigned several colors in such a way that no two neighbors share colors, and for any color  $c$ , at most  $m$  neighbors of any vertex share the color  $c$ . This coloring problem is particularly suited to assign rounds (slots) to processes (nodes) in broadcast/receive synchronous communication systems with communication or local memory constraints. Then, the paper presented a distributed algorithm which solve this vertex coloring problem for tree networks in a round-based programming model with conflicts and (multi-frequency) collisions. This algorithm is optimal with respect to the total number of colors that can be used, namely it uses only  $K = \lceil \frac{\Delta}{m} \rceil + 1$  different colors, where  $\Delta$  is the maximal degree of the graph.

It is possible to easily modify the coloring problem CCMC to express constraints capturing specific broadcast/receive communication systems. As an example, suppressing the conflict-freedom constraint and weakening the collision-freedom constraint into

$$\forall i, \forall c : |\{j : (id_j \in neighbors_i \cup \{id_i\}) \wedge (c \in colors_j)\}| \leq m, \quad (4)$$

captures bi-directional communication structures encountered in some practical systems in which nodes may send and receive on distinct channels during the same round. Interestingly, solving the coloring problem captured by (4) is equivalent to solving distance-2 coloring in the sense that a purely local procedure (i.e., a procedure involving no communication between nodes) executed on each node can transform a classical distance-2 coloring into a multi-coloring satisfying (4). More precisely, assuming a coloring  $col : V \mapsto [0..(K * m) - 1]$  providing a distance-2 coloring with  $K * m$  colors on a graph  $G = (V, E)$ , it is easy to show that the coloring (with one color per vertex)

$$\begin{aligned} col' : V &\mapsto [0 .. K - 1] \\ x &\rightarrow col(x) \bmod K, \end{aligned} \quad (5)$$

fulfills (4) on  $G$  <sup>(8)</sup>. Since the distance-2 problem with  $K * m$  colors is captured by  $CCMC(n, 1, K * m, 1)$  (as discussed in Section 3), the proposed algorithm can also solve the coloring condition captured by (4) on trees in our computing model.

Moreover, from an algorithmic point of view, the proposed algorithm is versatile, making it an attractive starting point to address other related problems. For instance, in an heterogeneous network, lines 19-20 could be modified to take into account additional constraints arising from the capacities of individual nodes, such as their ability to use only certain frequencies.

Last but not least, a major challenge for future work consists in solving the CCMC problem in general graphs. The new difficulty is then to take into account cycles.

## 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. and Welch J., *Distributed computing: fundamentals, simulations and advanced topics*, (2d Edition), Wiley-Interscience, 414 pages (2004)
- [3] Barenboim L. and Elkin M., Deterministic distributed vertex coloring in polylogarithmic time. *Journal of the ACM*, 58(5):23 (2011)
- [4] Barenboim L. and Elkin M., *Distributed graph coloring, fundamental and recent developments*, Morgan & Claypool Publishers, 155 pages (2014)

---

<sup>8</sup>This is because (a) distance-2 coloring ensures that any vertex and its neighbors have different colors, and (b) there are at most  $m$  colors  $c_1, \dots, c_x \in [0..(K * m) - 1]$  (hence  $x \leq m$ ), such that  $(c_1 \bmod K) = \dots = (c_x \bmod K) = c \in [0..(K - 1)]$ .

- [5] 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)
- [6] 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)
- [7] 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)
- [8] 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)
- [9] 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)
- [10] Cole R. and Vishkin U., Deterministic coin tossing with applications to optimal parallel list ranking. *Information and Control*, 70(1):32-53 (1986)
- [11] Frey D., Lakhlef H., Raynal M., Optimal collision/conflict-free distance-2 coloring in synchronous broadcast/receive tree networks. *Research Report*, <https://hal.inria.fr/hal-01248428> (2015)
- [12] 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)
- [13] Gebremedhin A.H., Manne F., and Pothen A., Parallel distance- $k$  coloring algorithms for numerical optimization. *Proc. European Conference on Parallel Processing (EUROPAR)*, Springer LNCS 2400, pp. 912-921 (2002)
- [14] Goldberg A., Plotkin S., and Shannon G., Parallel symmetry-breaking in sparse graphs. *SIAM Journal on Discrete Mathematics*, 1(4):434-446 (1988)
- [15] 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)
- [16] 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)
- [17] Linial N., Locality in distributed graph algorithms. *SIAM Journal on Computing*, 21(1):193-201 (1992)
- [18] Lynch N.A., *Distributed algorithms*. Morgan Kaufmann, 872 pages (1996)
- [19] Peleg D., *Distributed computing, a locally sensitive approach*. SIAM Monographs on Discrete Mathematics and Applications, 343 pages, ISBN 0-89871-464-8 (2000)
- [20] Raynal M., *Fault-tolerant agreement in synchronous message-passing systems*. Morgan & Claypool Publishers, 165 pages, (ISBN 978-1-60845-525-6) (2010)
- [21] Raynal M., *Distributed algorithms for message-passing systems*. Springer, 500 pages, ISBN 978-3-642-38122-5 (2013)

