Efficient self-stabilizing construction of disjoint MDSs in distance-2 model

Colette Johnen, Mohammed Haddad

▶ **To cite this version:**

Colette Johnen, Mohammed Haddad. Efficient self-stabilizing construction of disjoint MDSs in distance-2 model. [Research Report] Inria Paris, Sorbonne Université; LaBRI, CNRS UMR 5800; LIRIS UMR CNRS 5205. 2021. hal-03138979

HAL Id: hal-03138979
https://hal.science/hal-03138979

Submitted on 11 Feb 2021

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.

Efficient self-stabilizing construction of disjoint MDSs in distance-2 model ⋆

Colette Johnen[1] and Mohammed Haddad[2]

[1] INRIA Paris, LIP6, UMR 7606, France
Université de Bordeaux, LaBRI, UMR 5800, France
johnen@labri.fr
[2] Université Claude Bernard Lyon, LIRIS, UMR 5205, France
mohammed.haddad@univ-lyon1.fr

Abstract. We study the deterministic silent self-stabilizing construction of two disjoint minimal dominating sets (MDSs) in anonymous networks. We focus on algorithms where nodes share only their status (i.e. the name of their MDS to which they belong, if they belong to a MDS). We prove that such an algorithm cannot be designed in distance-1 model under a central daemon; therefore, we study this problem in the distance-2 model under a central daemon.
We present an algorithm building two disjoint minimal dominating sets such that one of them is also a maximal independent set (MIS). Any execution of this algorithm converges in $5n$ moves. Our approach to compute this value is novel: the number of moves is not computed per node.
We propose a second algorithm faster than the first one at the expense of the independence property of one of the constructed sets. A node executes at most 2 moves.
If the network is not anonymous, the presented algorithms can be translated into a silent self-stabilizing algorithms converging in $O(n \cdot m)$ moves in the distance-1 model under the distributed daemon where m is the number of edges and n the number of nodes. This improves the complexity of $O(n \cdot m)$ moves of proposed algorithms with the same assumptions.

Keywords: algorithmic graph · distributed algorithm · self-stabilization · minimal dominating set · maximal independent set · distance-2 model · convergence time

1 Introduction

The distance-2 model or the expression model permits a higher-level description of a self-stabilizing distributed algorithm. Some authors prefer to use the distance-2 model [3,2,4,5] or the expression model [10,11,12,1] to design self-stabilizing algorithms for solving graph problems. In the distance-2 model, a

⋆ This study was partially supported by ANR project ESTATE : ANR-16 CE25-0009-03

node v can access to the state information of neighbors at distance 2 from it. In the expression model, a node v does not have access to the states of nodes at distance two, but has access only to aggregates of these states through the *expressions* of its neighbors. An *expression* on a node u is a value computed according to u and u's neighbors states. Nevertheless, the distance-2 model and the expression model are similar in terms of computation power (*i.e.* the translation of an algorithm in the expression model to the distance-2 model has no overhead in terms of moves).

As shown in [3] an algorithm designed for the distance-2 model or the expression model under the central daemon (i.e. at each step only a single node executes an action) can be translated into another algorithm for distance-1 model with distributed daemon (i.e. at each step one or several nodes execute an action) assuming a non-anonymous network. The slowdown is $O(m)$ per move in the expression model [10] where m is the number of edges.

In [6], deterministic self-stabilizing algorithms building two disjoint MDSs in the distance-2 model under the central daemon are presented, they converge in $O(n^2)$ moves. Their translations give self-stabilizing algorithms in the distance-1 model under the unfair distributed daemon converging in $(n^2 \cdot m)$ moves.
In [7,9] self-stabilizing distributed algorithm for finding two disjoint dominating sets are presented in distance-1 model where nodes have identifiers. The algorithm presented in [7] ensures safe convergence. The number of moves during any execution of algorithm in [7] is not computed. In [9] a central daemon is assumed, this algorithm converges in $O(n^3)$ moves where n is the number of nodes.

Contribution We study the deterministic self-stabilizing construction of two disjoint minimal dominating sets (MDSs) in anonymous networks. We focus on algorithms where nodes share only their status (i.e. the name of their MDS to which they belong, if they belong to a MDS).
In Section 3, we prove that such an algorithm cannot be designed in distance-1 model even under a central daemon.
In Section 4, we present an algorithm building two disjoint sets: a maximal independent set and a minimal dominating set. Any execution of that algorithm converges in $5n$ moves. Our approach to calculating the number of moves is new: the limit is not established on the number of moves of each node but on the number of executions of each rule. This algorithm improves the convergence time by a factor n of existing algorithms solving the same problem (self-stabilizing constructions of disjoint MIS and MDS) in distance-2 model.
In Section 6, we present an algorithm building two disjoint MDSs. During any execution, a node executes at most 2 moves. This algorithm is faster than the first one at the expense of the independence property of one of the constructed sets.
Using the transformer of [10], algorithms converging in $O(n.m)$ moves under the unfair distributed daemon in distance-1 model are obtained from the presented algorithms. The transformed algorithms are the first ones building two

disjoint MDS to converge in $O(n.m)$ moves under the unfair distributed daemon in distance-1 model.

2 Model and Concepts

A distributed system S is an undirected graph $G = (V, E)$ where the vertex set, V, is the set of nodes and the edge set, E, is the set of communication links. We have $|V| = n$ and $|E| = m$. A link $(u, v) \in E$ if and only if u and v can directly communicate (links are bidirectional); so, the nodes u and v are neighbors. $N(v)$ denotes the set of v's neighbors: $N(v) = \{u \in V \mid (u, v) \in E\}$. $N[v]$ denotes the closed neighborhood of v (i.e. the set of nodes at distance less than 2 from v): $N[v] = N(v) \cup \{v\}$.

