Self-Stabilizing *(f,g)*-Alliances with Safe Convergence*

Fabienne Carrier Ajoy K. Datta Stéphane Devismes Lawrence L. Larmore
Yvan Rivierre

Abstract

Given two functions f and g mapping nodes to non-negative integers, we give a silent self-stabilizing algorithm that computes a minimal (f, g)-alliance in an asynchronous network with unique node IDs, assuming that every node p has a degree at least $g(p)$ and satisfies $f(p) \geq g(p)$. Our algorithm is *safely converging* in the sense that starting from any configuration, it first converges to a (not necessarily minimal) (f, g)-alliance in at most four rounds, and then continues to converge to a minimal one in at most $5n + 4$ additional rounds, where n is the size of the network. Our algorithm is written in the shared memory model. It is proven assuming an unfair (distributed) daemon. Its memory requirement is $\Theta(\log n)$ bits per process, and it takes $O(n \cdot \Delta^3)$ steps to stabilize, where Δ is the degree of the network.

Keywords:

1 Introduction

In 1974, Dijkstra introduced the notion of *self-stabilization* of a distributed system [2]. He defined a system to be self-stabilizing if, "regardless of its initial configuration, it is guaranteed to arrive at a legitimate configuration in finite time." Thus, a self-stabilizing system can withstand *any* finite number of transient faults. Indeed, after transient faults hit the system and place it in some arbitrary configuration, a self-stabilizing algorithm allows the system to recover without external (*e.g.*, human) intervention in finite time. Thus, self-stabilization makes no hypothesis on the nature or extent of transient faults that could hit the system, and recovers from the effects of those faults in a unified manner. However, self-stabilization has some drawbacks; perhaps the main one is *temporary loss of safety*, *i.e.*, after the occurrence of transient faults, there is a finite period of time — called the *stabilization phase* — before the system returns to a legitimate configuration. During this phase, there is no guarantee of safety. Several approaches have been introduced to offer more stringent guarantees during the stabilization phase, *e.g.*, *fault-containment* [3], *superstabilization* [4], *time-adaptivity* [5], and *safe convergence* [6].

We consider here the notion of *safe convergence*. The main idea behind this concept is the following: For a large class of problems, it is often hard to design self-stabilizing algorithms that guarantee a small stabilization time, even after few transient faults [7]. A long stabilization time is frequently due to the strong specifications that a legitimate configuration must satisfy. The goal of a *safely converging self-stabilizing algorithm* is to first "quickly" (within $O(1)$ rounds is the usual rule) converge to a *feasible* legitimate configuration, where a minimum quality of service is guaranteed. Once such a feasible legitimate configuration is reached, the system continues to converge to an *optimal* legitimate configuration, where more stringent conditions are required. Safe convergence is especially interesting for self-stabilizing algorithms that compute optimized data structures, *e.g.*, minimal dominating sets [6], approximation of the minimum weakly connected dominating set [8], and approximately minimum connected dominating sets [9, 10].

In this work, we consider the (f, g)-*alliance problem*. Let $G = (V, E)$ be an undirected graph and f and g two non-negative integer-valued functions on nodes. For every node $p \in V$, let $\mathcal{N}_p$ (resp. δ_p) denote the

*A preliminary version of this paper has been presented in SSS'2013 [1].

set of neighbors (resp. the degree) of p in G. A subset of nodes $A \subseteq V$ is an (f,g)-*alliance* of G if and only if every node $x \notin A$ has at least $f(x)$ neighbors in A, and every node $y \in A$ has at least $g(y)$ neighbors in A. More formally, A is an (f,g)-alliance if and only if

$$(\forall p \in V \setminus A, |\mathcal{N}_p \cap A| \geq f(p)) \wedge (\forall q \in A, |\mathcal{N}_q \cap A| \geq g(q))$$

We say that an (f,g)-*alliance* is *minimal* if no proper subset of A is an (f,g)-alliance of G. The (minimal) (f,g)-alliance problem is then the problem of finding a (minimal) (f,g)-alliance for given f and g on a given graph.

The (f,g)-alliance problem is a generalization of several problems that are of interest in distributed computing. Consider any subset S of nodes:

1. S is a (minimal) domination set [11] if and only if S is a (minimal) $(1,0)$-alliance;
2. more generally, S is a (minimal) k-domination set [11] if and only if S is a (minimal) $(k,0)$-alliance;
3. S is a (minimal) k-tuple dominating set [12] if and only if S is a (minimal) $(k,k-1)$-alliance;
4. S is a (minimal) global offensive alliance [13] if and only if S is a (minimal) $(f,0)$-alliance, where $f(p) = \lceil \frac{\delta_p+1}{2} \rceil$ for all p;
5. S is a (minimal) global defensive alliance [14] if and only if S is a (minimal) $(1,g)$-alliance, where $g(p) = \lceil \frac{\delta_p+1}{2} \rceil$ for all p;
6. S is a (minimal) global powerful alliance [15] if and only if S is a (minimal) (f,g)-alliance, such that $f(p) = \lceil \frac{\delta_p+1}{2} \rceil$ and $g(p) = \lceil \frac{\delta_p}{2} \rceil$ for all p.

We remark that (f,g)-alliances have applications in the fields of population protocols [16] and server allocation in computer networks [17].

1.1 Our Contribution

Let f and g be two functions mapping nodes to non-negative integers. We say $f \geq g$ if and only if $\forall p \in V, f(p) \geq g(p)$.

In this paper, we give a silent self-stabilizing algorithm, $\mathcal{MA}(f,g)$, that computes a minimal (f,g)-alliance in an asynchronous network with unique process IDs, where $f \geq g$ and $\delta_p \geq g(p)$ for all p.[1]

We remark that the class of minimal (f,g)-alliances with $f \geq g$ generalizes the classes of minimal dominating sets, k-dominating sets, k-tuple dominating sets, global offensive alliances, and global powerful alliances. However, minimal global defensive alliances do not belong to this class.

Our algorithm $\mathcal{MA}(f,g)$ is *safely converging* in the sense that, starting from any configuration, it first converges to a (not necessarily minimal) (f,g)-alliance in at most four rounds, and then continues to converge to a minimal (f,g)-alliance in at most $5n+4$ additional rounds, where n is the size of the network. Our algorithm is written in the shared memory model, and is proven assuming an *unfair* (distributed) daemon, the strongest daemon of this model. $\mathcal{MA}(f,g)$ uses $\Theta(\log n)$ bits per process, and stabilizes to a terminal (legitimate) configuration in $O(n \cdot \Delta^3)$ steps, where Δ is the degree of the network. Finally, $\mathcal{MA}(f,g)$ does not make use of any bound on global parameters of the network (such as its size or its diameter).

1.2 Related Work

The (f,g)-alliance problem is introduced by Dourado *et al.* [18]. In that paper, the authors give several distributed algorithms for that problem and its variants, but none of them is self-stabilizing. To the best of our knowledge, the conference presentation of this paper [1] is the first publication on the subject of (f,g)-alliances after the introductory paper by Dourado *et al.* However, there are several self-stabilizing solutions

[1]We assume that $\delta_p \geq g(p)$ to ensure that an (f,g)-alliance always exists; namely $A = V$.

for particular instances of (minimal) (f,g)-alliances, *e.g.*, [19, 6, 20, 21, 22, 15]. However, safely converging solutions are given only in [19, 6].

Algorithms given in [20, 22] work in anonymous networks and require $O(1)$ bits per process, however, they both assume a central daemon. More precisely, Srimani and Xu [20] give several algorithms which compute minimal global offensive and 1-minimal[2] defensive alliances in $O(n^3)$ steps. Wang *et al* [22] give a self-stabilizing algorithm to compute a minimal k-dominating set in $O(n^2)$ steps.

All other solutions [19, 6, 21, 15] consider arbitrary identified networks and require $\Theta(\log n)$ bits per process. Turau [21] gives a self-stabilizing algorithm to compute a minimal dominating set in $9n$ steps, assuming an unfair distributed daemon. Yahiaoui *et al* [15] give self-stabilizing algorithms to compute a minimal global powerful alliance. Their solution assumes an unfair distributed daemon and stabilizes in $O(n \cdot m)$ steps, where m is the number of edges in the network.

A safely converging self-stabilizing algorithm is given in [6] for computing a minimal dominating set. The algorithm first computes a (not necessarily minimal) dominating set in $O(1)$ rounds and then safely stabilizes to a *minimal* dominating set in $O(\mathcal{D})$ rounds, where $\mathcal{D}$ is the diameter of the network. However, a synchronous daemon is required. A safely converging self-stabilizing algorithm for computing minimal global offensive alliances is given in [19]. This algorithm also assumes a synchronous daemon. It first computes a (not necessarily minimal) global offensive alliance within two rounds, and then safely stabilizes to a *minimal* global offensive alliance within $O(n)$ additional rounds.

1.3 Roadmap

In the next section we describe our model of computation and give some basic definitions. We define our algorithm $\mathcal{MA}(f,g)$ in Section 3. In Section 4, we prove correctness of $\mathcal{MA}(f,g)$ and analyze its complexity. We write concluding remarks and perspectives in Section 5.

2 Preliminaries

2.1 Distributed Systems

We consider distributed systems of n processes equipped of *unique identifiers* (simply referred to as IDs in the following). As is common in the literature, we assume an ID is stored using $\Theta(\log n)$ bits. Further, by an abuse of notation, we identify a process with its identifier whenever convenient.

Each process p can directly communicate with a subset $\mathcal{N}_p$ of other processes, called its *neighbors*. We assume bidirectional communications, *i.e.*, if $q \in \mathcal{N}_p$, then $p \in \mathcal{N}_q$. For every process p, let $\delta_p = |\mathcal{N}_p|$, the *degree of* p. Let $\Delta = \max_{p \in V} \delta_p$, the *degree of the network*. The topology of the system is a simple undirected graph $G = (V,E)$, where V is the set of processes and E is the set of edges, each edge being an unordered pair of neighboring processes.

2.2 Computational Model

We assume the *shared memory model* of computation introduced by Dijkstra [2], where each process communicates with its neighbors using a finite set of *locally shared variables*, henceforth called simply *variables*. Each process can read its own variables and those of its neighbors, but can write only to its own variables. Each process operates according to its (local) *program*. We define a *distributed algorithm* to be a collection of n *programs*, each operating on a single process. The program of each process is a finite ordered set of actions, where the ordering defines *priority*. This priority is the order of appearance of actions in the text of the program: Action A has higher priority than Action B if A appears before B in the text. A process p is not enabled to execute any (lower priority) action if it is enabled to execute an action of higher priority. Let $\mathcal{A}$ be a distributed algorithm, consisting of a local program $\mathcal{A}(p)$ for each process p. Each action in $\mathcal{A}(p)$ is

[2]The definition of 1-minimal is given in Section 2.

of the following form:

$$\langle label \rangle \ :: \ \langle guard \rangle \ \to \ \langle statement \rangle$$

Labels are only used to identify actions. The *guard* of an action in $\mathcal{A}(p)$ is a Boolean expression involving the variables of p and its neighbors. The *statement* of an action in $\mathcal{A}(p)$ updates some variables of p. The *state* of a process in $\mathcal{A}$ is defined by the values of its variables in $\mathcal{A}$. A *configuration* of $\mathcal{A}$ is an instance of the states of processes in $\mathcal{A}$. $\mathcal{C}_\mathcal{A}$ is the set of all possible configurations of $\mathcal{A}$. (When there is no ambiguity, we omit the subscript $\mathcal{A}$.) An action can be executed only if its guard evaluates to TRUE; in this case, the action is said to be *enabled*. A process is said to be enabled if at least one of its actions is enabled. We denote by *Enabled*(γ) the subset of processes that are enabled in configuration γ. When the configuration is γ and *Enabled*(γ) $\neq \emptyset$, a *daemon*[3] (or scheduler) selects a non-empty set $\mathcal{X} \subseteq$ *Enabled*(γ); then every processof $\mathcal{X}$ *atomically* executes its highest priority enabled action, leading to a new configuration γ', and so on. The transition from γ to γ' is called a *step* (of $\mathcal{A}$). The possible steps induce a binary relation over configurations of $\mathcal{A}$, denoted by $\mapsto$. An *execution* of $\mathcal{A}$ is a maximal sequence of its configurations $e = \gamma_0\gamma_1 \ldots \gamma_i \ldots$ such that $\gamma_{i-1} \mapsto \gamma_i$ for all $i > 0$. The term "maximal" means that the execution is either infinite, or ends at a *terminal* configuration in which no action of $\mathcal{A}$ is enabled at any process. As previously stated, each step from a configuration to another is driven by a daemon. In this paper we assume the daemon is *distributed* and *unfair*. "Distributed" means that while the configuration is not terminal, the daemon should select at least one enabled process, maybe more. "Unfair" means that there is no fairness constraint, *i.e.*, the daemon might never select an enabled process unless it is the only enabled process.

We say that a process p is *neutralized* during the step $\gamma_i \mapsto \gamma_{i+1}$ if p is enabled at γ_i and not enabled at γ_{i+1}, but does not execute any action between these two configurations. An enabled process is neutralized if at least one neighbor of p changes its state between γ_i and γ_{i+1}, and this change makes the guards of all actions of p false. To evaluate time complexity, we use the notion of *round*. This notion captures the execution rate of the slowest process in any execution. The first round of an execution e, noted e', is the minimal prefix of e in which every process that is enabled in the initial configuration either executes an action or becomes neutralized. Let e'' be the suffix of e starting from the last configuration of e'. The second round of e is the first round of e'', and so forth.