## A Proof of Theorem 2

**Lemma 10.**  $K' > \lfloor \frac{\Delta_j}{m} \rfloor \iff K' \times m > \Delta_j$ .

**Proof** From  $K' > \lfloor \frac{\Delta_j}{m} \rfloor$  we can derive the following. Because  $K'$  is an integer, we have  $K' \geq \lfloor \frac{\Delta_j}{m} \rfloor + 1$ . Because  $\lfloor x \rfloor > x - 1$ , we obtain  $K' > \frac{\Delta_j}{m} - 1 + 1$ , from which we conclude  $K' \times m > \Delta_j$ . Conversely, if we have  $K' \times m > \Delta_j$ , then  $K' > \frac{\Delta_j}{m}$ , and, because  $x \geq \lfloor x \rfloor$ , we obtain  $K' > \frac{\Delta_j}{m} \geq \lfloor \frac{\Delta_j}{m} \rfloor$ , which concludes the proof of the lemma. □ Lemma 10

**Theorem 2** Let  $K = \lceil \frac{\Delta}{m} \rceil + 1$ .  $\text{CCMC}(n, m, K, > 1)$  can be solved on a tree of maximal degree  $\Delta$ , if and only if

$$\exists i : K > \max \left( \left\{ \left\lceil \frac{\Delta_i}{m} \right\rceil \right\} \cup \left\{ \left\lfloor \frac{\Delta_j}{m} \right\rfloor \mid p_j \in \text{neighbors}_i \right\} \right) + 1.$$



**Proof** The terms “process” and “vertex” are considered here as synonyms. To simplify notation, we consider in the following that  $id_i = i$ . Let us first notice that it follows from its definition that  $K \geq 2$ .

*Proof of the “if direction”.*

The proof of this direction consists in a sequential algorithm that associates two colors to a process  $p_i$  whose position in the tree satisfies the previous predicate.

Algorithm 2 is a sequential algorithm solving  $CCMC(n, m, \lceil \frac{\Delta}{m} \rceil + 1, 1)$ . Using the control flow defined by a simple depth-first tree traversal algorithm, it takes two input parameters, a process  $p_j$ , and its color. Then, assuming a coloring of both  $p_j$  and its parent, it recursively colors the vertices of the tree rooted at  $p_j$ . The initial call is  $DF\_MColoring(i, 0)$  where  $p_i$  is a vertex satisfying the predicate stated in the theorem, and 0 the color assigned to it. The function  $parent\_color(j)$  returns the color of the parent of  $p_j$  if  $p_j \neq p_i$  and returns no value if  $p_j = p_i$ . Let us notice that, except for  $p_j = p_i$ ,  $parent\_color(j)$  is called only after the parent of  $p_j$  obtained a color.

```

procedure DF_MColoring( $j, c$ ) is
(01)   $color_i \leftarrow c$ ;
(02)  if  $|neighbors_j| > 1$  then
(03)     $tokens \leftarrow \{ m \text{ colored tokens for each color in } \{0, 1, \dots, (K-1)\} \setminus \{color_j, parent\_color(j)\} \}$ 
         $\cup \{ (m-1) \text{ tokens with color } parent\_color(j) \}$ ;
(04)    for each  $p_k \in (neighbors_j \setminus \{parent_j\})$  do  %this loop is executed  $(\Delta_j - 1)$  times %
(05)       $token \leftarrow$  a smallest token in  $tokens$ ; suppress  $token$  from  $tokens$ ;
(06)      DF_MColoring( $k, color$  of  $token$ )
(07)    end for
(08)  end if.

