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

Hicham Lakhlef[1], Michel Raynal[1,2], and François Taïani[1]

[1] IRISA, Université de Rennes, France
[2] Institut Universitaire de France
{hicham.lakhlef,raynal,francois.taiani}@irisa.fr

Abstract. This paper considers the broadcast/receive communication model in which message collisions and message conflicts can occur because processes share frequency bands. (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.) 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 Δ is the maximal degree of the communication graph), and an associated coloring algorithm, which is optimal with respect to K.

Keywords: Broadcast/receive, Bounded local memory, Collision-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., [22,24])[1]. 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 [1,19].

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 [10,19][2]. Other distance-1 coloring algorithms are described in several articles (e.g. [3,5,14,17]). They differ in the number of rounds they need and in the number of colors they use to implement distance-1 coloring. Let Δ be the maximal degree of the graph (the degree of a vertex is the number of its neighbors). For instance [5] presents a vertex coloring algorithm which uses $(\Delta+1)$ colors which requires $O(\Delta + \log^* n)$ rounds. An algorithm is described in [14] for trees, which uses three colors and $O(\log^* n)$ rounds. The algorithm presented in [17] requires $O(\Delta \log \Delta + \log^* n)$ rounds. These algorithms assume that the processes have distinct identities[3], 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

[1] The case where processes may exhibit faulty behaviors (such as crashes or Byzantine failures) is addressed in several books (e.g., [2,20,22,23]).

[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, $\log^* n \simeq 5$.

[3] Some initial asymmetry is necessary to solve *breaking symmetry problems* with a deterministic algorithm.

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 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 C2m-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[4]. 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.

C2m-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 C2m-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.

[4] 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.

We call *Coloring with Communication/Memory Constraints* (CCMC) the coloring problem described above. 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. "≥ 1" means that there is no constraint on the number of colors that that can be assigned to a process. 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.

CCMC is closely related to, but different from the β-frugal coloring problem, first introduced in [16]. β-frugal coloring considers a traditional distance-1 coloring in which a color is used at most β times in each node's neighborhood. A first difference lies in the number of colors assigned to each node. β-frugal coloring is a special case of CCMC, in which nodes are assigned a single color, rather than a set ($CCMC(n, \beta, K, 1)$ with our notation). A second, more fundamental difference lies in the computing model we adopt in our work, which assumes (undetected) conflicts and collisions between nodes. These conflicts and collisions complicate the task of individual nodes that produce a coloring while coordinating their communications to avoid message losses. Finally, while existing works on β-frugal coloring [16,21] have primarily focused on asymptotic bounds for graphs exhibiting sufficiently large values of Δ (e.g. $\Delta > \Delta_0 = e^{10^7} \approx 10^{4342944}$ [16]), we do not impose any such minimal bound on Δ in our work, providing results that apply to graphs actually encountered in practical systems.

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 (missing proofs can be found in [18]). Finally, Section 7 concludes the paper.

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 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.

Δ_i denotes the degree of process p_i (i.e. $|neighbors_i|$) and Δ denotes the maximal degree of the graph ($\mathsf{max}\{\Delta_1, \cdots, \Delta_n\}$). 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).

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 bcast() and receive(). A process p_i invokes bcast TAG(m) to send the message m (whose type is TAG) to its neighbors. It is assumed that a process invokes bcast() 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 bcast() 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 it 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 bcast() during the same time slot (round), a message *collision* occurs.
- If p_i and one of its neighbors invoke bcast() 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 Coloring with Communication/Memory Constraints

Definition of the CCMC problem Let $\{p_1, \cdots, p_n\}$ 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 $\{0, 1, 2, 3, 4\}$).

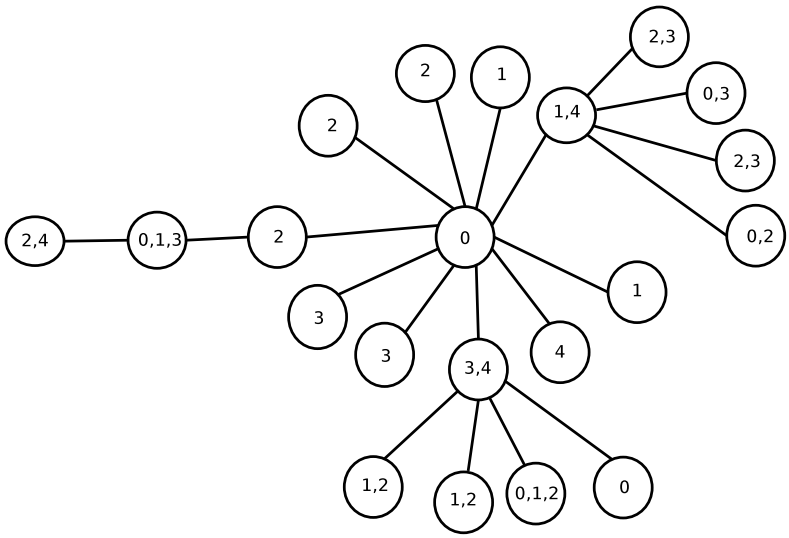

Fig. 1. Multi-coloring of a 21-process 10-degree tree with the constraint $m = 3$