2.3 Self-Stabilization, Silence, and Safe Convergence

Let $\mathcal{A}$ be a distributed algorithm. Let $\mathbb{P}$ and $\mathbb{P}'$ be two predicates on $\mathcal{C}$, the set of all possible configurations of $\mathcal{A}$. We say that a configuration γ *satisfies* $\mathbb{P}$ if $\mathbb{P}(\gamma) =$ TRUE. We say that $\mathbb{P}$ is *closed* under $\mathcal{A}$ if for each possible step $\gamma \mapsto \gamma'$ of $\mathcal{A}$, $\mathbb{P}(\gamma) \Rightarrow \mathbb{P}(\gamma')$. We say that $\mathcal{A}$ *converges* from $\mathbb{P}$ to $\mathbb{P}'$ if every execution of $\mathcal{A}$ which starts from a configuration satisfying $\mathbb{P}$ contains a configuration which satisfies $\mathbb{P}'$.

We are interesting in algorithms which converge from an arbitrary configuration to a configuration where output variables define a data structure, namely an (f, g)-alliance. Hence, we define a specification as a predicate $\mathbb{S}$ on $\mathcal{C}$ which is TRUE if and only if the outputs define the expected data structure.

Self-Stabilization $\mathcal{A}$ is *self-stabilizing w.r.t. specification* $\mathbb{S}$ if there is a predicate $\mathbb{P}$ on $\mathcal{C}$, called *legitimacy predicate*, such that:

Correctness: $\forall \gamma \in \mathcal{C}, \mathbb{P}(\gamma) \Rightarrow \mathbb{S}(\gamma)$;

Closure: $\mathbb{P}$ is *closed* under $\mathcal{A}$;

Convergence: $\mathcal{A}$ *converges* from TRUE to $\mathbb{P}$.

The configurations which satisfy the legitimacy predicate $\mathbb{P}$ are simply called *legitimate configurations*, and other configurations are said to be *illegitimate*. The *stabilization time* is the maximum time (in steps or rounds) to reach a legitimate configuration starting from any configuration.

[3]The daemon achieves the asynchrony of the system.

Silence $\mathcal{A}$ is *silent* if all its executions are finite [23]. By definition, $\mathcal{A}$ is silent and self-stabilizing *w.r.t.* specification $\mathbb{S}$ if the following two conditions hold:

1. all executions of $\mathcal{A}$ are finite; and
2. all terminal configurations of $\mathcal{A}$ satisfy $\mathbb{S}$.

Safely Converging Self-Stabilization In [6], Kakugawa and Masuzawa defined safely converging self-stabilization as follows.

A distributed algorithm $\mathcal{A}$ is *safely converging self-stabilizing with respect to an ordered pair of specifications* $(\mathbb{S}_1, \mathbb{S}_2)$ (see Figure 1) if the following three properties hold:

1. $\mathcal{A}$ is *self-stabilizing w.r.t.* $\mathbb{S}_1$;
2. Let $\mathbb{P}_1$ be the legitimacy predicate of $\mathcal{A}$ *w.r.t.* specification $\mathbb{S}_1$. There exists a predicate $\mathbb{P}_2$ on $\mathcal{C}$ such that:
   (a) $\forall \gamma \in \mathcal{C}, \mathbb{P}_2(\gamma) \Rightarrow \mathbb{S}_2(\gamma)$
   (b) $\mathbb{P}_2$ is *closed* under $\mathcal{A}$; and
   (c) $\mathcal{A}$ *converges* from $\mathbb{P}_1$ to $\mathbb{P}_2$.

The configurations satisfying $\mathbb{P}_1$ are said to be *feasible legitimate*. The configurations satisfying $\mathbb{P}_2$ are said to be *optimal legitimate*. Although not specified by the definition, safely converging self-stabilization is normally used when convergence to a feasible legitimate configuration (which provides a minimal level of service) is very quick, typically within $O(1)$ rounds, but where subsequent convergence to an optimal legitimate configuration may take longer. Accordingly, we define the *first convergence time* as the maximum time to reach a feasible legitimate configuration, starting from any configuration. The *second convergence time* is the maximum time to reach an optimal legitimate configuration, starting from any feasible legitimate configuration. So, the *stabilization time* is, by definition, less or equal to the sum of the first and second convergence times.

Notice that, by definition, $\mathcal{A}$ is *safely converging self-stabilizing w.r.t.* $(\mathbb{S}_1, \mathbb{S}_2)$ if the following three conditions hold:

1. $\mathcal{A}$ is *self-stabilizing w.r.t.* $\mathbb{S}_1$;
2. $\forall \gamma \in \mathcal{C}, \mathbb{S}_2(\gamma) \Rightarrow \mathbb{S}_1(\gamma)$; and
3. $\mathcal{A}$ is *self-stabilizing w.r.t.* $\mathbb{S}_2$.

Notice that, by definition, the *stabilization time* of $\mathcal{A}$ *w.r.t.* specification $(\mathbb{S}_1, \mathbb{S}_2)$ (resp. $\mathbb{S}_2$) is, by definition, less or equal to the sum of the first and second convergence times.

2.4 (f, g)-alliances, Minimality, and 1-Minimality

We now introduce some notation that will be useful in our discussion of (f, g)-alliances.

Given two non-negative integer-valued functions f, g on a network $G = (V, E)$ and a set of nodes $A \subseteq V$, we define the non-negative integer-valued function $h_A(p) = \begin{cases} f(p) \text{ if } p \notin A \\ g(p) \text{ if } p \in A \end{cases}$

Thus, A is an (f, g)-alliance of G if and only if $|\mathcal{N}_p \cap A| \geq h_A(p)$ for all p.

We recall that an (f, g)-alliance A of a graph G is *minimal* if and only if no proper subset of A is an (f, g)-alliance. We define A to be a 1-*minimal* (f, g)-alliance if deletion of just one member of A causes A to not be an (f, g)-alliance. Surprisingly, a 1-*minimal* (f, g)-alliance is not necessarily a *minimal* (f, g)-alliance, [18]. However, we have the following property:

Property 1 [18] *Given two non-negative integer-valued functions f and g on nodes*

1. *Every minimal (f, g)-alliance is a 1-minimal (f, g)-alliance, and*
2. *if $f \geq g$, every 1-minimal (f, g)-alliance is a minimal (f, g)-alliance.*

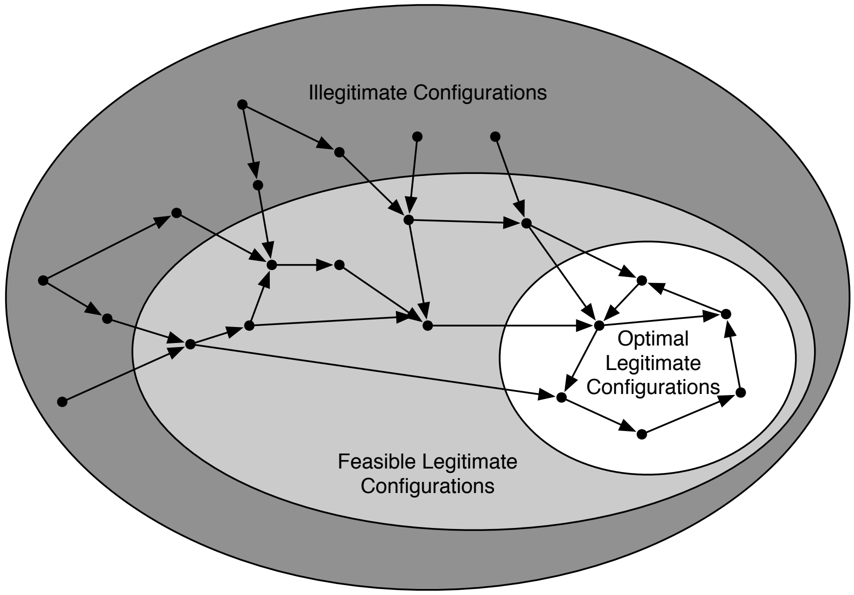

Figure 1: Safe Converging Self-Stabilization.

3 The Algorithm

The formal code of $\mathcal{MA}(f, g)$ is given as Algorithm 1. Given input functions f and g, $\mathcal{MA}(f, g)$ computes a single Boolean variable $p.inA$ for each process p. For any configuration γ, let $A_\gamma = \{p \in V : \ p.inA\}$. (We omit the subscript γ when it is clear from the context.) If γ is terminal, then A is a 1-minimal (f, g)-alliance, and consequently, if $f \geq g$, A_γ is a minimal (f, g)-alliance.

As a consequence, we instantiate the previous function $h_{_A}(p)$ as follows:

$$h_{_A}(p) = \begin{cases} f(p) \text{ if } \neg p.inA \\ g(p) \text{ if } p.inA \end{cases}$$

During an execution, a process may need to leave or join A. The basic idea of safe convergence is that processes are easily added to A to achieve feasible legitimacy quickly, and then processes are more carefully deleted to achieve optimal legitimacy.

3.1 Leaving A

Action `Leave` allows a process to leave A. To obtain 1-minimality, we allow a process p to leave A if

Requirement 1: p will have enough neighbors in A (*i.e.*, at least $f(p)$) once it has left A, and

Requirement 2: each $q \in \mathcal{N}_p$ will still have enough neighbors in A (*i.e.*, at least $h_{_A}(q)$) once p has been deleted from A.

Ensuring Requirement 1 To maintain Requirement 1, we implement our algorithm in such a way that deletion from A is *locally sequential*, *i.e.*, during a step, at most one process can leave A in the neighborhood of each process p (including p itself). Using this locally sequential mechanism, if a process p wants to leave A, it must first verify that $\texttt{NbA}(p) \geq f(p)$ before leaving A. Since no neighbor of p can leave A at the same step, Requirement 1 still holds once p has left A.

Algorithm 1 $\mathcal{MA}(f, g)$, code for each process p

Variables:

$p.inA$, $p.busy$: Booleans
$p.choice \in \mathcal{N}_p \cup \{\bot\}$
$p.nbA \in [0..\delta_p]$

Macros:

$\texttt{NbA}(p)$	$=$	$\lvert\{q \in \mathcal{N}_p, q.inA\}\rvert$
$\texttt{Cand}(p)$	$=$	$\{q \in \mathcal{N}_p, q.inA \wedge \neg q.busy\}$
$\texttt{MinCand}(p)$	$=$	$\min(\texttt{Cand}(p) \cup \{\infty\})$
$\texttt{ChosenCand}(p)$	$=$	**if** $\texttt{Cand}(p) \neq \emptyset \wedge \texttt{HasExtra}(p) \wedge (\texttt{IamCand}(p) \Rightarrow \texttt{MinCand}(p) < p)$ **then** $\texttt{MinCand}(p)$ **else** $\bot$
$\texttt{Choice}(p)$	$=$	**if** $p.choice = \bot$ **then** $\texttt{ChosenCand}(p)$ **else** $\bot$

Predicates:

$\texttt{IsMissing}(p)$	$\equiv$	$\exists q \in \mathcal{N}_p(q.nbA < h_A(q))$
$\texttt{IsExtra}(p)$	$\equiv$	$\forall q \in \mathcal{N}_p(q.nbA > h_A(q))$
$\texttt{HasExtra}(p)$	$\equiv$	$\texttt{NbA}(p) > h_A(p)$
$\texttt{IsBusy}(p)$	$\equiv$	$\texttt{NbA}(p) < f(p) \vee \neg\texttt{IsExtra}(p)$
$\texttt{IamCand}(p)$	$\equiv$	$p.inA \wedge \neg\texttt{IsBusy}(p)$
$\texttt{MustJoin}(p)$	$\equiv$	$\neg p.inA \wedge (\texttt{NbA}(p) < f(p) \vee \texttt{IsMissing}(p)) \wedge (\forall q \in \mathcal{N}_p, q.choice \neq p)$
$\texttt{CanLeave}(p)$	$\equiv$	$p.inA \wedge \texttt{NbA}(p) \geq f(p) \wedge (\forall q \in \mathcal{N}_p, q.choice = p) \wedge p.choice = \bot$

Actions:

`Join`	::	$\texttt{MustJoin}(p)$	$\rightarrow$	$p.inA \leftarrow \text{TRUE}$ $p.choice \leftarrow \bot$ $p.nbA \leftarrow \texttt{NbA}(p)$
`Vote`	::	$p.choice \neq \texttt{ChosenCand}(p)$	$\rightarrow$	$p.choice \leftarrow \texttt{Choice}(p)$ $p.nbA \leftarrow \texttt{NbA}(p)$ $p.busy \leftarrow \texttt{IsBusy}(p)$
`Count`	::	$p.nbA \neq \texttt{NbA}(p)$	$\rightarrow$	$p.nbA \leftarrow \texttt{NbA}(p)$
`Flag`	::	$p.busy \neq \texttt{IsBusy}(p)$	$\rightarrow$	$p.busy \leftarrow \texttt{IsBusy}(p)$
`Leave`	::	$\texttt{CanLeave}(p)$	$\rightarrow$	$p.inA \leftarrow \text{FALSE}$

The locally sequential mechanism is implemented using a neighbor pointer $p.choice$ at each process p, which takes values in $\mathcal{N}_p \cup \{\perp\}$; $p.choice = q \in \mathcal{N}_p$ means that p authorizes q to leave A, while $p.choice = \perp$ means that p does not authorize any neighbor to leave A. The value of $p.choice$ is maintained using Action `Vote`, which will be defined later.