```

Algorithm 2: Sequential multi-coloring of a tree with a depth-first traversal algorithm (code for  $p_i$ )

A call to  $DF\_MColoring(j, c)$  works as follows. First, color  $c$  is assigned to  $p_j$ . If  $p_j$  has a single neighbor (its parent, which issued this call), the current procedure call terminates. Otherwise, the current invocation computes the multiset of colored tokens which includes (a)  $m$  identical tokens for each possible color, except the colors of  $p_j$  and its parent, and (b)  $(m-1)$  identical tokens with the color of  $parent_j$  (line 03). Let us notice that all the colored tokens are ordered by their color number, hence the notion of a “token with a smallest color” is well-defined. Then, for each of  $p_j$ ’s neighbor  $p_k$  (except its parent), taken one after the other (line 04), a token with a smallest color is selected for  $p_k$  (which will inherit the corresponding color) and withdrawn from the multiset  $tokens$  (hence, this token can no longer be used to associate a color to another neighbor of  $p_j$ , line 04). Due to line 03, the multiset  $tokens$  used to assign colors to  $p_j$ ’s neighbors, is such that  $|tokens| = (K-2) \times m + (m-1) = (K-1) \times m - 1$  (Property P1). (The factor  $(K-2)$  comes from the fact the colors are in the color set  $\{0, 1, \dots, (K-1)\} \setminus \{color_j, parent\_color(j)\}$ . The term  $(m-1)$  comes from the fact that  $color_i = 0$  is already used once for the neighbor  $p_i$ .) It follows that the loop is well-defined. The subtree rooted at  $p_k$  is then depth-first recursively colored (line 05).

It follows that (a) no two neighbors can be assigned the same color (line 03), (b) each process is assigned a color as small as possible (line 04), and (c) at most  $m$  neighbors of a process can be assigned the same color (lines 02 and 05).

We now show that, given the previous coloring, it is possible to assign (at least) one more color to (at least) the root  $p_i$ . The set of  $K$  colors used in Algorithm 2 is the set  $\{0, 1, \dots, (K-1)\}$ . Let us consider any vertex  $p_j$ , which is a neighbor of  $p_i$ . Due to the property stated in the theorem, we have  $K > \lfloor \frac{\Delta}{m} \rfloor + 1$ , which, due to Lemma 10 translates as  $(K-1) \times m > \Delta_j$  (P2).

As  $color_i = 0$  and  $K \geq 2$ , we have  $color_i \neq K-1$ . Moreover, we also have  $color_j \neq K-1$ . This follows from the assumption  $K > \lceil \frac{\Delta}{m} \rceil + 1$ , and the fact that, to color  $p_i$  and its neighbors, Algorithm 2



uses  $\lceil \frac{\Delta_i}{m} \rceil + 1 \leq K - 1$  colors, namely the color set  $\{0, \dots, (K - 2)\}$ .

When executing  $\text{DF\_MColoring}(j, c)$  for a neighbor  $p_j$  of  $p_i$ , Algorithm 2 executes  $(\Delta_j - 1)$  times the body of the “for” loop (lines 04-06), (once for each neighbor of  $p_j$ , except  $p_i$ , which has already been assigned a color). It follows from (P2) that  $(K - 1) \times m - 1 > \Delta_j - 1$ . Combined with (P1) we obtain  $|\text{tokens}| > \Delta_j - 1$ , from which we conclude that  $\text{tokens} \neq \emptyset$  is always true. It is consequently a loop invariant in each call related to a neighbor  $p_j$  of  $p_i$ . It follows that  $\text{tokens}$  always contains a colored token with the highest color, namely  $(K - 1)$ . This color can consequently be assigned to  $p_i$ , in addition of color 0, without violating the conflict-freedom,  $m$ -collision-freedom, and efficiency defining the CCMC problem solved by Algorithm 2. This concludes the proof of the “if” part of the theorem.

*Proof of the “only if direction”.*

In the following we use the following notations, where  $M$  is a multi-set.

- $|M|$  is the size of  $M$  (in the following all sets and multisets are finite),
- $\text{set}(M)$  is the underlying set of  $M$ , the set of elements present at least once in  $M$ ,
- $\mathbf{1}_M(x)$  is the multiplicity of an element  $x$  in  $M$ . By construction we have

$$\mathbf{1}_M(x) \geq 1 \iff x \in \text{set}(M) \quad \text{and} \quad |M| = \sum_{x \in \text{set}(M)} \mathbf{1}_M(x). \quad (6)$$

If  $A$  and  $B$  are two multisets,  $A \uplus B$  is the multiset union of  $A$  and  $B$ . In particular we have:

$$|A \uplus B| = |A| + |B| \quad \text{and} \quad \mathbf{1}_{A \uplus B}(x) = \mathbf{1}_A(x) + \mathbf{1}_B(x). \quad (7)$$

We consider a set  $S$  as a special case of a multi-set in which all elements of  $S$  have a multiplicity of 1:  $x \in S \iff \mathbf{1}_S(x) = 1$ .