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 ≤ 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

Theorem 1. *Neither* CCMC$(n, m, K, 1)$*, nor* 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 CCMC$(n, m, K, 1)$ when $K \leq \lceil \frac{\Delta}{m} \rceil$. To this end, let us consider a process p_ℓ, which has Δ neighbors (by the very definition of Δ, 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_ℓ. As p_ℓ 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 $\{p_i\} \cup neighbors_i$, which proves the theorem for CCMC$(n, m, K, \geq 1)$.

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

4.2 A necessary and sufficient condition for multicoloring

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

Theorem 2. CCMC$(n, m, \lceil \frac{\Delta}{m} \rceil + 1, > 1)$ *can be solved on a tree of maximal degree* Δ*, if and only if* (Proof in[18])

$$\exists i : \lceil \frac{\Delta}{m} \rceil + 1 > \max\Big(\{ \lceil \tfrac{\Delta_i}{m} \rceil \} \cup \Big\{ \Big\lfloor \tfrac{\Delta_j}{m} \Big\rfloor \;\Big|\; p_j \in neighbors_i \Big\} \Big) + 1.$$

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

The algorithm presented in this section use as a skeleton a parallel traversal of a tree [24]. 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 CCMC($n, m, \lceil \frac{\Delta}{m} \rceil + 1, \geq 1$) problem. It assumes that a single process initially receives an external message 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 START() causes the receiving process (say p_r) to simulate the reception of a fictitious message COLOR(), which initiates the sequential traversal.

Messages The algorithm uses two types of messages, denoted COLOR() and TERM().

- The messages 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 *sender*, *cl_map*, and *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 $id_j \in neighbor_x$, $cl\_map[id_j]$ 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 $\{0, 1, \ldots, (max\_cl - 1)\}$. Each child p_i of the message sender will use the color domain defined by $\mathsf{max}(max\_cl, \sigma_i)$ to propose colors to its own children (σ_i is defined below). Moreover, all the children of the sender will use the same slot span $\{0, 1, \ldots, (max\_cl - 1)\}$ to broadcast their messages. This ensures that their message broadcasts will be collision-free[5].
- 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.

[5] As we will see, conflicts are prevented by the message exchange pattern imposed by the algorithm.

- $state_i$ (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 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.
- $parent_i$ 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 $parent_r = id_r$.
- $colored_i$ is a set containing the identities of the neighbors of p_i that have been colored.
- to_color_i is the set of neighbors to which p_i must propagate the coloring (network traversal).
- $color\_map_i[neighbors_i \cup \{id_i\}]$ is a dictionary data structure where p_i stores the colors of its neighbors in $color\_map_i[neighbors_i]$, and its own colors in $color\_map_i[id_i]$; $colors_i$ is used as a synonym of $color\_map_i[id_i]$.
- max_cl_i 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.
- $slot\_span_i$ 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, σ_i) where $cl\_map[id_i]$ defines its color, namely, $(CLOCK + 1) \bmod \sigma_i$ (lines 2-3). Hence, at round number 1, the root will send a message COLOR() to its children (lines 26-27).