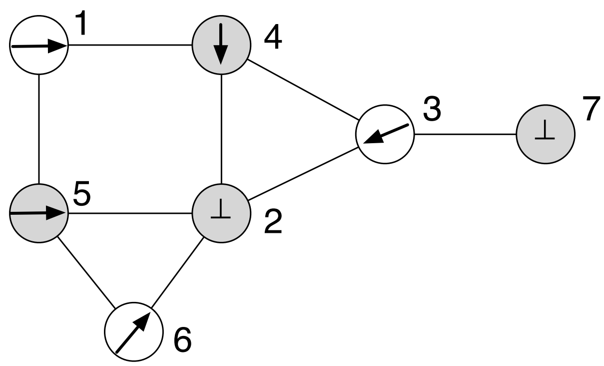

Figure 2: Neighbor pointers when computing a minimal $(1, 0)$-alliance. Numbers indicate IDs. A is the set of gray nodes. The value of *choice* is represented by an arrow or a tag "$\perp$" inside the node.

To leave A, a process p should not authorize any neighbor to leave A ($p.choice = \perp$) and should be authorized to leave by all of its neighbors ($\forall q \in \mathcal{N}_p,\ q.choice = p$). For example, consider the $(1, 0)$-alliance in Figure 2. Only Process 2 is able to leave A. Process 2 can leave A because it has enough neighbors in A (*i.e.*, 2 neighbors, while $f(2) = 1$); if Process 2 leaves A, it will still have two neighbors in A, and Requirement 1 will still hold.

Ensuring Requirement 2 This requirement is also maintained by the fact that a process p must have authorization from each of its neighbors to leave A. A neighbor q can give such an authorization to p only if q still has enough neighbors in A without p. For a process q to authorize a neighbor p to leave A, p must currently be in A, *i.e.*, $p.inA = $ TRUE, and q must have more than $h_A(q)$ neighbors in A, *i.e.*, the predicate `HasExtra`(q) should be true, For example, consider the $(1, 0)$-alliance in Figure 2. Processes 4 and 5 can designate Process 2 because they belong to A and $g(4) = g(5) = 0$. Moreover, Processes 3 and 6 can designate Process 2 because they do not belong to A and $f(3) = f(6) = 1$: if Process 2 leaves A, Process 3 (resp. Process 6) still has one neighbor in A, which is Process 7 (resp. Process 5).

Busy Processes It is possible that a neighbor p of q cannot leave A — in this case p is said to be *busy* — because one of these two conditions is TRUE:

(i) **`NbA`$(p) < f(p)$:** in this case, p does not have enough neighbors in A to be allowed to leave A.

(ii) **$\neg$`IsExtra`(p):** in this case, at least one neighbor of p needs p to stay in A.

If q chooses such a neighbor p, this may lead to a deadlock. We use the Boolean variable $p.busy$ to inform q that one of the two aforementioned conditions holds for p. Action `Flag` maintains $p.busy$. So, to prevent deadlock, q must not choose any neighbor p for which $p.busy = $ TRUE.

A process p evaluates Condition (i) by reading the values of inA of all its neighbors. On the other hand, evaluation of Condition (ii) requires that p knows, for each of its neighbors, both its status (inA) and the number of its own neighbors that are in A. This latter information is obtained using an additional variable, nbA, in which each process maintains, using Action `Count`, the number of its neighbors that are in A.

In Figure 3, consider the $(2, 0)$-alliance. Process 5 is busy because of Condition (i): it has only one neighbor in A, while $f(5) = 2$. Process 2 is busy because of Condition (ii): its neighbor 1 is not in A,

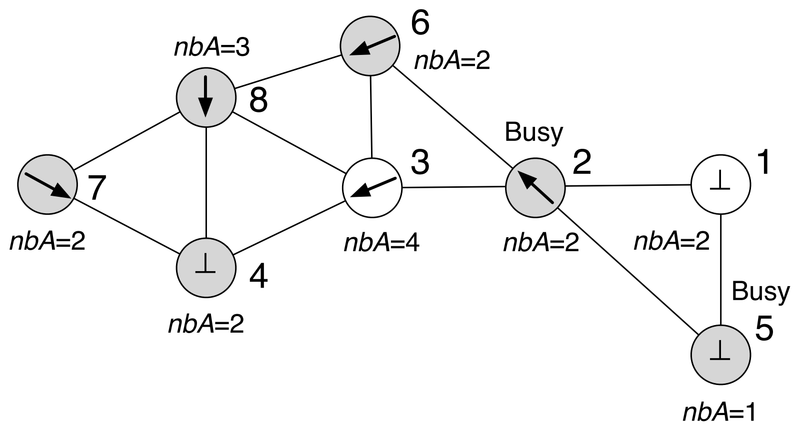

Figure 3: Busy processes when computing a minimal $(2, 0)$-alliance. Values of *nbA* are also given.

$f(1) = 2$, and it has only two neighbors in A, so it cannot authorize either of those neighbors to leave A. Consequently, Process 1 cannot designate any neighbor (all its neighbors in A are busy); while Process 3 should not designate Process 2.

Action `Vote` Hence, the value of $p.choice$ is chosen, using Action `Vote`, as follows:

1. $p.choice$ is set to $\bot$ if one of the following conditions holds:
   - $\texttt{Cand}(p) = \emptyset$, which means that no neighbor of p can leave A.
   - $\texttt{HasExtra}(p) = \text{FALSE}$, which means that p cannot authorize any neighbor to leave A.
   - $\texttt{IamCand}(p) \wedge p < \texttt{MinCand}(p)$, which means that p is also a candidate to leave A and has higher priority to leave A than any other candidate in its neighborhood. (Remember that to be allowed to leave A, p should, in particular, satisfy $p.choice = \bot$.)

     The aforementioned priorities are based on process IDs, *i.e.*, for every two process u and v, u has higher priority than v if and only if the ID of u is smaller than the ID of v.

2. Otherwise, p uses $p.choice$ to designate a neighbor that is in A, and not busy, in order to authorize it to leave A. If p has several possible candidates among its neighbors, it selects the one of highest priority (*i.e.*, of smallest ID). For example, if we consider the $(2, 0)$-alliance in Figure 3, then we can see that Process 3 designates Process 4 because it is its smallest neighbor that is both in A and not busy.

There is one last problem: A process q may change its pointer while simultaneously one of its neighbors p leaves A, and consequently Requirement 2 may be no longer hold. Indeed, q chooses a new candidate assuming that p remains in A. This may happen only if the previous value of $q.choice$ was p. To avoid this situation, we do not allow q to directly change $q.choice$ from one neighbor to another; the value must be set to $\bot$, then to a new value in $\mathcal{N}_q$.

Figures 4 and 5 illustrate this last issue in the case of a $(1, 0)$-alliance. In the step from Configuration (a) to Configuration (b) of Figure 4, Process 2 directly changes its pointer from 3 to 1. Simultaneously, 3 leaves A. Process 2 then authorizes Process 1 to leave A, but it must not yet leave A. In a subsequent step, that, Process 1 is authorized to leave A and does so at the step from Configuration (b) to Configuration (c), and thus Requirement 2 no longer holds. Figure 5 illustrates how we solve the problem. In Configuration (b), Process 3 has left, but the pointer of Process 2 is equal to $\bot$. So, Process 1 should not leave A yet, and Process 2 will not authorize it to leave.

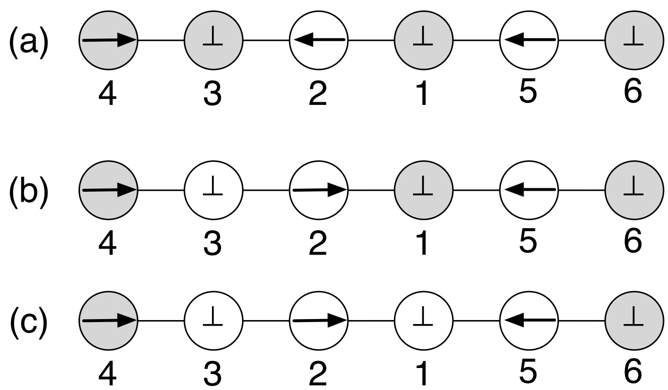

Figure 4: Requirement 2 violation when computing a minimal $(1,0)$-alliance. (We only show the values that are needed in the discussion.)

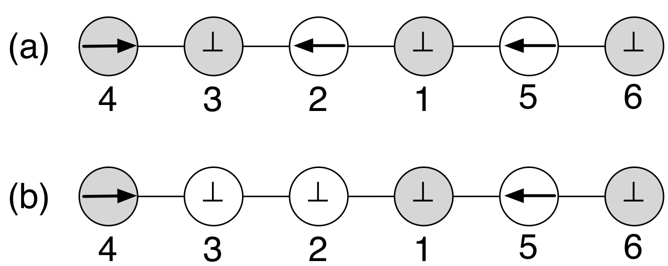

Figure 5: The reset of the neighbor pointer is applied to the example of Figure 4.

3.2 Joining A

Action `Join` allows a process to join A. A process p not in A must join A if:

(1) p does not have enough neighbors in A ($\mathtt{NbA}(p) < f(p)$), or

(2) a neighbor of p needs p to join A ($\mathtt{IsMissing}(p)$).

Moreover, to prevent p from cycling in and out of A, we require that every neighbor of p stop designating it (with their *choice* pointer) before p can join A (again). Note that all neighbors of p stop designating p immediately after it leaves A; see Action `Vote`. (Actually, this introduces a delay of only one round.)

A process evaluates condition (1) by reading the variables inA of all its neighbors. To evaluate condition (2), it needs to know, for each neighbor q, both its status *w.r.t.* A ($q.inA$) and the number of its neighbors that are in A ($q.nbA$).

4 Correctness

In this section, we prove that $\mathcal{MA}(f,g)$ is correct provided $f \geq g$. In Subsection 4.1, we define a number of useful predicates. In Subsection 4.2 we show that $\mathcal{MA}(f,g)$ is self-stabilizing under the unfair daemon and computes a minimal (f,g)-alliance, assuming $f \geq g$. In Subsection 4.3, we analyze the first and second safe convergence times of $\mathcal{MA}(f,g)$ in terms of rounds.

4.1 Predicates

For every process p we let

$$\mathtt{Fga}(p) \stackrel{\text{def}}{=} \mathtt{NbA}(p) \geq h_{A}(p)$$

When a process p satisfies $\mathtt{Fga}(p)$, this means that it is locally correct, *i.e.*, $A \cap \mathcal{N}_p$ has cardinality at least $f(p)$ or $g(p)$ if $p \notin A$ or $p \in A$, respectively. Then, by definition we have:

Remark 1 *A is an (f,g)-alliance if and only if $\mathtt{Fga}(p) = $ TRUE for all $p \in V$.*

For every process p, we let

$$\mathtt{NbAOk}(p) \stackrel{\text{def}}{=} p.nbA \geq h_{A}(p)$$

This predicate is always used in conjunction with $\mathtt{Fga}(p)$. When both predicates are TRUE at p, this means that p is locally correct and the variable $p.nbA$ gives this information to the neighbors of p.

For every process p,

$$\mathtt{ChoiceOk}(p) \stackrel{\text{def}}{=} (p.choice \neq \perp \wedge p.choice.inA) \Rightarrow \mathtt{HasExtra}(p)$$

Once $\mathtt{ChoiceOk}(p)$ holds, no neighbor of p can make p locally incorrect by leaving A.

The following predicates are defined over configurations of $\mathcal{MA}(f,g)$:

$$\mathbb{S}_{1-Minimal} \stackrel{\text{def}}{=} A \text{ is a 1-minimal } (f,g)\text{-alliance}$$

$$\mathbb{S}_{Minimal} \stackrel{\text{def}}{=} A \text{ is a minimal } (f,g)\text{-alliance}$$

4.2 Self-stabilization of $\mathcal{MA}(f,g)$

Partial Correctness We first show that if γ is a terminal configuration, it satisfies $\$_{Minimal}$, the expected specification. To prove this, we first show that A is an (f,g)-alliance at γ (Lemma 2). We then show that A is also minimal at γ (Lemma 3). To show these two results, we use two intermediate results, Lemma 1 and Corollary 1. The former states that every process of A is busy at γ, meaning that either p does not have enough neighbors in A to leave A, or that at least one neighbor of p requires that p stays in A, *i.e.,* A is 1-minimal. The latter is a simple corollary of Lemma 1 and states that no process authorizes a neighbor to leave A at γ.

At any terminal configuration, Action `Count` is disabled at every process. Thus:

Remark 2 *In any terminal configuration of $\mathcal{MA}(f,g)$, for every process p, $p.\mathrm{nbA} = \mathtt{NbA}(p) = |\mathcal{N}_p \cap A|$.*

Lemma 1 *In any terminal configuration of $\mathcal{MA}(f,g)$, $p.\mathrm{inA} \Rightarrow p.\mathrm{busy}$ for any process p.*

Proof. By contradiction. Let γ be a terminal configuration of $\mathcal{MA}(f,g)$ and assume that there is at least one process p such that $p.inA = \text{TRUE}$ and $p.busy = \text{FALSE}$ at γ. Then, for each such process p, we have $\mathtt{IsBusy}(p) = \text{FALSE}$ at γ, because Action `Flag` is disabled at every process.