Dominating set A subset D of V is dominating if any node of $V - D$ is neighbor of a node of D. A dominating set D is *minimal* if any subset of D is not a dominating set of V.

Independent set A subset of V, I, is independent if its does not contain two neighboring nodes. A independent set I is maximal (denoted MIS) if the adding of any node to I would falsify the independence property. A maximal independent set is also a minimal dominating set.
Ore [8] observed that any graph without isolated nodes always contains a pair of disjoint dominating sets. A corollary of Ore's observation is that in any graph without isolated nodes, two disjoint minimal dominating sets exist.
For the rest of this paper, we assume that there are not any isolated nodes.

Deterministic Algorithm Each node maintains a set of variables. A node can modify only the values of its own variables. The *state* of a node is defined by the values of its variables. The Cartesian product of states of all nodes determines the *configuration* of the system. The *program* of each node is a set of *deterministic rules*. Each rule has the form: $Rule_i :< Guard_i > \longrightarrow < Action_i >$. The *guard* of a v's rule is a Boolean expression. If $Guard_i$ is satisfied by v then $Rule_i$ is enabled by v; v is said to be enabled. A *move* by a node v is the execution of an enabled rule by v (i.e. the updating of v' variables as it is defined in the *deterministic action* part of the executed rule).
A *computation* e is a sequence of configurations $e = c_0, c_1, \ldots, c_i, \ldots$, where c_{i+1} is reached from c_i by a computation step, $\forall i \geqslant 0$. A computation e is *maximal* if it is infinite, or if it reaches a terminal configuration. A configuration is *terminal*, if and only if no node is enabled.
Anonymous network. In an anonymous network, nodes do not have identifiers or node identifiers are not used by the algorithm; so all nodes execute the same algorithm (i.e. set of rules). In a non-anonymous network, every node v in the network has an identifier, denoted by id_v. It is possible to order the identifier values.
Distance-2 versus distance-1 model. In the distance-1 model, the *guard* of a rule on the node v is a Boolean expression involving the state of the nodes

in the closed neighborhood of v (i.e. states of nodes in $N[v]$). In the distance-2 model, the *guard* of a rule on the node v is a Boolean expression involving the state of the nodes in the closed neighborhood of any node of $N[v]$ (i.e. states of nodes in $N[u]$ with u being any node of $N[v]$).

Central daemon versus distributed daemon. Under the central daemon, during a *computation step*, a single enabled node executes a move (it is chosen without any fairness constraint). Under the distributed daemon, during a computation step, one or several enabled nodes are chosen without any fairness constraint to execute simultaneously a move.

Silent Deterministic Self-Stabilization. Let $\mathcal{L}$ be a predicate on the configurations. A distributed system S is a silent deterministic self-stabilizing system to $\mathcal{L}$ if and only if (1) all terminal configurations satisfy $\mathcal{L}$; (2) all computations reach a terminal configuration.

Stabilization time of a deterministic algorithm. The stabilization time of a deterministic algorithm may be estimated in term of the total number of moves required in the worst case to reach a terminal configuration. Under a central daemon, the number of moves is the number of computation steps; but under a distributed daemon, the number of moves may be as large as n times the number of steps.

Closure of Predicate. Let $\mathcal{P}$ be a predicate on the configurations. Let $\mathcal{L}_{\mathcal{P}}$ be the set of configurations satisfying $\mathcal{P}$. $\mathcal{L}_{\mathcal{P}}$ is closed, if and only if any step from a configuration of $\mathcal{L}_{\mathcal{P}}$ reaches a configuration of $\mathcal{L}_{\mathcal{P}}$.

3 Complexity of Self-stabilizing construction of two disjoint MDSs

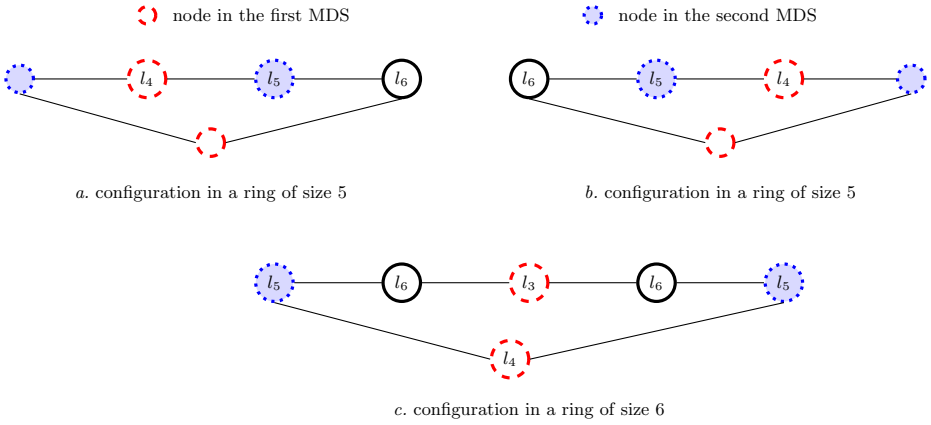
In this section, the complexity of a self-stabilizing algorithm building two disjoint minimal dominating sets is studied. The status of a node has three possible values: either it belongs to the first minimal dominating set, to the second one or it does not belong to any minimal dominating set.