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 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 6-24. First, p_i saves the identity of its parent (the sender of the message) and its proposed color set (line 6), initializes $colored_i$ to $\{sender\}$, and to_color_i to its other neighbors (line 7). Then p_i obtains a color set proposal from the dictionary cl_map carried by the message (line 8), 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 9). Let us remind that the value max_cl_i allows it to

```
1  Init: σ_i = ⌈Δ_i/m⌉ + 1; state_i ← 0; colors_i ← ∅; colors_i stands for color_map_i[id_i]
2  when START() is received do        ▷A single process p_i receives this message.
3      p_i executes lines 5-25 as if it received the message COLOR(id_i, cl_map, σ_i)
       where cl_map[id_i] = {(CLOCK + 1) mod σ_i}

4  when COLOR(sender, cl_map, max_cl) is received do
5      if first message COLOR() received then
6          parent_i ← sender ; color_map_i[parent_i] ← cl_map[sender]
7          colored_i ← {sender} ; to_color_i ← neighbors_i \ {sender}
8          color_map_i[id_i] ← cl_map[id_i]                    ▷Synonym of colors_i
9          max_cl_i ← max(max_cl, σ_i) ; slot_span_i ← max_cl
10         if (to_color_i ≠ ∅) then          ▷next lines: tokens_i is a multiset.
11             tokens_i ← {m tokens with color x, ∀x ∈ ([0..(max_cl_i − 1)] \ colors_i)}
                          \ {1 token with color z, ∀z ∈ color_map_i[parent_i] }
12             while (|tokens_i| < |to_colors_i|) do
13                 if (|colors_i| > 1) then
14                     let cl ∈ colors_i ; suppress cl from colors_i
15                     add m tokens colored cl to tokens_i
16                 else
17                     let cl be the maximal color in color_map_i[parent_i]
18                     add one token colored cl to tokens_i
19                     color_map_i[parent_i] ← color_map_i[parent_i] \ {cl}
20             Extract |to_colors_i| non-empty non-intersecting multisets tk[id]
               (where id ∈ to_color_i) from tokens_i such that no tk[id] contains
               several tokens with the same color
21             foreach id ∈ to_color_i do
22                 color_map_i[id] ← {colors of the tokens in tk[id]}
23             state_i ← 1                                      ▷p_i has children
24         else state_i ← 3                                     ▷p_i is a leaf
25     else color_map_i[id_i] ← color_map_i[id_i] ∩ cl_map[id_i]

26 when ((CLOCK mod slot_span_i) ∈ colors_i) ∧ (state_i ∈ {1, 3})) do
27     case (state_i = 1) bcast COLOR(id_i, color_map_i, max_cl_i) ; state_i ← 2
28     case (state_i = 3) bcast TERM(parent_i, id_i); state_i ← 4 ▷p_i 's subtree is done

29 when TERM(dest, id) is received do
30     if (dest ≠ id_i) then discard the message (do not execute lines 31-34)
31     colored_i ← colored_i ∪ {id}
32     if colored_i = neighbors_i then
33         if parent_i = id_i then the root p_i claims termination
34         else state_i ← 3
```

Algorithm 1: C2m-free algorithm solving CCMC$(n, m, \lceil \frac{\Delta}{m} \rceil + 1, \geq 1)$ in tree networks (code for p_i)

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 24).

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 $\{0, 1, \ldots, (max\_cl_i - 1)\}$, and considers that each of these colors c is represented by m tokens colored c. Then, it computes the multiset[6], denoted $tokens_i$, containing all the colored tokens it can use to build a color set proposal for each of its children (line 11). 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 sets of colors to its children (lines 20-22). 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 14-15), and re-enters the "while" loop (line 12). 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 16-19). 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 11.

As already said, when the predicate $|tokens_i| < |to\_color_i|$ (line 12) becomes false, $tokens_i$ contains enough colored tokens to assign to p_i's children. This assignment is done at lines 20-22. 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 20 means that, if each of z multisets $tk[id_x]$ contains a token with the same color, this token appears at least z times in $tokens_i$.

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 25). Hence, when p_i has

[6] 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.

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 8, 14, and 25), 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 11-23). 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 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).

Algorithm: reception of a message TERM() When a process p_i receives such a message it discards it if it is not the intended destination process (line 30). If the message is for it, p_i adds the sender identity to the set $colored_i$ (line 31). Finally, if $colored_i = neighbors_i$, p_i learns that the subtree rooted at it is colored (line 32). It follows that, if p_i is the root ($parent_i = i$), 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 CCMC($n, m, K, 1$). When a process enters state 3 (at line 24 or line 34), it reduces $color\_map_i[id_i]$ (i.e., $colors_i$) to obtain a singleton.

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

The proof assumes $n > 1$. Let us remember that $colors_i$ and $color\_map_i[id_i]$ are the same local variable of p_i, and p_r denotes the dynamically defined root.

Cost of the algorithm Each non-leaf process broadcasts one message COLOR(), and each non-root process broadcasts one message TERM(). Let x be the number of leaves. There are consequently $(2n - (x + 1))$ broadcasts. As $\Delta \leq x + 1$, 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 COLOR() and TERM(). These colors span the interval $[0..\lceil \frac{\Delta_r}{m} \rceil]$, which means that the broadcasts of messages 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 COLOR() requires at most $O(d\lceil \frac{\Delta}{m} \rceil)$ rounds. As the same occurs for the messages TERM(), returning from the leaves to the root, it follows that the time complexity is $O(d\lceil \frac{\Delta}{m} \rceil)$.

Proof of the algorithm

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.