Let

$$p_{\min} = \min\{p \in V, p.inA = \text{TRUE} \wedge p.busy = \text{FALSE}\} \text{ at } \gamma \qquad (1)$$

Since $\neg\mathtt{IsBusy}(p_{\min})$ at γ, we also have:

$$\begin{array}{ll} \mathtt{IsExtra}(p_{\min}) & \\ \forall q \in \mathcal{N}_{p_{\min}}(q.nbA > h_A(q)) & \\ \forall q \in \mathcal{N}_{p_{\min}}(\mathtt{NbA}(q) > h_A(q)) & \text{by Remark 2} \\ \forall q \in \mathcal{N}_{p_{\min}}, \mathtt{HasExtra}(q) & (2) \end{array}$$

Then, because $p_{\min}.inA = \text{TRUE} \wedge p_{\min}.busy = \text{FALSE}$ at γ we have:

$$\begin{array}{ll} \forall q \in \mathcal{N}_{p_{\min}}, p_{\min} \in \mathtt{Cand}(q) & (3) \\ \forall q \in \mathcal{N}_{p_{\min}}, \mathtt{Cand}(q) \neq \emptyset & (4) \end{array}$$

By (1) and (3), at γ we have:

$$\forall q \in \mathcal{N}_{p_{\min}}, \mathtt{MinCand}(q) = p_{\min} \quad (5)$$

By (1) and (5), at γ we have:

$$\forall q \in \mathcal{N}_{p_{\min}}, (\mathtt{IamCand}(q) \Rightarrow \mathtt{MinCand}(q) < q) \quad (6)$$

By (2), (4), (5), (6) and the fact that Action `Vote` is disabled, at γ we have:

$$\begin{array}{ll} \forall q \in \mathcal{N}_{p_{\min}}, \mathtt{ChosenCand}(q) = p_{\min} & \\ \forall q \in \mathcal{N}_{p_{\min}}, q.choice = p_{\min} & (7) \end{array}$$

By definition, $\mathtt{IamCand}(p_{\min})$ holds at γ. Moreover, by (1), $\mathtt{MinCand}(p_{\min}) > p_{\min}$ at γ. So, $\mathtt{MinCand}(p_{\min}) < p_{\min}$ is FALSE at γ. Hence, at γ we have $(\mathtt{IamCand}(p_{\min}) \Rightarrow \mathtt{MinCand}(p_{\min}) < p_{\min}) = \text{FALSE}$, and consequently:

$$\begin{array}{ll} \mathtt{ChosenCand}(p_{\min}) = \bot & \\ p_{\min}.choice = \bot & (\text{Action } \mathtt{Vote} \text{ is disabled}) \quad (8) \end{array}$$

Finally, because $\neg\mathtt{IsBusy}(p_{\min})$ holds at γ, we have $\mathtt{NbA}(p_{\min}) \geq f(p_{\min})$ at γ. So, by (7), (8), and the fact that $p_{\min}.inA = \text{TRUE}$ at γ, we can conclude that $\mathtt{CanLeave}(p_{\min})$ holds at γ, that is, $p_{\min}$ is enabled at γ, contradiction. □

By Lemma 1, for every process p, $\mathtt{Cand}(p) = \emptyset$ in any terminal configuration γ. Thus $\mathtt{ChosenCand}(p) = \bot$ at γ, and from the negation of the guard of Action `Vote`, we have:

Corollary 1 *In any terminal configuration of $\mathcal{MA}(f,g)$, for every process p, p.choice $= \perp$.*

Lemma 2 *In any terminal configuration of $\mathcal{MA}(f,g)$, A is an (f,g)-alliance.*

Proof. Let γ be a terminal configuration. By Remark 1, we merely need show that every process p satisfies $\texttt{Fga}(p)$ at γ. Consider the following two cases:

$p \notin A$ in γ: First, by definition, $p.inA = \text{FALSE}$ at γ. Then, γ being terminal, $\neg\texttt{MustJoin}(p)$ holds at γ. $\neg\texttt{MustJoin}(p) = \neg(\neg p.inA \wedge (\texttt{NbA}(p) < f(p) \vee \texttt{IsMissing}(p)) \wedge (\forall q \in \mathcal{N}_p, q.choice \neq p)) = p.inA \vee (\texttt{NbA}(p) \geq f(p) \wedge \neg\texttt{IsMissing}(p)) \vee (\exists q \in \mathcal{N}_p, q.choice = p)$. By Corollary 1, and since $\neg p.inA$, $\neg\texttt{MustJoin}(p)$ at γ implies that $\texttt{NbA}(p) \geq f(p) \wedge \neg\texttt{IsMissing}(p)$ at γ. Thus, $\neg p.inA \wedge \texttt{NbA}(p) \geq f(p)$ holds at γ, which implies that $\texttt{Fga}(p)$ holds at γ.

$p \in A$ at γ: First, by definition, $p.inA = \text{TRUE}$ at γ. We need to show that $\texttt{Fga}(p) = \text{TRUE}$ at γ. Assume $\texttt{Fga}(p) = \text{FALSE}$. Then, $\texttt{NbA}(p) < g(p)$. Since $\delta_p \geq g(p)$, there must be some $q \in \mathcal{N}_p$, such that $\neg q.inA$ at γ. By Remark 2, $p.nbA < g(p)$ at γ. Since $p \in \mathcal{N}_q$, $\texttt{IsMissing}(q)$ holds at γ. Now, as $q.inA = \text{FALSE}$ and $\texttt{IsMissing}(q) = \text{TRUE}$ at γ, by Corollary 1, we can conclude that $\texttt{MustJoin}(q)$ holds at γ, that is, q is enabled at γ, contradiction.

□

Lemma 3 *In any terminal configuration of $\mathcal{MA}(f,g)$, A is a 1-minimal (f,g)-alliance, and if $f \geq g$, then A is a minimal (f,g)-alliance.*

Proof. Let γ be a terminal configuration. We already know that at γ, A defines an (f,g)-alliance. Moreover, by Property 1, if A is 1-minimal and $f \geq g$, then A is a minimal (f,g)-alliance. Thus, we only need to show the 1-minimality of A.

Assume that A is not 1-minimal. Then there is a process $p \in A$ such that $A - \{p\}$ is an (f,g)-alliance. So:

1. $|A \cap \mathcal{N}_p| \geq f(p)$,
2. $\forall q \in \mathcal{N}_p, q \in A \Rightarrow |A \cap \mathcal{N}_q - \{p\}| \geq g(q)$, and
3. $\forall q \in \mathcal{N}_p, q \notin A \Rightarrow |A \cap \mathcal{N}_q - \{p\}| \geq f(q)$.

By 1, at γ we have:

$$\texttt{NbA}(p) \geq f(p) \qquad (a)$$

By 2, at γ we have:

$$\begin{array}{lll} \forall q \in \mathcal{N}_p, q.inA \Rightarrow \texttt{NbA}(q) - 1 \geq g(q) & & \\ \forall q \in \mathcal{N}_p, q.inA \Rightarrow \texttt{NbA}(q) > g(q) & & \\ \forall q \in \mathcal{N}_p, q.inA \Rightarrow q.nbA > g(q) & \text{by Remark 2} & (b) \end{array}$$

By 3, at γ we have:

$$\begin{array}{lll} \forall q \in \mathcal{N}_p, \neg q.inA \Rightarrow \texttt{NbA}(q) - 1 \geq f(q) & & \\ \forall q \in \mathcal{N}_p, \neg q.inA \Rightarrow \texttt{NbA}(q) > f(q) & & \\ \forall q \in \mathcal{N}_p, \neg q.inA \Rightarrow q.nbA > f(q) & \text{by Remark 2} & (c) \end{array}$$

By (b) and (c), $\texttt{IsExtra}(p)$ holds at γ. So, by (a), $\texttt{NbA}(p) \geq f(p)$ and $\texttt{IsExtra}(p)$ both hold γ, that is, $\neg\texttt{IsBusy}(p)$ holds at γ. $\texttt{Flag}$ is disabled at p at γ, and thus $p.busy = \text{FALSE}$ at γ. This contradicts Lemma 1, since we assumed that $p.inA = \text{TRUE}$ at γ, *i.e.* $p \in A$. □

Termination We now show that, if $f \geq g$, the unfair daemon cannot prevent $\mathcal{MA}(f, g)$ from reaching a terminal configuration, regardless of the initial configuration. The proof consists of proving that the number of steps to reach a terminal configuration, starting from an arbitrary configuration, is bounded, regardless of the choices of the daemon.

Let J be the maximum number of times any process executes Action `Join` in any execution. Lemma 4, below, states that the number of steps to reach a terminal configuration of $\mathcal{MA}(f, g)$ depends on J, as well as on both global parameters of the network, its degree Δ, and its size n.

Lemma 4 *Starting from any configuration, $\mathcal{MA}(f, g)$ reaches a terminal configuration within $O(J \cdot n \cdot \Delta^3)$ steps.*

Proof. Consider any process p in any execution e of $\mathcal{MA}(f, g)$. Let $J(p)$, $L(p)$, $C(p)$, $F(p)$, and $V(p)$ be the number of times p executes Actions `Join`, `Leave`, `Count`, `Flag` and `Vote` in e, respectively. By definition, $J(p) \leq J$.

After executing `Leave`, p should execute `Join` before executing `Leave` again. Thus

$$L(p) \leq 1 + J(p) \leq 1 + J$$

In the following discussion, let $\sharp nbA(p)$ be the number of times p modifies the value of its variable $p.nbA$; this variable can be modified either because $p.nbA \neq \mathtt{NbA}(p)$ in the initial configuration, or $p.nbA \neq \mathtt{NbA}(p)$ becomes TRUE after a neighbor of p joins or leaves A. Thus:

$$\sharp nbA(p) \leq 1 + \sum_{q \in \mathcal{N}_p} (J(q) + L(q)) \leq 1 + \Delta(2J + 1)$$

By definition, p executes Action `Count` at most $\sharp nbA(p)$ times. Thus:

$$C(p) \leq \sharp nbA(p) \leq 1 + \Delta(2J + 1)$$

In the following, we use $\sharp busy(p)$, the number of times p modifies the value of its variable $p.busy$. That variable is modified because either $p.busy \neq \mathtt{IsBusy}(p)$ holds in the initial configuration, or $p.busy \neq \mathtt{IsBusy}(p)$ becomes TRUE after a neighbor q of p joins or leaves A, or modifies its counter $q.nbA$. Thus:

$$\sharp busy(p) \leq 1 + \sum_{q \in \mathcal{N}_p} (J(q) + L(q) + \sharp nbA(q)) \leq 1 + (2 + 2J)\Delta + (1 + 2J)\Delta^2$$

By definition, p executes Action `Flag` at most $\sharp busy(p)$ times. Thus:

$$F(p) \leq \sharp busy(p) \leq 1 + (2 + 2J)\Delta + (1 + 2J)\Delta^2$$

Action `Vote` is enabled when p needs to change its pointer $p.choice$. That is, either (1) p does not want to authorize any neighbor to leave A (in this case, its pointer is reset to $\perp$), or (2) p has a new favorite candidate. In the latter case, p must first required to reset its pointer to $\perp$ (unless it is already $\perp$), because we impose a strict alternation in $p.choice$ between values of $\mathcal{N}_p$ and $\perp$. Hence, p may require up to two executions of Action `Vote` to fix the value of $p.choice$.

As for other actions, `Vote` can be initially enabled. Moreover, either case (1) or (2) occurs for p every time either (i): the variables inA of p or its neighbors are modified, or (ii): the variable $busy$ or nbA of one or more of its neighbors is modified. Therefore

$$V(p) \leq 2(1 + \textstyle\sum_{r \in \mathcal{N}_p \cup \{p\}} (J(r) + L(r)) + \sum_{q \in \mathcal{N}_p} (\sharp busy(q) + \sharp nbA(q)))$$
$$\text{and } V(p) \leq 4 + 4J + \Delta(6 + 4J) + \Delta^2(6 + 8J) + \Delta^3(2 + 4J)$$

Finally, the maximum number of steps before $\mathcal{MA}(f, g)$ reaches a terminal configuration is:

$$\begin{aligned} & n(J(p) + L(p) + C(p) + F(p) + V(p)) \\ \leq \ & n\big(7 + 6J + \Delta(9 + 8J) + \Delta^2(7 + 10J) + \Delta^3(2 + 4J)\big) \\ = \ & O(J \cdot \Delta^3 \cdot n) \end{aligned}$$

□

To complete the proof of convergence of $\mathcal{MA}(f, g)$, we now prove Lemma 11, which states that J is bounded by one if $f \geq g$. We use six technical results, Lemmas 5 through 10 below.

Lemma 5 states that processes less than two apart cannot leave A simultaneously.

Lemma 5 *Let p be a process. $\forall q, q' \in \mathcal{N}_p \cup \{p\}$, if $q' \neq q$, then q and q' cannot leave A in the same step.*

Proof. By contradiction. Assume, that there are two processes $q, q' \in \mathcal{N}_p \cup \{p\}$ such that $q' \neq q$, and both q and q' leave the alliance in some step $\gamma \mapsto \gamma'$. Consider the following two cases:

$p \in \{q, q'\}$: Without loss of generality, $p = q'$. From the guard of Action `Leave` at p, $p.choice = \perp$. Now, $p \in \mathcal{N}_q$, so from the guard of Action `Leave` at q, $p.choice = q \neq \perp$, contradiction.

$p \notin \{q, q'\}$: By definition, $p \in \mathcal{N}_q$ and $p \in \mathcal{N}_{q'}$. So, from the guard of Action `Leave` at q, we have $p.choice = q$; and from the guard of Action `Leave` at q', $p.choice = q'$, contradiction.