Theorem 1. *In the distance 1-model, there is no deterministic silent self- stabilizing construction of two disjoint MDSs converging under the central daemon, if nodes share only their status.*

Proof. Let us study the 3 following local configurations on node v:

1. no node of $N[v]$ belongs to a MDS (local configuration l_1).
2. no node of $N(v)$ belongs to a MDS and v belongs to the second MDS (local configuration l_2).
3. no node of $N(v)$ belongs to a MDS and v belongs to the first MDS (local configuration l_3).

Assume that a node in the local configuration l_1, l_2, or l_3 is enabled. Let us study the algorithm on a chain of 3 nodes ($v1$, $v2$, and $v3$). From a configuration where $v1$ and $v3$ are not in a MDS, there is an infinite execution where $v2$ is the only node to execute a rule at each step. So no one can design a silent self-stabilizing



a. configuration in a ring of size 5

b. configuration in a ring of size 5

c. configuration in a ring of size 6

Fig. 1. 3 configurations

algorithm where a node in the local configuration l_1, l_2, or l_3 is enabled. It is easy to see that any self-stabilizing algorithm would have to ensure that a node v in the local configuration l_1 is enabled. So either a node is disabled in the local configuration l_3 or in the local configuration l_2.

Without any loss of generality, we study in the following the algorithms where a node in the local configuration l_3 is disabled.

The label of a node in a configuration of the Figure 1 is the name of its local configuration. The configuration in Figure 1.*a* and the configuration in Figure 1.*b* have two disjoint MDSs : these configurations are the only legitimate ones up to a rotation in a ring of size 5. So in at least one of these 2 configurations, no node is enabled as we study silent algorithms. In the both configurations, there is a node having one of the following local configurations:

- a node in the first MDS having its two neighbors in the second MDS (local configuration l_4)
- a node in the second MDS having a single neighbor in the first MDS (local configuration l_5).
- a node not belonging to any MDS has a neighbor in the first MDS and the other one in the second MDS (local configuration l_6).

The local configuration of any node in the configuration Figure 1.*c* is l_i with $i \in \{3...6\}$. We conclude that all nodes are disabled in this configuration. So this configuration is terminal for any deterministic silent self-stabilizing construction of two disjoint MDSs where nodes share only their status.
In the configuration Figure 1.*c*, the ring is proper colored (the coloration is greedy); but there is not two MDSs. Thus this configuration is not legitimate, so we have established the impossibility result. □

4 Self-stabilizing algorithm of disjoint MIS and MDS

The algorithm 1 builds a MIS (the set $RedSet$) and a MDS (the set $BlueSet$). A node v belongs to $RedSet$ (resp. $BlueSet$) if $color(v) = red$ (resp. $color(v) = blue$). The membership to one of this set is determined by the value of a single shared variable $color$. So, the intersection of $RedSet$ and $BlueSet$ is empty.

If v is not dominated by a red node then v can take the color red ($notRedDom(v)$ is verified). If a red node v is dominated by another red node then it can quit $RedSet$ ($beNotRed(v)$ is verified).

The construction of $RedSet$ is priority to the construction of $BlueSet$. A node v can join of quit the set $BlueSet$ only when the red domination over itself is permanent (i.e. $redOk(v)$ is verified). When v is dominated by a node u verifying the predicate $isolatedRed$. The node u is a red node having not red neighbor; so u will stay in $RedSet$ forever.

If every node of $N[v]$ is dominated by several $blue$ nodes then v, a $blue$ node verifying $redOk(v)$, may quit $BlueSet$ - $beNotBlue(v)$ is verified.

If v verifying $redOk(v)$ is not dominated by a $blue$ node then v can take the color $blue$ ($beBlue2(v)$ is verified). If a neighbor of v verifying $isolatedRed$ is not dominated by a $blue$ node then v can take the color $blue$ ($beBlue1(v)$ is verified).

To compute the value of $redOk(v)$, $beBlue1(v)$, and $beNotBlue(v)$ the node v has to know the color of nodes at distance-2 of itself.

The Figure 2 illustrates the algorithm execution. During the first step, the node x quits the $BlueSet$ (executing $RBlueOut$) because all nodes of $N[x]$ has several $blue$ neighbors. During the second step, the node v executes $RRedOut$ because it has a red neighbor. During the third step, w takes the red color (i.e. it executes $RRedIn$); now $RedSet$ is a maximal independent set (configuration d). In the sequel of the execution, no red move will be executed ($RedSet$ will stay unchanged). In the configuration d, $beBlue1(x)$ is verified: w verifying $isolatedRed(w)$ has not $blue$ neighbor. So the node x takes the color $blue$ by executing $RBlueIn$: to give to w a $blue$ neighbor. In the configuration e, $beBlue2(v)$ is verified; so, the node v takes also the $blue$ color. In the configuration f, $BlueSet$ is a dominating set not minimal, $beNotBlue(x)$ and $beNotBlue(z)$ are verified. After the execution of $RBlueOut$ by x a terminal configuration is reached. $BlueSet$ is a minimal dominating set (configuration g).