Let us assume that  $\text{CCMC}(n, m, K, > 1)$  can be solved on a tree, such that at least one process, i.e.  $p_i$ , is allocated more than one color:

$$|\text{colors}_i| \geq 2. \quad (8)$$

For ease of exposition, and without loss of generality, we assume all other processes are allocated only one color:

$$\forall j \neq i : |\text{colors}_j| = 1. \quad (9)$$

Let  $C_{\text{neighbors}_i}$  denote the multiset of colored tokens allocated to the neighbors of  $p_i$ :

$$C_{\text{neighbors}_i} = \biguplus_{p_j \in \text{neighbors}_i} \text{colors}_j. \quad (10)$$

From (9) and (10) we derive (by way of (7))

$$|C_{\text{neighbors}_i}| = \left| \biguplus_{p_j \in \text{neighbors}_i} \text{colors}_j \right| = \sum_{p_j \in \text{neighbors}_i} |\text{colors}_j| = |\text{neighbors}_i| = \Delta_i. \quad (11)$$

This means that  $\Delta_i$  colored tokens are needed to color the neighbors of  $p_i$ .

Because the coloring solves  $\text{CCMC}(n, m, K, > 1)$ ,  $m$ -Collision-freedom means that

$$\forall c \in \text{set}(C_{\text{neighbors}_i}) : \mathbf{1}_{C_{\text{neighbors}_i}}(c) \leq m. \quad (12)$$

Using (12) in (6) applied to  $C_{\text{neighbors}_i}$  gives us

$$|C_{\text{neighbors}_i}| = \sum_{c \in \text{set}(C_{\text{neighbors}_i})} \mathbf{1}_{C_{\text{neighbors}_i}}(c) \leq |\text{set}(C_{\text{neighbors}_i})| \times m, \quad (13)$$

which yields, with (11) (required number of colors for  $p_i$ 's neighbors):

$$|set(C_{neighbors_i})| \geq \frac{|C_{neighbors_i}|}{m} \geq \frac{\Delta_i}{m}. \quad (14)$$

Because  $|set(C_{neighbors_i})|$  is an integer, (14) implies that

$$|set(C_{neighbors_i})| \geq \left\lceil \frac{\Delta_i}{m} \right\rceil. \quad (15)$$

Because the coloring solves  $CCMC(n, m, K, > 1)$ , it respects *Conflict-freedom*, implying that

$$set(C_{neighbors_i}) \cap colors_i = \emptyset, \quad (16)$$

and hence

$$|set(C_{neighbors_i}) \cup colors_i| = |set(C_{neighbors_i})| + |colors_i|, \quad (17)$$

$$\geq |set(C_{neighbors_i})| + 2 \quad \text{using (8),} \quad (18)$$

$$\geq \left\lceil \frac{\Delta_i}{m} \right\rceil + 2 > \left\lceil \frac{\Delta_i}{m} \right\rceil + 1. \quad \text{using (15)} \quad (19)$$

By definition  $K \geq |set(C_{neighbors_i}) \cup colors_i|$ , which yields

$$K > \left\lceil \frac{\Delta_i}{m} \right\rceil + 1 \quad (20)$$

which concludes the first part of the proof on the ‘‘only if’’ direction.

Let us now turn to the neighbors of  $p_i$ . For  $p_j \in neighbors_i$  we consider similarly to  $p_i$  the set of colored tokens allocated to  $p_j$ 's neighbors (which include  $p_i$ ):

$$C_{neighbors_j} = \biguplus_{p_k \in neighbors_j} colors_k. \quad (21)$$

Hence (as for  $p_i$ ) we have:

$$|C_{neighbors_j}| = \sum_{p_k \in neighbors_j} |colors_k|. \quad (22)$$

Contrary to  $p_i$  however, all of  $p_j$ 's neighbors do not have only one color allocated:  $p_i$  has at least two, by assumption. This yields

$$|C_{neighbors_j}| = |colors_i| + \sum_{p_k \in neighbors_j \setminus \{p_i\}} |colors_k|, \quad (23)$$

$$\geq 2 + |neighbors_j - 1| \times 1 \quad \text{using (8) and (9),} \quad (24)$$

$$\geq 2 + \Delta_j - 1 \geq \Delta_j + 1. \quad (25)$$

As for  $p_i$ ,  $m$ -Collision-freedom means that

$$|C_{neighbors_j}| \leq |set(C_{neighbors_j})| \times m. \quad (26)$$

(25) and (26) yield

$$|set(C_{neighbors_j})| \times m \geq \Delta_j + 1. \quad (27)$$

As for  $p_i$  we have (*Conflict-freedom*)

$$\text{set}(C_{\text{neighbors}_j}) \cap \text{colors}_j = \emptyset, \quad (28)$$