□

Corollary 2 *If a process p leaves A during the step $\gamma \mapsto \gamma'$, then `Fga`(p) holds at γ'.*

Proof. Assume that process p leaves A during $\gamma \mapsto \gamma'$. From the guard of Action `Leave`, we have `NbA`$(p) \geq f(p)$. By Lemma 5, no neighbor of p leaves A during $\gamma \mapsto \gamma'$. So, $p.inA =$ FALSE and `NbA`$(p) \geq f(p)$ in γ', and we are done. □

Lemma 6 *If a process p executes `Leave`, or p.choice is assigned the ID of some neighboring process during $\gamma \mapsto \gamma'$, then `NbAOk`(p) holds at γ'.*

Proof. Let X be the value of `NbA`(p) at γ.

If p executes `Leave` during $\gamma \mapsto \gamma'$, then from the guard of `Leave`, we know that $X \geq f(p)$. Moreover, as Action `Count` is disabled at p (otherwise, `Leave` is not executed because `Count` has higher priority), $p.nbA = X$ at γ. So, $p.inA =$ FALSE and $p.nbA = X \geq f(p)$ at γ', *i.e.* `NbAOk`(p) holds at γ'.

If p executes $p.choice \leftarrow q \in \mathcal{N}_p$ at $\gamma \mapsto \gamma'$, then `HasExtra`(p) holds at γ, p does not change the value of $p.inA$ during $\gamma \mapsto \gamma'$, and $p.nbA \leftarrow X$ during $\gamma \mapsto \gamma'$. Consequently, `NbAOk`(p) holds at γ'. □

Lemma 7 *For every process p, `ChoiceOk`(p) is closed under $\mathcal{MA}(f, g)$.*

Proof. By contradiction. Assume that there is a process p such that `ChoiceOk`(p) is not closed under $\mathcal{MA}(f, g)$: There exists a step $\gamma_i \mapsto \gamma_{i+1}$ where `ChoiceOk`(p) holds at γ_i, but not at γ_{i+1}. That is: $p.choice \neq \perp \wedge p.choice.inA \wedge \neg$`HasExtra`$(p)$ holds at γ_{i+1}.

Assume that the value of $p.inA$ changes between γ_i and γ_{i+1}. Then, p executes `Join` or `Leave` during $\gamma_i \mapsto \gamma_{i+1}$. In the former case, $p.choice = \perp$ at γ_{i+1}, and consequently, `ChoiceOk`(p) still holds at γ_{i+1}, contradiction. In the latter case, from the guard of `Leave`, we can deduce that $p.choice = \perp$ at γ_i and, as Action `Leave` does not modify the variable *choice*, $p.choice = \perp$ still holds at γ_{i+1}, contradiction. So, the value of $p.inA$ does not change during $\gamma_i \mapsto \gamma_{i+1}$. Consider the following two cases:

A) $p.choice = \perp$ at γ_i: $p.choice \neq \perp$ at γ_{i+1}. So, p executes Action `Vote` during $\gamma_i \mapsto \gamma_{i+1}$. Consequently, the guard of Action `Vote` holds at p at γ_i. In particular, `ChosenCand`$(p) \neq \perp$ at γ_i, and so `HasExtra`(p) also holds at γ_i. As the value of $p.inA$ does not change during $\gamma_i \mapsto \gamma_{i+1}$, a neighbor of p should leave A during $\gamma_i \mapsto \gamma_{i+1}$, so that `HasExtra`$(p)$ becomes FALSE. Since $p.choice = \perp$ at γ_i, no neighbor of p can execute Action `Leave` at $\gamma_i \mapsto \gamma_{i+1}$, contradiction.

B) $p.choice \neq \perp$ at γ_i: If p executes `Vote` at $\gamma_i \mapsto \gamma_{i+1}$, then $p.choice = \perp$ at γ_{i+1} and `ChoiceOk`(p) still holds at γ_{i+1}, contradiction. So, the value of $p.choice$ is the same at γ_i and γ_{i+1}. Let q be this value. Recall that $q \in \mathcal{N}_p$, and consider the following two subcases:

$\neg q.inA$ in γ_i: $q.inA$ holds at γ_{i+1}. So, q executes Action `Join` during $\gamma_i \mapsto \gamma_{i+1}$. Now, as $p.choice = q$ at γ_i, Action `Join` is disabled at q at γ_i, contradiction.

$q.inA$ in γ_i: Since $\texttt{ChoiceOk}(p)$ holds at γ_i, we have $\texttt{HasExtra}(p) = \text{TRUE}$ at γ_i. Now, $\texttt{HasExtra}(p)$ is FALSE at γ_{i+1}. Moreover, we already know that the value of $p.inA$ does not change during $\gamma_i \mapsto \gamma_{i+1}$. So, by Lemma 5, exactly one neighbor of p executes Action `Leave` during $\gamma_i \mapsto \gamma_{i+1}$. As $p.choice = q$ at γ_i, the neighbor that leaves A during $\gamma_i \mapsto \gamma_{i+1}$ must be q. Thus, $q.inA =$ FALSE at γ_{i+1}, and since $p.choice = q$ still holds at γ_{i+1}, we have $p.choice.inA = \text{FALSE}$ at γ_{i+1}. Consequently, $\texttt{ChoiceOk}(p)$ still holds at γ_{i+1}, contradiction.

□

Lemma 8 *For every process p, $\texttt{ChoiceOk}(p)$ holds forever after p executes any action.*

Proof. Let p be a process that executes any action during $\gamma \mapsto \gamma'$. By Lemma 7, we need only show that $\texttt{ChoiceOk}(p)$ is TRUE at either γ or γ'.

Consider the following three cases:

A) p executes `Join`. Then, $p.choice = \perp$ at γ', and consequently $\texttt{ChoiceOk}(p)$ is TRUE at γ'.

B) p executes `Vote`. Then, $p.choice = \perp$ in either γ or γ', and $\texttt{ChoiceOk}(p)$ is TRUE at γ or γ'.

C) p executes any other action. As in the previous cases, if $p.choice = \perp$ at γ, we conclude that $\texttt{ChoiceOk}(p)$ is TRUE at γ. Suppose $p.choice \neq \perp$ at γ. Since `Join` and `Vote` have higher priority than any other action, we deduce that their respective guards are FALSE at γ. In particular, from the negation of the guard of Action `Vote`, we can deduce that $p.choice = \texttt{ChosenCand}(p) \neq \perp$ at γ. So, $\texttt{HasExtra}(p)$ holds at γ, and thus $\texttt{ChoiceOk}(p)$ holds at γ. □

Lemma 9 *If $f \geq g$, $\texttt{ChoiceOk}(p) \wedge \texttt{Fga}(p)$ is closed under $\mathcal{MA}(f,g)$ for every process p.*

Proof. Let p be a process. Let $\gamma \mapsto \gamma'$ be any step such that both $\texttt{ChoiceOk}(p)$ and $\texttt{Fga}(p)$ hold at γ. By Lemma 7, we have: (*) $\texttt{ChoiceOk}(p)$ *holds at γ'.*

Hence, we need only show that $\texttt{Fga}(p)$ still holds at γ'. Let X be the value of $\texttt{NbA}(p)$ at γ. Let Y be the value of $\texttt{NbA}(p)$ at γ'. By Lemma 5, $Y \geq X - 1$. Consider the following two cases:

- A) *The value of* p.inA *is the same at γ and γ'.*

  If $p.choice = \perp$ at γ, then no neighbor of p can leave A during $\gamma \mapsto \gamma'$. Consequently, $Y \geq X$, which also implies that $\texttt{Fga}(p)$ still holds at γ'. Otherwise, $p.choice \neq \perp$ at γ. There are two cases.

  $p.choice.inA$ at γ: By (*), $X > h_A(p)$ at γ. So, as the value of $p.inA$ is the same at γ and γ', and $Y \geq X - 1$, we have $Y \geq h_A(p)$ at γ', which implies that $\texttt{Fga}(p)$ still holds at γ'.

  $\neg p.choice.inA$ at γ: There is no neighbor q of p such that $q.inA$ and $p.choice = q$ at γ. So, no neighbor of p leaves A during $\gamma \mapsto \gamma'$. Consequently, $Y \geq X$ and, as the value of $p.inA$ is the same at γ and γ', $\texttt{Fga}(p)$ still holds at γ'.

- B) *The value of* p.inA *changes during $\gamma \mapsto \gamma'$.* Consider the following two cases:

  p executes `Leave` in $\gamma \mapsto \gamma'$: Since $p.inA = \text{FALSE}$ at γ', we have that $\texttt{Fga}(p)$ holds at γ' only if $Y \geq f(p)$. Then, from the guard of Action `Leave`, we have (1) $X \geq f(p)$ and (2) $p.choice = \perp$ at γ. By (2), no neighbor of p leaves A at $\gamma \mapsto \gamma'$. Thus, $Y \geq X \geq f(p)$, which implies that $\texttt{Fga}(p)$ still holds at γ'.

  p executes `Join` during $\gamma \mapsto \gamma'$: Since $p.inA = \text{TRUE}$ at γ', we have that $\texttt{Fga}(p)$ holds at γ' only if $Y \geq g(p)$. (Recall that $f \geq g$.) Consider the following two cases:

  $X > Y$: Then $Y = X - 1$. Let q be the neighbor of p that leaves A during $\gamma \mapsto \gamma'$. $q.inA = \text{TRUE} \wedge p.choice = q$ at γ. So, by (*), $p.inA = \text{FALSE}$ at γ implies that $X > f(p)$. So, $Y \geq f(p) \geq g(p)$, which implies that $\texttt{Fga}(p)$ still holds at γ'.

$\boldsymbol{X \leq Y}$**:** Then, $Y \geq X \geq f(p) \geq g(p)$, which implies that $\texttt{Fga}(p)$ still holds at γ'.

$\square$

Lemma 10 *If $f \geq g$, then the predicate*

$$\texttt{ChoiceOk}(p) \wedge \texttt{Fga}(p) \wedge \texttt{NbAOk}(p)$$

is closed under $\mathcal{MA}(f,g)$ for every process p.

Proof. Let p be a process. Let $\gamma \mapsto \gamma'$ be any step such that $\texttt{ChoiceOk}(p) \wedge \texttt{Fga}(p) \wedge \texttt{NbAOk}(p)$ holds at γ. By Lemma 9, $\texttt{ChoiceOk}(p) \wedge \texttt{Fga}(p)$ is TRUE at γ'. Therefore, we need only show that $\texttt{NbAOk}(p)$ still holds at γ'.

Assume the contrary. Let X be the value of $\texttt{NbA}(p)$ at γ and consider the following two cases:

- **The value of $p.inA$ does not change during $\gamma \mapsto \gamma'$:** Assume that $p.inA$ is TRUE at γ. Thus, if $\texttt{NbAOk}(p)$ becomes FALSE at γ', p must modify $p.nbA$ during $\gamma \mapsto \gamma'$. From the algorithm, p executes $p.nbA \leftarrow X$ during $\gamma \mapsto \gamma'$. Then, $X \geq g(p)$ since $\texttt{Fga}(p)$ at γ. Thus, $p.inA = \text{TRUE}$ and $p.nbA \geq g(p)$ at γ', *i.e.*, $\texttt{NbAOk}(p)$ still holds at γ', contradiction.

  Assume that $p.inA$ is FALSE at γ. By similar reasoning, we obtain a contradiction in this case as well.

- **The value of $p.inA$ changes during $\gamma \mapsto \gamma'$:** There are two cases:

  p leaves A during $\gamma \mapsto \gamma'$: Then, $\texttt{NbAOk}(p)$ still holds at γ' by Lemma 6, contradiction.

  p joins A during $\gamma \mapsto \gamma'$: Then, $X \geq f(p)$, because $p.inA = \text{FALSE}$, and $\texttt{Fga}(p)$ holds at γ. Then, $p.nbA \leftarrow X$ during $\gamma \mapsto \gamma'$. So, $p.inA = \text{TRUE}$ and $p.nbA \geq f(p) \geq g(p)$ at γ', *i.e.*, $\texttt{NbAOk}(p)$ still holds at γ', contradiction.

$\square$

Lemma 11 *If $f \geq g$, then in any execution of $\mathcal{MA}(f,g)$, $J \leq 1$, that is, every process joins the (f,g)-alliance at most once.*

Proof. Our proof is by contradiction, and is illustrated by Figure 6. Assume that some process p executes Action $\texttt{Join}$ at least two times. Note that p must execute Action $\texttt{Leave}$ between two executions of Action $\texttt{Join}$. Thus, there exist $0 \leq i < j < k$ such that p joins A during $\gamma_i \mapsto \gamma_{i+1}$, leaves A during $\gamma_j \mapsto \gamma_{j+1}$, and joins A again during $\gamma_k \mapsto \gamma_{k+1}$.

From the guard of Action $\texttt{Join}$, $q.choice \neq p$ at γ_i for all $q \in \mathcal{N}_p$. From the guard of Action $\texttt{Leave}$, $q.choice = p$ at γ_j for all $q \in \mathcal{N}_p$. Thus:

(1) Every $q \in \mathcal{N}_p$ executes $q.choice \leftarrow p$ using Action $\texttt{Vote}$ before γ_j.