5 Proof of correctness of algorithm 1

Lemma 1. *In a terminal configuration c, RedSet is a maximal independent set, and BlueSet is a minimal dominating set*

Proof. Let c be a terminal configuration.
In c, no node verifies the predicate $notRedDom$; so a node that does not have the color red has a neighbor having the color red. In c, no node verifies the predicate

Algorithm 1: self-stabilizing construction of disjoint MIS and MDS

Shared variable of v

- $color(v) \in \{red, blue, \perp\}$

Macro on v

- $\#Blue(v)$ is $|\{w \in N[v] \mid color(w) = blue\}|$
- $myRedNgb(v)$ is the nodes set defined as $\{u \in N(v) \mid isolatedRed(u)\}$

Predicates in v

- $isolated(v) \equiv \forall w \in N(v)$ we have $color(w) \neq red$
- $notRedDom(v) \equiv color(v) \neq red \wedge isolated(v)$
- $beNotRed(v) \equiv color(v) = red \wedge \neg isolated(v)$
- $isolatedRed(v) \equiv color(v) = red \wedge isolated(v)$
- $redOk(v) \equiv \exists u \in N[v]$ verifying $isolatedRed(u)$
- $beBlue1(v) \equiv color(v) = \perp \wedge \exists u \in myRedNgb(v)$ verifying $\#Blue(u) = 0$
- $beBlue2(v) \equiv color(v) = \perp \wedge \#Blue(v) = 0$
- $beNotBlue(v) \equiv color(v) = blue \wedge \forall u \in N[v]$ we have $\#Blue(u) > 1$

Rules on v

RRedIn(v) : $notRedDom(v) \longrightarrow color(v) := red$;

RRedOut(v) : $beNotRed(v) \longrightarrow color(v) := \perp$;

RBlueIn(v) : $redOk(v) \wedge (beBlue1(v) \vee beBlue2(v)) \longrightarrow color(v) := blue$;

RBlueOut(v) : $redOk(v) \wedge beNotBlue(v) \longrightarrow color(v) := \perp$;

beNotRed; so all neighbors of a *red* node have not the color *red*. We conclude that in c, $redOk(v)$ is verified by any node and $(isolated(v) \iff color(v) = red)$.

Assume that a node v verifies $\#Blue(v) = 0$, in c. By hypothesis, v's color is not *blue*. v has the color *red* otherwise v could executed the rule $RBlueIn$. The node v has a least a neighbor u (there is no isolated node) : $v \in myRedNgb(u)$. The color u is $\perp$ by hypothesis and because $isolatedRed(v)$ is verified in c. So, u verifies the predicate $beBlue1(u)$: u is enabled in c. There is a contradiction. We conclude that in a terminal configuration, $BlueSet$ is a dominating set.

In c no node verifies the predicate $beNotBlue$; so $BlueSet$ is a minimal dominating set. □

5.1 Convergence of algorithm 1

First, we compute the number of *red* moves along any execution; then the number of *blue* moves. The execution of the rule $RBlueIn$, or $RBlueOut$ (resp. $RRedIn$ or $RRedOut$) by a node is a *blue* (resp. *red*) move.

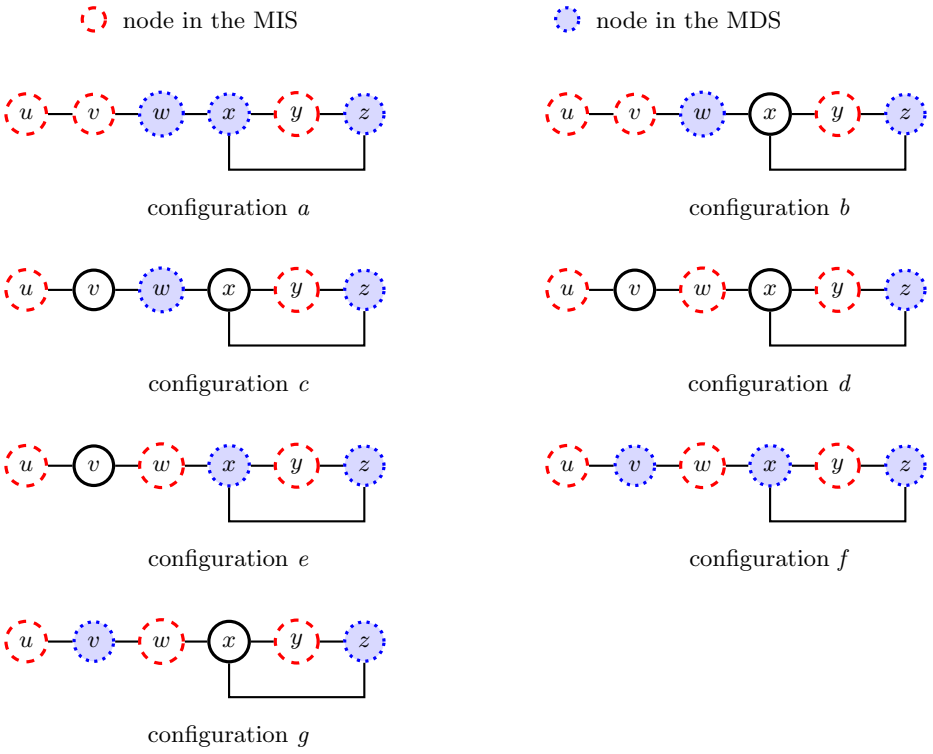


Fig. 2. an execution of algorithm 1 reaching a terminal configuration

Number of *Red* moves. We prove that a node executes at most one time the rule *RRedIn* - Lemma 3. Then, we prove a node executes at most one time the rule *RRedOut* - Lemma 4.

Lemma 2. *The predicate* $isolatedRed(v)$ *is closed.*

Proof. Let $c1$ be a configuration where $isolatedRed(v) = true$. Let cs be a computation step from $c1$ reaching the configuration $c2$. The node v is disabled in $c1$; so v has the *red* color in $c2$.
Let u be a node of $N(v)$. u verifies $\neg notRedDom(u)$ in $c1$. So, the node u cannot take the *red* color during cs. We conclude that in $c2$, $isolatedRed(v)$ is still verified. □

Corollary 1. *The predicate* $redOk(v)$ *is closed.*

On a node verifying the predicate $redOk(v)$, the rule *RRedIn* is disabled.