The proof is decomposed into nine lemmas showing that the algorithm (a) is itself conflict-free and collision-free, (b) terminates, and (c) associates with each process p_i a non-empty set $colors_i$ satisfying the Conflict-freedom, m-Collision-freedom and Efficiency properties defined in Section 3.

To this end, a notion of *well-formedness* suited to COLOR() messages is introduced. More precisely 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 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 $\forall\ id \in (neighbors_i \cup \{id_i\}) :\ cl\_map[id] \subseteq [0..max\_cl - 1]$.

Due to page limitation, the missing lemmas are proved in [18]. Here we only give the proof of one them.

Lemma 1. *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 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 COLOR$(-, cl_map, -)$ received by the root satisfies this property. Hence, any process that receives a message 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 25 when p_i receives a message COLOR() from one of its children. If p_i is a leaf, it has no children, and consequently never executes line 25. So, let us assume that p_i is not a leaf and receives a message COLOR$(id_j, cl_map, -)$ from one of its children p_j. In this case p_i previously broadcast at line 27 a message COLOR$(id_i, color_map_i, -)$ that was received by p_j and this message is well-formed (this is proved in another lemma).

A color c that is suppressed at line 25 when p_i processes 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 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 6) when p_j received the well-formed message COLOR() from p_i. When processing COLOR() received from p_i, process p_j can suppress colors from $color\_map_j[id_i]$ only at line 19, where it suppresses colors starting from the greatest remaining color. We have the following.

– If p_i is not the root, the message COLOR() it received was well-formed (this is proved in another lemma). In this case, it follows from a lemma proved in [18] 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" 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 11) we have
$|tokens_j| = m \times \mathsf{max}(\sigma_r, \sigma_j) - m = m\lceil \frac{\mathsf{max}(\Delta_r, \Delta_j)}{m} \rceil \geq \mathsf{max}(\Delta_r, \Delta_j) \geq \Delta_j = |to\_colors_j|$,
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 25, p_i never withdraws its smallest color from the set $color\_map_i[id_i]$. □

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 Δ 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

$$\forall i, \forall c : |\{j : (id_j \in neighbors_i \cup \{id_i\}) \wedge (c \in colors_j)\}| \leq m, \tag{1}$$

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 (1) 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 (1). More precisely, assuming a coloring $\mathsf{col} : V \mapsto [0..(K * m) - 1]$ 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)

$$\begin{array}{l} \mathsf{col}' : V \mapsto [0 \,..\, K-1] \\ \qquad x \to \mathsf{col}(x) \ \mathsf{mod}\ K, \end{array} \tag{2}$$

fulfills (1) on G ([7]). 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 (1) 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 a heterogeneous network, lines 20-22 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.

Acknowledgments

This work has been partially supported by the Franco-German DFG-ANR Project 40300781 DISCMAT (devoted to connections between mathematics and distributed computing), and the Franco-Hong Kong ANR-RGC Joint Research Programme 12-IS02-004-02 CO2Dim.

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)
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)

[7] 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 $c_1, ..., c_x \in [0..(K * m) - 1]$ (hence $x \leq m$), such that $(c_1 \bmod K) = \cdots = (c_x \bmod K) = c \in [0..(K - 1)]$.

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. Hugh Hind H., Molloy M., and Reed B., Colouring a graph frugally. *Combinatorica*, 17(4):469–482, 1997.
17. 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)
18. Lakhlef H., Raynal M., and Taïani F., Vertex coloring with communication and local memory constraints in synchronous broadcast networks. *Tech Report 2035*, 24 pages, IRISA, Université de Rennes (F) https://hal.inria.fr/hal-01300095 (2016)
19. Linial N., Locality in distributed graph algorithms. *SIAM Journal on Computing*, 21(1):193-201 (1992)
20. Lynch N.A., *Distributed algorithms*. Morgan Kaufmann, 872 pages (1996)
21. Molloy M. and Bruce Reed B., Asymptotically optimal frugal colouring. *Journal of Combinatorial Theory, Series B*, 100(2):226–246 (2010), Corrigendum in *Journal of Combinatorial Theory, Series B*, 100(2):247–249 (2010)
22. Peleg D., *Distributed computing, a locally sensitive approach*. SIAM Monographs on Discrete Mathematics and Applications, 343 pages, ISBN 0-89871-464-8 (2000)
23. Raynal M., *Fault-tolerant agreement in synchronous message-passing systems*. Morgan & Claypool Publishers, 165 pages, (ISBN 978-1-60845-525-6) (2010)
24. Raynal M., *Distributed algorithms for message-passing systems*. Springer, 500 pages, ISBN 978-3-642-38122-5 (2013)