Let q be any neighbor of p. Let $\gamma_\ell \mapsto \gamma_{\ell+1}$ be a step during which $q.choice \leftarrow p$, using Action $\texttt{Vote}$, for $i < l < j$. Such a step exists by (1). By Lemma 8, $\texttt{ChoiceOk}(q)$ is TRUE at $\gamma_{\ell+1}$. Moreover, by (1) and the code of Action $\texttt{Vote}$, we can deduce that (a) $q.choice = \perp$ and (b) $p.inA = \text{TRUE}$ at γ_ℓ. By (a), $p.inA$ is still TRUE at $\gamma_{\ell+1}$. Now, $q.choice = p$ at $\gamma_{\ell+1}$. So, $\texttt{ChoiceOk}(q)$ at $\gamma_{\ell+1}$ implies that $\texttt{HasExtra}(q)$ holds at $\gamma_{\ell+1}$, which in turn implies that $\texttt{Fga}(q)$ holds at $\gamma_{\ell+1}$. Finally, $\texttt{NbAOk}(q)$ at $\gamma_{\ell+1}$ by Lemma 6. So, by Lemma 10, $\texttt{ChoiceOk}(q) \wedge \texttt{Fga}(q) \wedge \texttt{NbAOk}(q)$ is true forever, starting at $\gamma_{\ell+1}$. Hence:

(2) Every neighbor q of p satisfies $\texttt{ChoiceOk}(q) \wedge \texttt{Fga}(q) \wedge \texttt{NbAOk}(q)$ forever, starting at γ_j.

As p leaves A during $\gamma_j \mapsto \gamma_{j+1}$, by Corollary 2 and Lemmas 8 and 9, we have:

(3) $\texttt{ChoiceOk}(p) \wedge \texttt{Fga}(p)$ holds forever, starting at γ_{j+1}.

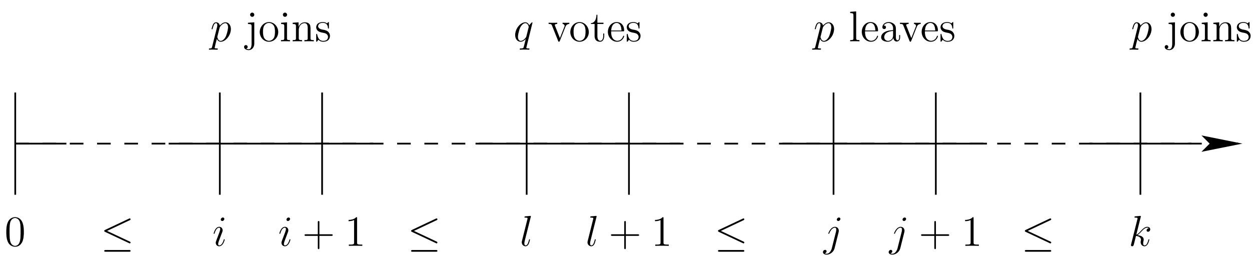

Figure 6: Execution of $\mathcal{MA}(f,g)$

As p joins A during $\gamma_k \mapsto \gamma_{k+1}$, (a) $\neg p.inA \wedge \texttt{NbA}(p) < f(p)$ or (b) $\texttt{IsMissing}(p)$ holds at γ_k. Now, (a) contradicts (3) and (b) contradicts (2). $\square$

From Lemmas 4 and 11, we have:

Corollary 3 *Starting from any configuration, if $f \geq g$, $\mathcal{MA}(f,g)$ reaches a terminal configuration in $O(n \cdot \Delta^3)$ steps.*

By Lemma 3 and Corollary 3, we have:

Theorem 1 *If $f \geq g$, $\mathcal{MA}(f,g)$ is silent and self-stabilizing* w.r.t. $\mathbb{S}_{Minimal}$, *and stabilizes within $O(\Delta^3 n)$ steps.*

4.3 Complexity Analysis and Safe Convergence

We should exhibit a set of *feasible legitimate configurations*. We choose the set of configurations γ that satisfies

$$\mathbb{S}_{\mathit{flc}} \stackrel{\texttt{def}}{=} \forall p \in V,\ \texttt{ChoiceOk}(p) \wedge \texttt{Fga}(p)$$

as the set of feasible legitimate configuration. Indeed, in such configurations, A is an (f,g)-alliance, by Remark 1.

Then, from Lemma 9, we already know that the set of *feasible legitimate configurations* is closed under $\mathcal{MA}(f,g)$ if $f \geq g$:

Corollary 4 *If $f \geq g$, then $\mathbb{S}_{\mathit{flc}}$ is closed under $\mathcal{MA}(f,g)$.*

We should also exhibit a set of *optimal legitimate configurations*. We choose the set of terminal configurations to be the set of *optimal legitimate configurations*. Indeed, the closure property is trivially realized in such configurations. Moreover, any terminal configuration satisfies $\mathbb{S}_{Minimal}$, by Lemma 3. This means, in particular, that every terminal configuration also satisfies $\mathbb{S}_{\mathit{flc}}$, *i.e.*, every optimal legitimate configurations is also a feasible legitimate configuration.

So, to establish safe convergence of $\mathcal{MA}(f,g)$, we now show that $\mathcal{MA}(f,g)$ gradually converges to more and more specific closed predicates — including the set of feasible configurations — until reaching a terminal configuration. The gradual convergence to those specific closed predicates is given in Figure 7.

Lemma 12 *For every process p, after at most one round,* $\texttt{ChoiceOk}(p)$ *is* TRUE *forever.*

Proof. It is sufficient to show that $\texttt{ChoiceOk}(p)$ becomes TRUE during the first round, by Lemma 7. If p is continuously enabled from the initial configuration, then p executes at least one action during the first round, and then by Lemma 8, we are done.

Otherwise, the first round contains a configuration γ in which every action is disabled at p. In particular, from the negation of the guard of Action `Vote`, we have $p.choice = \texttt{ChosenCand}(p)$ at γ. Two cases are then possible at γ:

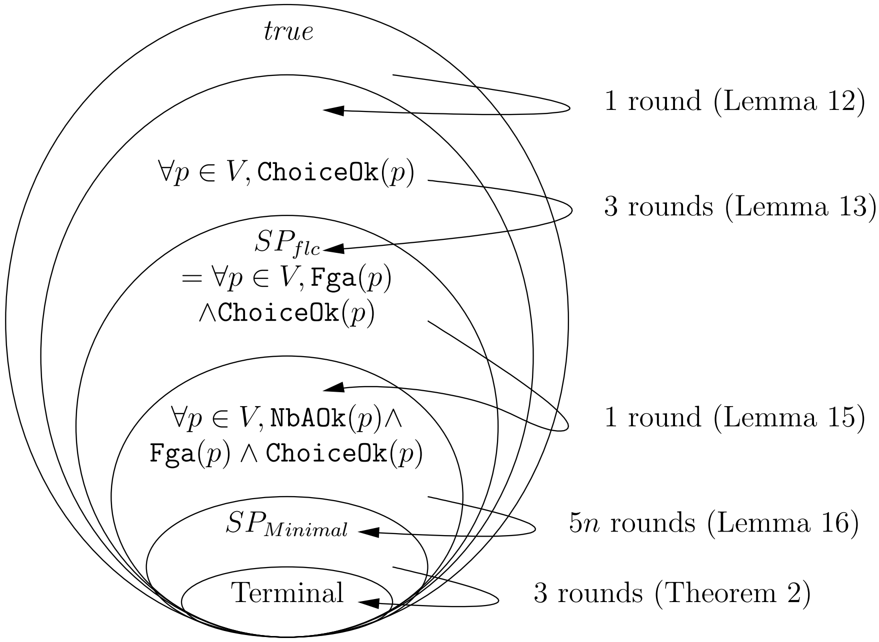

Figure 7: Safe Convergence of $\mathcal{MA}(f, g)$

***p.choice* $= \perp$:** By definition, $\texttt{ChoiceOk}(p)$ holds at γ.

***p.choice* $\neq \perp$:** Since $p.choice = \texttt{ChosenCand}(p)$, we have $p.choice = \texttt{MinCand}(p)$ at γ. Thus, $\texttt{HasExtra}(p)$ holds at γ, which implies that $\texttt{ChoiceOk}(p)$ holds at γ.

$\square$

Lemma 13 *Assume $f \geq g$. Let $\gamma_0 \ldots \gamma_i \ldots$ be an execution of $\mathcal{MA}(f, g)$. $\forall i \geq 0$, if $\texttt{ChoiceOk}(p)$ for all $p \in V$ at γ_i, then $\exists j \geq i$ such that γ_j is within three rounds of γ_i, and $\texttt{ChoiceOk}(p) \wedge \texttt{Fga}(p)$ holds at γ_j for all p.*

Proof. Let γ_{t_0} be a configuration where $\texttt{ChoiceOk}(p)$ for all p. Consider any execution $e = \gamma_{t_0} \ldots \gamma_{t_1} \ldots \gamma_{t_2} \ldots \gamma_{t_3} \ldots$, where $\gamma_{t_1}, \gamma_{t_2}$, and γ_{t_3} are the last configurations of the first, second, and third rounds of e, respectively. By Lemma 7, it is sufficient to show that there is some $t \in [t_0..t_3]$ such that $\forall p \in V, \texttt{Fga}(p)$ at γ_t. Suppose no such configuration exists. By Lemmas 7 and 9, this means that there exists a process v such that:

(1) $\forall t \in [t_0..t_3]$, $\neg\texttt{Fga}(v)$ at γ_t.

We now derive a contradiction using the following six claims.

(2) $\forall t \in [t_1..t_3]$, $v.choice = \perp$ at γ_t.

Proof of (2): By (1) $\texttt{HasExtra}(v) = \textsc{false}$ at γ_t for all $t \in [t_0..t_3]$. So, from the definition of $\texttt{ChosenCand}(v)$, we can deduce that $\forall t \in [t_0..t_3]$, if $v.choice = \perp$ at γ_t, then $\forall t' \in [t..t_3]$, $v.choice = \perp$ at $\gamma_{t'}$.

Hence, to show the claim, it is sufficient to show that $\exists t \in [t_0..t_1]$ such that $v.choice = \perp$ at γ_t. Suppose the contrary. Then, $\forall t \in [t_0..t_1]$, $v.choice \neq \perp \wedge \neg\texttt{HasExtra}(v)$ at γ_t, that is, the guard of $\texttt{Vote}$ is $\textsc{true}$ at v at γ_t. So, v executes (at least) one of the first two actions in the first round to set $v.choice$ to $\perp$, and we are done.

(3) $\forall t \in [t_1..t_3]$, $\neg v.inA \Rightarrow (\forall q \in \mathcal{N}_v, q.choice \neq v)$ at γ_t.

Proof of (3): Let $\gamma_t \mapsto \gamma_{t+1}$ such that $t \in [t_0..t_3-1]$. Assume that $\neg v.inA \Rightarrow (\forall q \in \mathcal{N}_v, q.choice \neq v)$ holds at γ_t.

If $v.inA = \text{TRUE}$ at γ_t, then $v.inA = \text{TRUE}$ at γ_{t+1} by (1) and Corollary 2. In particular, this implies that $\neg v.inA \Rightarrow (\forall q \in \mathcal{N}_v, q.choice \neq v)$ still holds at γ_{t+1}. Otherwise, $\neg v.inA \wedge (\forall q \in \mathcal{N}_v, q.choice \neq v)$ holds at γ_t and, from the definition of $\texttt{ChosenCand}(q)$, no neighbor of v can execute $\texttt{Vote}$ to designate v with its pointer during $\gamma_t \mapsto \gamma_{t+1}$. Hence, $\neg v.inA \Rightarrow (\forall q \in \mathcal{N}_v, q.choice \neq v)$ still holds at γ_{t+1}.

Consequently, $\forall t \in [t_0..t_3]$, if $\neg v.inA \Rightarrow (\forall q \in \mathcal{N}_v, q.choice \neq v)$ holds at γ_t, then $\forall t' \in [t..t_3]$, $\neg v.inA \Rightarrow (\forall q \in \mathcal{N}_v, q.choice \neq v)$ still holds at $\gamma_{t'}$. Thus, to show the claim, it suffices to show that $\exists t \in [t_0..t_1]$ such that $\neg v.inA \Rightarrow (\forall q \in \mathcal{N}_v, q.choice \neq v)$ at γ_t. Assume the contrary: $\forall t \in [t_0..t_1]$, $\neg v.inA \wedge (\exists q \in \mathcal{N}_v, q.choice = v)$ holds at γ_t. Then, $\forall q \in \mathcal{N}_v$, if $q.choice \neq v$ at γ_t with $t \in [t_0..t_1]$, then $\forall t' \in [t..t_1]$, $q.choice \neq v$ at $\gamma_{t'}$. Thus, v has a neighbor q such that $\forall t \in [t_0..t_1]$, $q.choice = v$ at γ_t. In this case, $\forall t \in [t_0..t_1]$, the guard of $\texttt{Vote}$ is TRUE at q at γ_t. So, q executes (at least) one of the first two actions in the first round to set $q.choice$ to $\perp$, contradiction.

(4) $\forall t \in [t_2..t_3]$, $v.nbA \leq \texttt{NbA}(v)$ at γ_t.

Proof of (4): By (2), no neighbor of v can leave A during the second and third rounds, that is, $\texttt{NbA}(v)$ is monotonically nondecreasing during $[t_1..t_3]$. So, $\forall t \in [t_1..t_3]$, if $v.nbA \leq \texttt{NbA}(v)$ at γ_t, then $\forall t' \in [t..t_3]$, $v.nbA \leq \texttt{NbA}(v)$ at $\gamma_{t'}$. Hence, to show the claim, it suffices to show that $\exists t \in [t_1..t_2]$ such that $v.nbA \leq \texttt{NbA}(v)$ at γ_t. Assume the contrary, namely that $v.nbA > \texttt{NbA}(v)$ at γ_t, $\forall t \in [t_1..t_2]$. Then, $\forall t \in [t_1..t_2]$, the guard of $\texttt{Count}$ is TRUE at v. Consequently, v executes one of the first three actions, in particular $v.nbA \leftarrow \texttt{NbA}(v)$, during the second round, and, as $\texttt{NbA}(p)$ is monotonically nondecreasing during $[t_1..t_3]$, we obtain a contradiction.