Lemma 3. *A node* v *is disabled forever after executing a RRedIn move.*

Proof. Assume that from the configuration c, v executes the rule *RRedIn* to reach the configuration c'. In c, none v's neighbor has the color *red*.

As v is the single node to execute an action during this computation step; in c', we have $isolatedRed(v)$. A node verifying $isolatedRed$ is disabled.
According to Lemma 2, $isolatedRed(v)$ stays verified along any execution. So, the node v is and stay disabled along any execution after its $RRedIn$ move. □

Lemma 4. *Along any execution, a node executes at most one time a RRedOut move.*

Proof. Between two consecutive $RRedOut$ moves by a node v, this node executes at least one time the rule $RRedIn$. After a $RRedIn$ move, a node v is disabled forever (Lemma 3). So a node v cannot execute severals $RRedOut$ moves. □

Number of *Blue* moves. We compute the number of *blue* moves along any execution. Notice that the execution of $RBlueIn$ (or $RBlueOut$) may be not the last move of a node as show in the execution of the Figure 2. We does not compute the number of *blue* moves done by a node but the global number of *blue* moves during an execution. We prove that after a computation step where a node execute $RBlueIn$ a new node verifies the following predicate $blueOk(u) \wedge \#Blue(u) \geq 1$ (Lemma 7). This predicate is closed (Lemma 6).

A *blue* node verifying $BlueStable(v)$ will never take the red color.

Definition 1. $BlueStable(v) \equiv (color(v) \neq blue \ \vee \ redOk(v))$

Lemma 5. *The predicate $BlueStable(v)$ is closed.*

Proof. Let $c1$ be a configuration where $BlueStable(v)$ is verified.
Assume that in $c1$, $color(v) \neq blue$. Till v does not execute the rule $RBlueIn$ $BlueStable(v)$ is verified (as $color(v) \neq blue$). Let cs be a step from c to c' where the node v executes a $RBlueIn$ move. The predicate $redOk(v)$ is verified in c according to the guard rule.
If $redOk(v)$ is verified in c then $BlueStable(v)$ is verified along any execution from c because $redOk(v)$ is closed (Corollary 1). □

Corollary 2. $blueOk(v) \equiv (\forall u \in N[v]$ *we have* $BlueStable(u))$. *The predicate $blueOk(v)$ is closed.*

If a node v verifying $blueOk$ has a $Blue$ neighbor then along any execution it will have at least a $Blue$ neighbor (because v's *blue* neighbor can only execute the rule $RBlueOut$).

Lemma 6. *On node v, the predicate $blueOk(v) \wedge \#Blue(v) \geq 1$ is closed.*

Proof. The predicate $blueOk(v)$ is closed (Corollary 2).
Let $c1$ be a configuration where $blueOk(v) \wedge \#Blue(v) = 1$. Let us name u, the single node of $N[v]$ having the color *blue* in $c1$. The node u verifies $\neg notRedDom(u)$ in $c1$ because $redOk(u)$ is verified.

The node u verifies $\neg beNotBlue(u)$ in $c1$ because $\#Blue(v) = 1$. So, the node u is disabled in $c1$. We conclude that in the reached configuration after any computation step from $c1$, we have $\#Blue(v) \geq 1$.
Let $c2$ be a configuration where $\#Blue(v) > 1$. We denotes by nb the value of $\#Blue(v)$ in $c2$. During any computation step from $c2$, a single node of $N[v]$ executes a move (because a central daemon is assumed). So, in the reached configuration, we have $\#Blue(v) \in [nb - 1, nb + 1]$.
We conclude that $\#Blue(v) \geq 1$ in $c2$. □

Lemma 7. *Let $c \rightarrow c'$ be a computation step where a node v executes a $RBlueIn$ move. There is at least a node u such that (1) the predicate $blueOk(u) \wedge \#Blue(u) \geq 1$ is not verified in c but (2) it is verified in c'.*

Proof. Let cs be a step from $c1$ to $c2$ where a node v executes a $RBlueIn$ move. Assume that $beBlue2(v)$ is verified in c. We have $\#Blue(v) = 0$ in c. The predicate $blueOk(v)$ is verified in c. The predicate $blueOk(v)$ is closed (Corollary 2). As v is the single node to execute an action during cs; $blueOk(v) \wedge \#Blue(v) = 1$ is verified, in c' but not in c.
Assume that $beBlue1(v)$ is verified in c. In c, v has a neighbor u verifiying $\#Blue(u) = 0$ and $isolatedRed(u)$. The predicate $blueOk(u)$ is verified in c because $(isolatedRed(u) \Rightarrow blueOk(u))$. As v is the single node to execute an action during cs; $blueOk(u) \wedge \#Blue(u) = 1$ is verified, in c' but not in c. □

Theorem 2. *Along any execution, there is at most n $RBlueIn$ moves and $2n$ $RBlueOut$ moves.*

Proof. After a $RBlueIn$ move a new node u verifies the predicate $blueOk(u) \wedge \#Blue(u) \geq 1$ (Lemma 7). As this predicate is closed (Lemma 6), we conclude that any execution has at most n $RBlueIn$ moves.
On a node, the number of $RBlueOut$ moves is at most the number of $RBlueIn$ moves plus one more. We conclude that any execution has at most $2n$ $RBlueIn$ moves. □

6 Self-stabilizing algorithm building two disjoint MDSs

The algorithm 2 builds the sets $RedSet$ and $BlueSet$. These set are two disjoint MDSs. The algorithm 2 is a self-stabilizing algorithm in the distance-2 model under a central daemon; the two disjoint MDSs are built in less than $2n$ moves

A node v belongs to $RedSet$ (resp. $BlueSet$) if and only if $color(v) = red$ (resp. $color(v) = blue$). As, the membership to one of these sets is determined by the value of a single variable shared: $color$; the intersection of $RedSet$ and $BlueSet$ is empty. Moreover, macros $\#Red(u)$ and $\#Blue(u)$ may also be considered as expressions in the expression model.