leading to

$$K \geq |\text{set}(C_{\text{neighbors}_j}) \cup \text{colors}_j| \quad \text{by definition,} \quad (29)$$

$$\geq |\text{set}(C_{\text{neighbors}_j})| + |\text{colors}_j| \quad \text{because of (28),} \quad (30)$$

$$\geq |\text{set}(C_{\text{neighbors}_j})| + 1 \quad \text{because of (9),} \quad (31)$$

$$K - 1 \geq |\text{set}(C_{\text{neighbors}_j})|. \quad (32)$$

Injecting (32) into (27) gives us

$$(K - 1) \times m \geq \Delta_j + 1, \quad (33)$$

$$(K - 1) \times m > \Delta_j, \quad (34)$$

$$K - 1 > \frac{\Delta_j}{m} \geq \left\lfloor \frac{\Delta_j}{m} \right\rfloor \quad \text{because of Lemma 10,} \quad (35)$$

$$K > \left\lfloor \frac{\Delta_j}{m} \right\rfloor + 1. \quad (36)$$

(36) concludes the proof of necessary condition to solve  $\text{CCMC}(n, m, K, > 1)$ .  $\square_{\text{Theorem 2}}$

## B Defining the slots of the upper layer programming level

When the root process claims termination, the other processes are in their local state 4, but cannot exploit the multi-coloring assignment. As indicated in the paragraph ‘‘Using the colors’’ (just before Section 4), to do that, they need to know the value  $K = \lceil \frac{\Delta}{m} \rceil + 1$ .

The knowledge of  $K$  can be brought to the root process by the messages  $\text{TERM}()$ , and then disseminated from the root to all the processes. Algorithm 1 is slightly modified and enriched with Algorithm 3 to allow all processes to know the value of  $K$ . Modified lines are postfixed by a ‘‘prime’’, and new lines are numbered Nxy.

Each process  $p_i$  manages a new local variable  $ak_i$  (approximate  $k$ ), initialized to  $\sigma_i = \lceil \frac{\Delta_i}{m} \rceil + 1$ , and whose final value will be  $K$ . The additional behavior of  $p_i$  is now as follows.

- First, when a non-root process informs its parent of its local termination, it now broadcasts the message  $\text{TERM}(\text{parent}_i, id_i, ak_i)$  (line 28'). Hence, the values  $ak$  of the leaves will be the first to be known by their parent.
- When a process receives a message  $\text{TERM}(\text{dest}, id, ak)$  (line 30'), in addition to its previous statements, it updates  $ak_i$  to  $\max(ak_i, ak)$  (line N1). It follows that the root learns the value of  $K$  (which is the maximal value  $ak$  it receives). When it learns it, the root progresses to the local state 5 (line 34').
- Starting from the root, when a non-leaf process is in state 5 and allowed to broadcast (predicate of line N2), it broadcasts the message  $\text{END}(id_i, ak)$  and progresses to the final state 6 (line N3).
- Finally, when a process  $p_i$ , in state 4, receives a message  $\text{END}(\text{sender}, ak)$  from its parent, it updates  $ak_i$  to  $\max(ak_i, ak)$  and progresses to state 5 (line N6). It will then forward the message  $\text{END}(\text{sender}, ak)$  to its children if it has some, and in all cases will enter local state 6 (line N3).

After the local state of a process  $p_i$  became 6, we have  $ak_i = K$ . Hence, all processes are provided with a round-based programming level in which each process  $p_i$  can  $\text{C}2m$ -freely broadcast messages at all the rounds such that  $(\text{CLOCK} \bmod ak_i) \in \text{colors}_i$ .

```

(28')    else broadcast TERM(parenti, idi, kprime); statei ← 4

(30') when TERM(dest, id, ak) is received do
(31)   if (dest ≠ idi) then discard the message (do not execute lines 31-35) end if;
(32)   coloredi ← to_coloredi ∪ {id};
(N1)   aki ← max(aki, ak);
(33)   if (coloredi = neighborsi)
(34')  then if (parenti = idi) then statei ← 5 else statei ← 3 end if
(35)   end if.

(N2) when ((CLOCK mod aki) ∈ colori) ∧ (statei = 5) do
(N3)   if (|neighborsi ≠ 1) then broadcast END(idi, ki) end if; statei ← 6.

(N4) when END(sender, k) is received do
(N5)   if (sender = parenti) ∧ (statei = 4)
(N6)   then ki ← max(ki, k); statei ← 5
(N7)   end if.

```

Algorithm 3: Obtaining the value  $K = \lceil \frac{\Delta}{m} \rceil + 1$  and informing all processes