(5) $\forall t \in [t_2..t_3]$, $v.inA$ at γ_t.

Proof of (5): $\forall t \in [t_0..t_3]$, if $v.inA = \text{TRUE}$ at γ_t, then $\forall t' \in [t..t_3]$, $v.inA = \text{TRUE}$ at $\gamma_{t'}$ by (1) and Corollary 2. Hence, to show the claim, it is sufficient to show that there is some $t \in [t_0..t_2]$ such that $v.inA = \text{TRUE}$ at γ_t. Assume the contrary: $\forall t \in [t_0..t_2]$, $v.inA = \text{FALSE}$ at γ_t. Then, by (1), $\forall t \in [t_0..t_2]$, $\texttt{NbA}(v) < f(v)$ at γ_t. By (3), $\forall t \in [t_1..t_3]$, $\forall q \in \mathcal{N}_v, q.choice \neq v$ at γ_t. Thus, the guard of the highest priority action of v, $\texttt{Join}$, is true in particular at every γ_t such that $t \in [t_1..t_2]$. Thus v joins the alliance during the second round, contradiction.

(6) $\forall t \in [t_2..t_3]$, $\forall q \in \mathcal{N}_v, \neg q.inA \Rightarrow (\forall r \in \mathcal{N}_q, r.choice \neq q)$ at γ_t.

Proof of (6): Let q be a neighbor of v. Pick $\gamma_t \mapsto \gamma_{t+1}$ such that $t \in [t_1..t_3-1]$. Assume that $\neg q.inA \Rightarrow (\forall r \in \mathcal{N}_q, r.choice \neq q)$ holds at γ_t.

If $q.inA = \text{TRUE}$ at γ_t, then by (2), the guard of $\texttt{Leave}$ is disabled at q, so $q.inA = \text{TRUE}$ at γ_{t+1}, and consequently, $\neg q.inA \Rightarrow (\forall r \in \mathcal{N}_q, r.choice \neq q)$ still holds at γ_{t+1}. Otherwise, $\neg q.inA \wedge (\forall r \in \mathcal{N}_q, r.choice \neq q)$ holds at γ_t and, from the definition of $\texttt{ChosenCand}(r)$, no neighbor r of q can execute $\texttt{Vote}$ to designate q with its pointer during $\gamma_t \mapsto \gamma_{t+1}$. Hence, $\neg q.inA \Rightarrow (\forall r \in \mathcal{N}_q, r.choice \neq q)$ still holds at γ_{t+1}. Consequently, $\forall t \in [t_1..t_3]$, $\forall q \in \mathcal{N}_v$, if $\neg q.inA \Rightarrow (\forall r \in \mathcal{N}_q, r.choice \neq q)$ holds at γ_t, then $\forall t' \in [t..t_3]$, $\neg q.inA \Rightarrow (\forall r \in \mathcal{N}_q, r.choice \neq q)$ holds at $\gamma_{t'}$.

Hence, to show this claim, it is sufficient to show that $\forall q \in \mathcal{N}_v$, $\exists t \in [t_1..t_2]$ such that $\neg q.inA \Rightarrow (\forall r \in \mathcal{N}_q, r.choice \neq q)$ at γ_t. Assume the contrary: let q be a neighbor of v such that $\forall t \in [t_1..t_2]$, $\neg q.inA \wedge (\exists r \in \mathcal{N}_q, r.choice = q)$ holds at γ_t. We have that $\forall r \in \mathcal{N}_q$, if $r.choice \neq q$ at γ_t with $t \in [t_1..t_2]$, then $\forall t' \in [t..t_2]$, $r.choice \neq q$. Thus, there is a neighbor r of q such that $\forall t \in [t_1..t_2]$, $r.choice = q$. Then, from the definition of $\texttt{ChosenCand}(r)$, $\forall t \in [t_1..t_2]$, the guard of $\texttt{Vote}$ is *true* at r in γ_t. So, r executes (at least) one of the first two actions in the second round to set $r.choice$ to $\perp$, contradiction.

(7) $\forall q \in \mathcal{N}_v, q.inA$ at γ_{t_3}.

Proof of (7): Let q be a neighbor of v. By (2), $\forall t \in [t_2..t_3]$, $\texttt{CanLeave}(q) = \text{FALSE}$. Thus, $\forall t \in [t_2..t_3]$, if $q.inA$ at γ_t, then $\forall t' \in [t..t_3]$, $q.inA$ at $\gamma_{t'}$. Hence, to show this claim, it is sufficient to show that $\exists t \in [t_2..t_3]$

such that $q.inA$ at γ_t. Assume the contrary: $\forall t \in [t_2..t_3], \neg q.inA$. By (1) and (4), $\forall t \in [t_2..t_3]$, $\texttt{IsMissing}(q)$ holds at γ_t. Then, using (6), we deduce that the guard of the highest priority action of q, $\texttt{Join}$, is true at every configuration γ_t, $t \in [t_2..t_3]$. So, q joins the alliance in the third round, contradiction.

By (5), (7), and the fact that $\delta_v \geq g(v)$, $\texttt{Fga}(v)$ holds at γ_{t_3}, contradiction. □

By Remark 1, Lemmas 9, 12, and 13, we have the following:

Corollary 5 *If $f \geq g$, $\mathcal{MA}(f,g)$ is self-stabilizing* w.r.t. $\$_{flc}$, *and the first convergence time of $\mathcal{MA}(f,g)$ is at most four rounds.*

Lemma 14 *If $f \geq g$, then from any configuration at which* $\texttt{ChoiceOk}(p) \wedge \texttt{Fga}(p) \wedge \texttt{NbAOk}(p)$ *holds for all p, Action* $\texttt{Join}$ *is forever disabled at every process.*

Proof. Let γ by any configuration where $\texttt{ChoiceOk}(p) \wedge \texttt{Fga}(p) \wedge \texttt{NbAOk}(p)$ holds for all p. Then, $\texttt{Fga}(p)$ implies that $\neg p.inA \Rightarrow \texttt{NbA}(p) \geq f(p)$ at γ. Moreover, $(\forall q \in \mathcal{N}_p, \texttt{Fga}(q) \wedge \texttt{NbAOk}(q))$ implies $\neg\texttt{IsMissing}(p)$ at γ. So, Action $\texttt{Join}$ is disabled at every process p at γ. By Lemma 10, we are done. □

Lemma 15 *Let γ be any configuration at which* $\texttt{ChoiceOk}(p) \wedge \texttt{Fga}(p)$ *holds for all p. If $f \geq g$, then, within at most one additional round,* $\texttt{ChoiceOk}(p) \wedge \texttt{Fga}(p) \wedge \texttt{NbAOk}(p)$ *holds for all p forever.*

Proof. By Lemmas 9 and 10, it is sufficient to show that $\forall p \in V$, there is a configuration in the first round starting from γ where $\texttt{NbAOk}(p)$ holds. Let p be a process. Consider the following two cases:

- **The value of $\boldsymbol{p.inA}$ changes during the first round from $\boldsymbol{\gamma}$:** If p leaves A, then by Lemma 6, we are done. Otherwise, p executes $\texttt{Join}$ during some step $\gamma' \mapsto \gamma''$ of the round. So, $\texttt{NbA}(p) \geq f(p)$ at γ' (Lemma 9) and consequently, $p.nbA \geq f(p)$ at γ''. As $f(p) \geq g(p)$ and $p.inA = \text{TRUE}$ at γ'', we are done.

- **The value of $\boldsymbol{p.inA}$ does not change during the first round from $\boldsymbol{\gamma}$:** Assume that $\texttt{NbAOk}(p) = \text{FALSE}$ throughout the first round from γ. Then, as $\texttt{Fga}(p)$ is always TRUE (by Lemma 9) the guard of Action $\texttt{Count}$ is always TRUE during this round, and consequently p executes at least one of its first three actions in the round, in particular, $p.nbA \leftarrow \texttt{NbA}(p)$. Again, as $\texttt{Fga}(p)$ is always TRUE during the round (by Lemma 9) we obtain a contradiction, and we are done.

□

Lemma 16 *If $f \geq g$ and* $\texttt{ChoiceOk}(p) \wedge \texttt{Fga}(p) \wedge \texttt{NbAOk}(p)$ *for all p, and if A is not 1-minimal, then at least one process* **permanently** *leaves A within the next five rounds.*

Proof. By contradiction. Let γ_{t_0} be a configuration at which $\texttt{ChoiceOk}(p) \wedge \texttt{Fga}(p) \wedge \texttt{NbAOk}(p)$ for all p. Consider an execution $e = \gamma_{t_0} \ldots \gamma_{t_1} \ldots \gamma_{t_2} \ldots \gamma_{t_3} \ldots \gamma_{t_4} \ldots \gamma_{t_5} \ldots$, where γ_{t_i} is the last configuration of the i^{th} round of e. By Lemma 14, it is sufficient to show that $\exists t \in [t_0..t_5 - 1]$ such that some process leaves the alliance during $\gamma_t \mapsto \gamma_{t+1}$. Assume that no such a configuration exists.

Let $S = \{p : p.inA \wedge \texttt{NbA}(p) \geq f(p) \wedge (\forall q \in \mathcal{N}_p, \texttt{HasExtra}(q))\}$. As A is not a 1-minimal (f,g)-alliance during the five first rounds after γ_{t_0}, $S \neq \emptyset$. Moreover, as no process leaves (by hypothesis) or joins (by Lemma 14) the alliance during the first five rounds after γ_{t_0}, S is constant during these rounds. Let $p_{\min} = \min(S)$.

We derive a contradiction, using the following six claims:

(1) $\forall t \in [t_1..t_5], \forall p \in V, p.nbA = \texttt{NbA}(p)$ at γ_t.

Proof of (1): First, by hypothesis and Lemma 14, $\forall p \in V$, the value of $\texttt{NbA}(p)$ is constant during the five first rounds. So, to show the claim, it is sufficient to prove that $\forall p \in V$, $\exists t \in [t_0..t_1]$, $p.nbA = \texttt{NbA}(p)$ at γ_t. Assume the contrary: there is a process p such that $\forall t \in [t_0..t_1]$, $p.nbA \neq \texttt{NbA}(p)$ at γ_t. Then, $\forall t \in [t_0..t_1]$, the guard of $\texttt{Count}$ is TRUE at p. As Action $\texttt{Join}$ is disabled forever at p (by Lemma 14), p executes the second or third action, in particular $p.nbA \leftarrow \texttt{NbA}(p)$, during the first round, and we obtain a contradiction.

(2) $\forall t \in [t_1..t_5]$, $\texttt{IsBusy}(p_{\min}) = \text{FALSE}$ at γ_t.

Proof of (2): From (1) and the definition of $p_{\min}$.

(3) $\forall t \in [t_2..t_5]$, $p_{\min}.choice = \bot$ at γ_t.

Proof of (3): By (2) and the definition of $p_{\min}$, $\forall t \in [t_1..t_5]$, $\texttt{IamCand}(p_{\min})$ is TRUE but $\texttt{MinCand}(p_{\min}) < p_{\min}$ is FALSE at γ_t.

Thus, $\forall t \in [t_1..t_5]$, $\texttt{ChosenCand}(p_{\min}) = \bot$ at γ_t. Hence to show the claim, it is sufficient to prove that $\exists t \in [t_1..t_2]$, $p_{\min}.choice = \bot$ at γ_t. Assume the contrary: $\forall t \in [t_1..t_2]$, $p_{\min}.choice \neq \bot$ at γ_t and consequently the guard of Action `Vote` is TRUE at γ_t. Now, $\forall t \in [t_1..t_2]$, `Join` is disabled at $p_{\min}$ at γ_t by Lemma 14. So, $p_{\min}$ executes Action `Vote` during the second round, and we are done.

(4) $\forall t \in [t_2..t_5]$, $\neg p_{\min}.busy$ at γ_t.

Proof of (4): By (2), if $\exists t \in [t_1..t_5]$ such that $\neg p_{\min}.busy$ at γ_t, then $\forall t' \in [t..t_5]$, $\neg p_{\min}.busy$ at $\gamma_{t'}$. Hence to show the claim, it is sufficient to prove that $\exists t \in [t_1..t_2]$ such that $\neg p_{\min}.busy$ at γ_t. Assume the contrary: $\forall t \in [t_1..t_2]$, $p_{\min}.busy = \text{TRUE}$ at γ_t. $\forall t \in [t_1..t_2]$, `Join` and `Count` are disabled at $p_{\min}$ at γ_t, by Lemma 14 and (1). By (2), $\forall t \in [t_1..t_2]$, the guard of Action `Flag` is TRUE at $p_{\min}$ at γ_t. Consequently, $p_{\min}$ executes `Vote` or `Flag` during the second round, and we are done.

(5) $\forall t \in [t_3..t_5]$, $\forall q \in \mathcal{N}_{p_{\min}}, q.choice \in \{\bot, p_{\min}\}$ at γ_t.

Proof of (5): By (4), and the definition of $p_{\min}$, we have $\forall t \in [t_2..t_5]$, $\forall q \in \mathcal{N}_{p_{\min}}, \texttt{ChosenCand}(q) = p_{\min}$ at γ_t. Hence, to show the claim, it is sufficient to prove that $\forall q \in \mathcal{N}_{p_{\min}}$, $\exists t \in [t_2..t_3]$ such that $q.choice \in \{\bot, p_{\min}\}$ at γ_t. Assume the contrary: let q be a neighbor of $p_{\min}$, and assume that $\forall t \in [t_2..t_3]$, $q.choice \notin \{\bot, p_{\min}\}$ at γ_t. Then, the guard of Action `Vote` is TRUE at q at γ_t. Now, $\forall t \in [t_2..t_3]$, `Join` is disabled at q at γ_t, by Lemma 14. So, q executes Action `Vote` during the second round, and we are done.