The construction of $RedSet$ has priority over the construction of $BlueSet$. If a node v has not a closed neighbor belonging to $RedSet$ (i.e. the predicate

$notRedDom(v)$ is satisfied) then v can join the $RedSet$ (i.e. the guard of the rule $RRedIn$ is satisfied). If every nodes of closed neighborhood of v has several nodes belonging to $RedSet$ and v belongs to $RedSet$ (i.e. the predicate $beNotRed(v)$ is satisfied) then v can quit $RedSet$ (i.e. the guard of the rule $RRedOut$ or the guard the rule $RBlueIn$ is satisfied).

Algorithm 2: self-stabilizing construction of two disjoint MDSs

Variable of v shared

- $color(v) \in \{red, blue, \perp\}$

Macros on v

- $\#Red(u)$ is $|\{w \in N[u] \mid color(w) = red\}|$
- $\#Blue(u)$ is $|\{w \in N[u] \mid color(w) = blue\}|$

Predicates on v

- $notRedDom(v) \equiv (\#Red(v) = 0)$
- $beNotRed(v) \equiv (\forall u \in N[v] \mid (\#Red(u) > 1)) \wedge (color(v) = red)$
- $beNotBlue(v) \equiv (\forall u \in N[v] \mid (\#Blue(u) > 1)) \wedge (color(v) = blue)$
- $redOk(v) \equiv (\forall u \in N[v] \mid (\#Red(u) \geq 1))$
- $blueOk(v) \equiv$
  $(\forall u \in N(v) \mid (color(u) \neq red) \vee (\#Blue(u) \geq 1)) \wedge (\#Blue(v) \geq 1)$
- $canBeBlue(v) \equiv redOk(v) \wedge (color(v) = \perp)$

Rules on v

RRedIn : $notRedDom(v) \longrightarrow color(v) := red$;

RRedOut : $beNotRed(v) \wedge blueOk(v) \longrightarrow color(v) := \perp$;

RBlueIn : $(beNotRed(v) \vee canBeBlue(v)) \wedge \neg blueOk(v) \longrightarrow color(v) := blue$;

RBlueOut : $redOk(v) \wedge beNotBlue(v) \longrightarrow color(v) := \perp$;

If a node v has not a closed neighbor belonging to $BlueSet$ then the predicate $\neg blueOk(v)$ is satisfied. If every nodes of closed neighborhood of v has several nodes belonging to $BlueSet$ and v belongs to $BlueSet$ then the predicate $beNotBlue(v)$ is satisfied. The verification of $\neg blueOk(v)$ (resp. $beNotBlue(v)$) is not enough to allow the node$v$ to quit $BlueSet$ (resp. to join it).
A node v can join or quit the set $BlueSet$ (i.e. executing the rule $RBlueIn$ or the rule $RBlueOut$) only when the red domination in its neighborhood is certain; more precisely, when every node in its neighborhood is dominated by at least node of $RedSet$ (i.e. the predicate $redOk(v)$ is satisfied).
Notice that the condition to join $RedSet$ is not similar to the condition to join $BlueSet$. If a red neighbor of node v, u is not dominated by a $blue$ node (i.e. the following predicate is verified $(color(u) = red) \wedge (\#Blue(u) = 0)$ then

$\neg blueOk(v)$ is verified. So the node v may join $BlueSet$ even if v is dominated by a *blue* node; in the case where one of its *red* neighbor is not dominated by a *blue* node.

The Figure 3 illustrates the algorithm execution. During the first step, the node u executes the rule $RRedIn$. Now, $RedSet$ is a minimal dominating set; hence no *red* move will be executed during the end of the execution. In the configuration b, $\#Blue(v) = 0$ and $\#Blue(z) = 0$; so, the node v joins $BlueSet$, then the node z does the same move. In the configuration d, $\#Blue(x) = 0$ and $color(x) = red$ so $\neg blueOk(z)$ is verified, during the last step, w executes the rule $RBlueIn$.

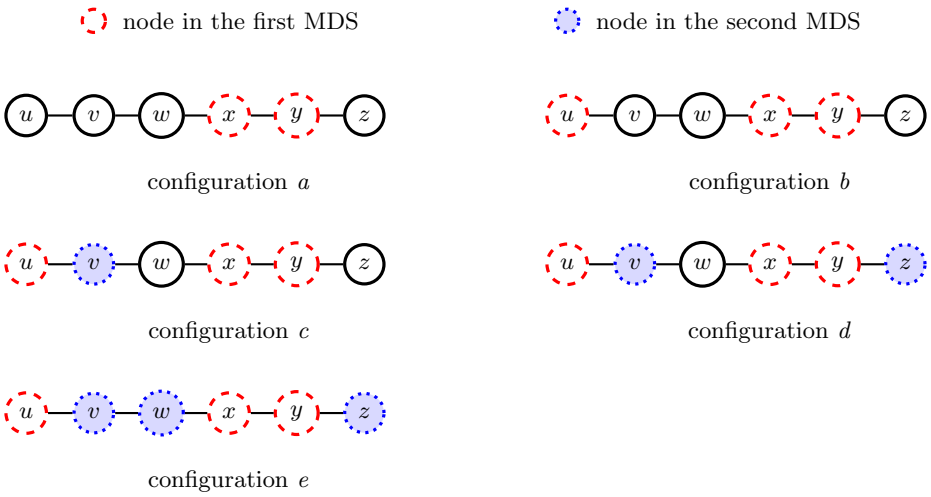


Fig. 3. an execution of algorithm 2 reaching a terminal configuration

7 Proof of correctness of Algorithm 2

Observation 1 *If a node satisfies the predicate notRedDom(v) (resp. $\neg blueOk(v)$) then $color(v) \neq red$ (resp. $color(v) \neq blue$).*

Lemma 8. *In a terminal configuration RedSet and BlueSet are minimal dominating sets.*

Proof. A node satisfying the predicate *notRedDom* is enabled (it can execute the rule $RRedIn$). A node satisfying the predicate *beNotRed* is enabled (it can execute the rule $RBlueIn$ or the rule $RRedOut$). We conclude that $RedSet$ is a minimal dominating set.
Let c be a terminal configuration. In c, the predicate *redOk* is satisfied by every node. So, no node satisfies the predicate *beNotBlue* in c.

Assume that a node v satisfies $\#Blue(v) = 0$, in c. As v is not isolated, we have $|N[v]| > 1$.
If $color(v) = \perp$ then v can execute the rule $RBlueIn$ from c. So, $color(v) = red$.
Let u be a neighbor of v, If $color(u) = \perp$ then $\neg blueOk(u)$ is satisfied so u can execute the rule $RBlueIn$ from c.
Therefore, all nodes of $N[v]$ have the color red in c. So $\forall w \in N[v]$ we have $\#Red(w) > 1$. We conclude that v satisfies the predicate $beNotRed(v)$. So, v is enabled: there is a contradiction.
Hence, in c, every node v satisfies $\#Blue(v) > 0$. We conclude that $BlueSet$ is a minimal dominating set, as no node satisfies the predicate $beNotBlue$ in c. □

7.1 Convergence of Algorithm 2

After the execution of $ROutBlue$ or $ROutRed$, a node is disabled forever - Lemma 12. The execution of $RRedIn$ is the last move of a node - Lemma 14.
Assume that during its first move, a node v executes $RInBlue$. The second move of v (if it exists), would be its last one as it would be the execution of rule $RBlueOut$ or rule $RRedIn$. So a node v executes at most two moves.

Lemma 9. *The predicate $redOk(v)$ is closed for every node v.*

Proof. We prove that the predicate $\#Red(v) \geq 1$ is closed for every node v.

Let $c1$ be a configuration where $\#Red(v) = 1$. Let u be the single node of $N[v]$ having the color red in $c1$. The node u satisfies $\neg beNotRed(u)$, and $\neg canBeBlue(u)$. So, the node u is disabled in $c1$. We conclude that in the configuration reached after any computation step from $c1$, we have $\#Red(v) \geq 1$.
Let $c2$ be a configuration where $nr = \#Red(v) > 1$. During any computation step from $c2$, a single node of $N[v]$ executes a move (because a central daemon is assumed). So, in the configuration reached after any computation step from $c2$, we have $\#Red(v) \in [nr-1, nr+1]$. We conclude that $\#Red(v) \geq 1$ in $c2$. □

Lemma 10. *The predicate $redOk(v) \wedge \#Blue(v) \geq 1$ is closed for every node v.*

Proof. Let $c1$ be a configuration where $redOk(v) \wedge \#Blue(v) = 1$. Let u be the single node of $N[v]$ having the color $blue$ in $c1$. The node u satisfies $\neg notRedDom(u)$ in $c1$ (because $redOk(v) \Rightarrow \#Red(u) > 0$). The node u satisfies $\neg beNotBlue(u)$ in $c1$ (because $\#Blue(v) = 1$). So, the node u is disabled in $c1$. Therefore in the configuration reached after any computation step from $c1$, we have $\#Blue(v) \geq 1$.
Let $c2$ be a configuration where $redOk(v) \wedge \#Blue(v) > 1$. We denoted nb the value of $\#Blue(v)$ in $c2$. During any computation step from $c2$, a single node of $N[v]$ executes a move (because a central daemon is assumed). So, in the configuration reached after a computation step from $c2$, we have $\#Blue(v) \in [nb-1, nb+1]$. So, $\#Blue(v) \geq 1$ in $c2$.
We conclude that $redOk(v) \wedge \#Blue(v) \geq 1$ is closed, as $redOk(v)$ is closed (Lemma 9). □

Lemma 11. *The predicate $redOk(v) \wedge blueOk(v)$ is closed.*

Proof. Let c be a configuration where the predicate $redOk(v) \wedge blueOk(v)$ is satisfied. Let c' be a configuration reached from c by a single computation step. In c, we have $redOk(v) \wedge \#Blue(v) \geq 1$ by hypothesis. As this predicate is closed (Lemma 10), in c' we have $redOk(v) \wedge \#Blue(v) \geq 1$.
Let u be a neighbor of v having the color red in c'. In c, by hypothesis we have $\#Red(u) \geq 1$ so the rule $RRedIn$ is disabled on u in c. Therefore the color u in c is also red. By definition of $blueOk(v)$, we have $\#Blue(u) \geq 1$, in c. Therefore, in c', we have $redOk(u) \wedge \#Blue(u) \geq 1$ as this predicate is closed (Lemma 10). Thus in c', any neighbor, w of v satisfies the predicate $(color(w) \neq red) \vee (\#Blue(w) \geq 1)$. We conclude that $redOk(v) \wedge blueOk(v)$ is satisfied in c'. □

If a node v can execute the $RRedOut$ rule or the $RBlueOut$ rule in the configuration c; then it verifies the predicate $(redOk(v) \wedge blueOk(v))$ in c and along any execution from c. So, the guard of rule $RRedIn$ and the guard of the rule $RBlueIn$ are never verified by v along any execution from c.