(6) $\forall t \in [t_4..t_5]$, $\forall q \in \mathcal{N}_{p_{\min}}, q.choice = p_{\min}$ at γ_t.

Proof of (6): By (4) and the definition of $p_{\min}$, $\forall t \in [t_3..t_5]$, $\texttt{ChosenCand}(q) = p_{\min}$ at γ_t for every $q \in \mathcal{N}_{p_{\min}}$. Hence to show the claim, it is sufficient to prove that $\forall q \in \mathcal{N}_{p_{\min}}$, there is some $t \in [t_3..t_4]$ such that $q.choice = p_{\min}$ at γ_t. Assume the contrary: Let q be a neighbor of $p_{\min}$. Assume that $\forall t \in [t_3..t_4]$, $q.choice \neq p_{\min}$ at γ_t. Then, by (5), $\forall t \in [t_3..t_4]$, $q.choice = \bot$ at γ_t. Consequently the guard of Action `Vote` is TRUE at q at γ_t. $\forall t \in [t_3..t_4]$, `Join` is disabled at q at γ_t, by Lemma 14. So, q executes Action `Vote` during the third round, and we are done.

Starting at γ_{t_0}, Action `Join` is disabled at $p_{\min}$ forever. By (3), (4), and the definition of $p_{\min}$, $\forall t \in [t_4..t_5]$ Action `Vote` is disabled at $p_{\min}$ at γ_t. By (1), $\forall t \in [t_4..t_5]$, Action `Count` is disabled at $p_{\min}$ at γ_t. By (2) and (4), $\forall t \in [t_4..t_5]$ Action `Flag` is disabled at $p_{\min}$ at γ_t. By (3), (6), and the definition of $p_{\min}$, $\forall t \in [t_4..t_5]$, `Leave` is enabled at $p_{\min}$ at γ_t. Thus, $p_{\min}$ leaves the alliance during the fifth round, contradiction. $\square$

Theorem 2 *If $f \geq g$, $\mathcal{MA}(f,g)$ is silent and self-stabilizing* w.r.t. $\mathbb{S}_{1-Minimal}$ *and its stabilization time is at most $5n + 8$ rounds.*

Proof. By Lemmas 12 through 16, starting from any configuration, the system reaches a configuration γ from which A is a 1-minimal (f,g)-alliance and Actions `Join` and `Leave` are disabled forever at every process, within $5n + 5$ rounds. So, it remains to show that the system reaches a terminal configuration after at most three rounds from γ.

The following three claims establish the proof:

(1) After one round from γ, $\forall p \in V$, $p.nbA = \texttt{NbA}(p)$ forever.

Proof of (1): From γ, for every process p, `Join` is disabled forever and $\texttt{NbA}(p)$ is constant. So, if necessary, p fixes the value of $p.nbA$ to $\texttt{NbA}(p)$ within the next round by `Vote` or `Count`.

(2) After two rounds from γ, $\forall p \in V$, $(p.inA \Rightarrow p.busy) \wedge p.busy = \mathtt{IsBusy}(p)$ forever.

Proof of (2): When the second round from γ begins, for every process p, values of $p.inA$ and $p.nbA$ are constant. Moreover `Join` and `Count` are disabled forever at p (by hypothesis and (1)). So, if necessary, p fixes the value of $p.busy$ to $\mathtt{IsBusy}(p)$ within the next round by `Vote` or `Flag`. Hence, after two rounds from γ, $\forall p \in V$, $p.busy = \mathtt{IsBusy}(p)$ holds forever.

Finally, assume that there is a process p such that $p.inA \wedge \neg p.busy$ after two rounds from γ. Then, $p.inA \wedge \mathtt{NbA}(p) \geq f(p) \wedge \mathtt{IsExtra}(p)$. By (1), this means that

$$p.inA \wedge \mathtt{NbA}(p) \geq f(p) \wedge (\forall q \in \mathcal{N}_p, (\neg q.inA \Rightarrow \mathtt{NbA}(q) > f(q)) \wedge$$

$$(q.inA \Rightarrow \mathtt{NbA}(q) > g(q)))$$

which contradicts the fact that A is a 1-minimal (f,g)-alliance. Hence, after two rounds from γ, $\forall p \in V$, $(p.inA \Rightarrow p.busy)$ holds forever.

(3) After three rounds from γ, $\forall p \in V$, $p.choice = \bot$ forever.

Proof of (3): When the third round from γ begins, for every process p, $\mathtt{Cand}(p) = \emptyset$ forever by (2), which implies that $\mathtt{ChosenCand}(p) = \bot$ forever. Remember also that `Join` is disabled forever for every process. So, if necessary, p fixes the value of $p.choice$ to $\bot$ within the next round by `Vote`.

From the three previous claims, we can deduce that after at most three rounds from γ (that is, at most $5n + 8$ rounds from the initial configuration), the system reaches a terminal configuration where $\mathbb{S}_{Minimal}$ holds, by Lemma 3. $\square$

By Property 1, Corollary 5, Theorem 2, and the fact that every optimal legitimate configuration (*i.e.*, every terminal configuration) is also a feasible legitimate configuration (see Lemma 2), we have:

Corollary 6 *If $f \geq g$, $\mathcal{MA}(f,g)$ is silent and safely converging self-stabilizing* w.r.t. $(\mathbb{S}_{flc}, \mathbb{S}_{Minimal})$*, its first convergence time is at most four rounds, its second convergence time is at most $5n + 4$ rounds, and its stabilization time is at most $5n + 8$ rounds.*

5 Conclusion and Perspectives

We have given a silent self-stabilizing algorithm, $\mathcal{MA}(f,g)$, that computes a minimal (f,g)-alliance in an asynchronous network with unique node IDs, assuming that $f \geq g$ and every process p has a degree at least $g(p)$. $\mathcal{MA}(f,g)$ is also *safely converging*: It first converges to a (not necessarily minimal) (f,g)-alliance in at most four rounds and then continues to converge to a minimal one in at most $5n + 4$ additional rounds. We have verified correctness and time complexity of $\mathcal{MA}(f,g)$ assuming the (distributed) unfair daemon, the strongest daemon in the model. The memory requirement of $\mathcal{MA}(f,g)$ is $\Theta(\log n)$ bits per process and its stabilization time in steps is $O(n \cdot \Delta^3)$.

It would also be interesting to investigate silent, safely converging, and self-stabilizing solutions of the (f,g)-alliance problem without the constraint that $f \geq g$. Another possible extension of our work would be to find an algorithm for the problem with stabilization time $O(\mathcal{D})$ rounds. However, we believe that this would be very hard.

References

[1] Fabienne Carrier, Ajoy Kumar Datta, Stéphane Devismes, Lawrence L. Larmore, and Yvan Rivierre. Self-stabilizing (f,g)-alliances with safe convergence. In Teruo Higashino, Yoshiaki Katayama, Toshimitsu Masuzawa, Maria Potop-Butucaru, and Masafumi Yamashita, editors, *Stabilization, Safety, and Security of Distributed Systems - 15th International Symposium (SSS)*, volume 8255 of *Lecture Notes in Computer Science*, pages 61–73, Osaka, Japan, November 13-16 2013. Springer.

[2] Edsger W. Dijkstra. Self-Stabilizing Systems in Spite of Distributed Control. *Commun. ACM*, 17:643–644, 1974.

[3] Sukumar Ghosh, Arobinda Gupta, Ted Herman, and Sriram V. Pemmaraju. Fault-containing self-stabilizing algorithms. In *Proceedings of the Fifteenth Annual ACM Symposium on Principles of Distributed Computing*, pages 45–54, Philadelphia, Pennsylvania, USA, May 23-26 1996. ACM.

[4] Shlomi Dolev and Ted Herman. Superstabilizing protocols for dynamic distributed systems. *Chicago J. Theor. Comput. Sci.*, 1997, 1997.

[5] Shay Kutten and Boaz Patt-Shamir. Time-adaptive self stabilization. In James E. Burns and Hagit Attiya, editors, *Proceedings of the Sixteenth Annual ACM Symposium on Principles of Distributed Computing*, pages 149–158, Santa Barbara, California, USA, August 21-24 1997. ACM.

[6] Hirotsugu Kakugawa and Toshimitsu Masuzawa. A Self-stabilizing Minimal Dominating Set Algorithm with Safe Convergence. In *Proceedings of the 20th International Conference on Parallel and Distributed Processing*, IPDPS'06, pages 263–263, Washington, DC, USA, 2006. IEEE Computer Society.

[7] Christophe Genolini and Sébastien Tixeuil. A lower bound on dynamic k-stabilization in asynchronous systems. In *21st Symposium on Reliable Distributed Systems (SRDS)*, pages 211–221, Osaka, Japan, October 13-16 2002. IEEE Computer Society.

[8] Sayaka Kamei and Hirotsugu Kakugawa. A self-stabilizing approximation algorithm for the minimum weakly connected dominating set with safe convergence. In *Proceedings of the First International Workshop on Reliability, Availability, and Security (WRAS)*, pages 57–67, Paris, France, September 2007.

[9] Sayaka Kamei, Tomoko Izumi, and Yukiko Yamauchi. An asynchronous self-stabilizing approximation for the minimum connected dominating set with safe convergence in unit disk graphs. In Teruo Higashino, Yoshiaki Katayama, Toshimitsu Masuzawa, Maria Potop-Butucaru, and Masafumi Yamashita, editors, *Stabilization, Safety, and Security of Distributed Systems - 15th International Symposium, (SSS)*, volume 8255 of *Lecture Notes in Computer Science*, pages 251–265, Osaka, Japan, November 13-16 2013. Springer.

[10] Sayaka Kamei and Hirotsugu Kakugawa. A self-stabilizing 6-approximation for the minimum connected dominating set with safe convergence in unit disk graphs. *Theoretical Computer Science*, 428:80–90, 2012.

[11] C. Berge. *The Theory of Graphs*. Dover books on mathematics. Dover, 2001.

[12] Chung-Shou Liao and Gerard J. Chang. k-tuple domination in graphs. *Inf. Process. Lett.*, 87(1):45–50, 2003.

[13] J.M. Sigarreta and J.A. Rodríguez. On the global offensive alliance number of a graph. *Discrete Applied Mathematics*, 157(2):219 – 226, 2009.

[14] Jose Maria Sigarreta and Juan Alberto Rodríguez-Velazquez. On defensive alliances and line graphs. *Appl. Math. Lett.*, 19(12):1345–1350, 2006.

[15] Saïd Yahiaoui, Yacine Belhoul, Mohammed Haddad, and Hamamache Kheddouci. Self-stabilizing algorithms for minimal global powerful alliance sets in graphs. *Inf. Process. Lett.*, 113(10-11):365–370, 2013.

[16] Dana Angluin, James Aspnes, David Eisenstat, and Eric Ruppert. The computational power of population protocols. *Distributed Computing*, 20(4):279–304, 2007.

[17] Anupam Gupta, Bruce M. Maggs, Florian Oprea, and Michael K. Reiter. Quorum placement in networks to minimize access delays. In Marcos Kawazoe Aguilera and James Aspnes, editors, *Proceedings of the Twenty-Fourth Annual ACM Symposium on Principles of Distributed Computing (PODC)*, pages 87–96, Las Vegas, NV, USA, July 17-20 2005. ACM.

[18] Mitre Costa Dourado, Lucia Draque Penso, Dieter Rautenbach, and Jayme Luiz Szwarcfiter. The south zone: Distributed algorithms for alliances. In Xavier Défago, Franck Petit, and Vincent Villain, editors, *Stabilization, Safety, and Security of Distributed Systems - 13th International Symposium, (SSS)*, volume 6976 of *Lecture Notes in Computer Science*, pages 178–192, Grenoble, France, October 10-12 2011. Springer.

[19] Yihua Ding, James Z. Wang, and Pradip K. Srimani. Self-stabilizing minimal global offensive alliance algorithm with safe convergence in an arbitrary graph. In T. V. Gopal, Manindra Agrawal, Angsheng Li, and S. Barry Cooper, editors, *Theory and Applications of Models of Computation - 11th Annual Conference (TAMC)*, volume 8402 of *Lecture Notes in Computer Science*, pages 366–377, Chennai, India, April 11-13 2014. Springer.

[20] Pradip K. Srimani and Zhenyu Xu. Distributed protocols for defensive and offensive alliances in network graphs using self-stabilization. In *International Conference on Computing: Theory and Applications (ICCTA)*, pages 27–31, Kolkata, India, March 5-7 2007. IEEE Computer Society.

[21] Volker Turau. Linear self-stabilizing algorithms for the independent and dominating set problems using an unfair distributed scheduler. *Inf. Process. Lett.*, 103(3):88–93, 2007.

[22] Guangyuan Wang, Hua Wang, Xiaohui Tao, and Ji Zhang. A self-stabilizing algorithm for finding a minimal k-dominating set in general networks. In Yang Xiang, Mukaddim Pathan, Xiaohui Tao, and Hua Wang, editors, *Data and Knowledge Engineering*, Lecture Notes in Computer Science, pages 74–85. Springer Berlin Heidelberg, 2012.

[23] Shlomi Dolev, Mohamed G. Gouda, and Marco Schneider. Memory requirements for silent stabilization. *Acta Inf.*, 36(6):447–462, 1999.