Lemma 12. *A node is disabled forever after executing the RRedOut rule or the RBlueOut rule.*

Proof. Let cs be a computation step from c to c' where a node v executes a $RRedOut$ move or a $RBlueOut$ move. In c, the guard of the $RRedOut$ rule or the guard of the $RBlueOut$ rule is satisfied.
The predicate $redOk(v)$ is satisfied in c (we have $beNotRed(v) \Rightarrow redOk(v)$). In c, the predicate $blueOk(v)$ is satified as $(beNotBlue(v) \Rightarrow blueOk(v))$. The predicate $redOk(v) \wedge blueOk(v)$ is closed (Lemma 11). The rule $RBlueIn$ and the rule $RRedIn$ are and stay disabled on v along any execution from c.
The next move by a node v after executing a $RRedOut$ move or a $RBlueOut$ move would be a $RRedIn$ move or a$RBlueIn$ move because $color(v) = \perp$. We conclude that from c', v is forever disabled. □

If a node v verifies $(\#Red(v) = 1 \wedge color(v) = red)$ in the configuration c; then v is disabled in c and $RRedIn$ is disabled on any v's neighbor in c. So, this predicate is always verified by v along any execution from c.
If this predicate is verified by v then v is disabled; this predicate is verified after the execution by v of rule $RRedIn$.

Lemma 13. *The predicate $\#Red(v) = 1 \wedge color(v) = red$ is closed.*

Proof. Let c be a configuration where v satisfies the predicate $\#Red(v) = 1 \wedge color(v) = red$. v is disabled in c because the predicate $\neg beNotRed(v)$ is satisfied. So after any computation step from c, we have $color(v) = red$.
In c, the predicate $\neg notRedDom(u)$ is satisfied by any v' neighbor; they cannot execute the rule $RRedIn$ in any computation step from c. So after any computation step from c, we have $\#Red(v) = 1$.
We conclude that after any computation step from c, we have $\#Red(v) = 1 \wedge color(v) = red$. □

Lemma 14. *A node v is disabled forever after executing a $RRedIn$ move.*

Proof. Assume that in the configuration c, v executes the rule $RRedIn$ to reach the configuration c'. In c, none v's neighbors has the color red.
As v is the single node to execute an action during this computation step; in c', we have $\#Red(v) = 1 \wedge color(v) = red$.
According to Lemma 13, along any execution from c', we have $\#Red(v) = 1 \wedge color(v) = red$. We conclude that the node v is and stays disabled along any execution after its $RRedIn$ move. □

Corollary 3. *A node performs at most two moves.*

Proof. If the first move of a node v is $RRedIn$, $ROutBlue$ or $ROutRed$, it would be it last move (Lemma 12 and 14).
Assume that the first move of a node v is $RInBlue$. The second move of v (if it exists), would be either a $RBlueOut$ move or a $RRedIn$ move. The second move would be its last one. So v performs at most two moves. □

References

1. Belhoul, Y., Yahiaoui, S., Kheddouci, H.: Efficient self-stabilizing algorithms for minimal total k-dominating sets in graphs. Information Processing Letters **114**(7), 339–343 (2014)
2. Dekar, L., Kheddouci, H.: Distance-2 self-stabilizing algorithm for a b-coloring of graphs. In: SSS'08, the 10th International Symposium on Stabilization, Safety, and Security of Distributed Systems, Springer LNCS: 5340. pp. 19–31 (2008)
3. Gairing, M., Goddard, W., Hedetniemi, S.T., Kristiansen, P., McRae, A.A.: Distance-two information in self-stabilizing algorithms. Parallel Processing Letters **14**(3-4), 387–398 (2004)
4. Hedetniemi, S.M., Hedetniemi, S.T., Jiang, H., Kennedy, K., McRae, A.A.: A self-stabilizing algorithm for optimally efficient sets in graphs. Information Processing Letters **112**(16), 621–623 (2012)
5. Hedetniemi, S.M., Hedetniemi, S.T., Kennedy, K.E., McRae, A.A.: Self-stabilizing algorithms for unfriendly partitions into two disjoint dominating sets. Parallel Processing Letters **23**(1) (2013)
6. Hedetniemi, S.T., Jacobs, D.P., Kennedy, K.E.: A theorem of Ore and self-stabilizing algorithms for disjoint minimal dominating sets. Theoretical Computer Science **593**, 132–138 (2015)
7. Kamei, S., Kakugawa, H.: A self-stabilizing algorithm for two disjoint minimal dominating sets with safe convergence. In: 24th IEEE International Conference on Parallel and Distributed Systems (ICPADS). pp. 365–372 (2018)
8. Ore, O.: Theory of graphs. Amer. Math. Soc. (1962)
9. Srimani, P.K., Wang, J.Z.: Self-stabilizing algorithm for two disjoint minimal dominating sets. Information Processing Letters **147**, 38–43 (2019)
10. Turau, V.: Efficient transformation of distance-2 self-stabilizing algorithms. Journal of Parallel and Distributed Computing **72**(4), 603–612 (2012)
11. Turau, V.: Self-stabilizing algorithms for efficient sets of graphs and trees. Information Processing Letters **113**(19-21), 771–776 (2013)
12. Yahiaoui, S., Belhoul, Y., Haddad, M., Kheddouci, H.: Self-stabilizing algorithms for minimal global powerful alliance sets in graphs. Information Processing Letters **113**(10-11), 365–370 (2013)