On the Complexity of the Boulinier *et al*'s Unison Algorithm

Stéphane Devismes and Franck Petit

Verimag Research Report n^o TR-2012-9

November 16, 2012

On the Complexity of the Boulinier *et al*'s Unison Algorithm

Stéphane Devismes and Franck Petit

November 16, 2012

Abstract

In this paper, we address the unison problem. We consider the self-stabilizing algorithm proposed by Boulinier *et al*. We exhibit a bound on the step complexity of its stabilization time. In more details, the stabilization time of this algorithm is at most $2\mathcal{D}n^3 + (\alpha + 1)n^2 + (\alpha - 2\mathcal{D})n$ steps, where n is the number of processes, $\mathcal{D}$ is the diameter of the network, and α is a parameter of the algorithm.

Keywords: Asynchronous unison, self-stabilization, unfair daemon, stabilization time, step complexity.

How to cite this report:

```
@techreport {TR-2012-9,
  title = {On the Complexity of the Boulinier et al's Unison Algorithm},
  author = {Stéphane Devismes and Franck Petit },
  institution = {{Verimag} Research Report},
  number = {TR-2012-9},
  year = {}
}
```

1 Introduction

Self-stabilization [6] is a versatile property, enabling an algorithm to withstand transient faults (*e.g.* topological changes [7]) in a distributed system. A self-stabilizing algorithm, after transient faults hit the system and place it in some arbitrary global state, makes the system recover in finite time without external (*e.g.*, human) intervention.

For many applications or networking protocols, it is mandatory to have a common view of time, available for all or part of the processes. In most of asynchronous distributed systems (being dynamic or not), the lack of a global common clock requires to maintain on processes logical clocks that should be synchronized. Such a mechanism is referred to as *phase* (or *barrier*, or *logical clock*) *synchronization*, or in short, *unison*. Unison consists in the design of a protocol ensuring that all the logical clocks are in phase. The phrase "in phase" has a natural meaning in *synchronous* systems. In such systems, a global signal is assumed to increment simultaneously all clock variables. So, the logical clocks are in phase if the values of all clock variables are identical. In an asynchronous system, there is no such global signal. So, the unison requirement must be relaxed:

1. the clocks of every two neighboring processes should not differ from more than 1, and

2. each process should increment its clock by 1 infinitely often.

A unison protocol can be used to emulate algorithms, designed for synchronous environments, in asynchronous settings [1]. Another application of unison is of particular interest in self-stabilization: any self-stabilizing unison algorithm being *fair* in the sense defined in [2], it can be used to make any self-stabilizing algorithm designed for a weakly fair scheduler working under an unfair scheduler, the weakest scheduling assumption [8].

In [3, 4], Boulinier *et al* propose a unison algorithm, called here $\mathcal{SSAU}$. This algorithm has two parameters α and K, and is shown to be self-stabilizing in any anonymous network G under an unfair scheduler if and only if $\alpha \geq T_G - 2$ and $K > C_G$, where T_G and C_G are two constants related to the topology of G. Namely, T_G is the length of a longest hole in G, if G contains a cycle, 2 otherwise. C_G is the length of the maximal cycle of the shortest maximal cycle basis, if G contains a cycle, 2 otherwise.

Actually, taking $\alpha \geq T_G - 2$ ensures that $\mathcal{SSAU}$ recovers in finite time a configuration where the clocks of every two neighboring processes differ from at most one tick. Then, taking $K > C_G$ ensures that the system is never deadlocked, *i.e.*, each process increments its local clock infinitely often. By definition, T_G and C_G are bounded by n, the number of processes. Hence, taking $\alpha \geq n - 2$ and $K \geq n + 1$ makes $\mathcal{SSAU}$ self-stabilizing in any arbitrary network of n processes under any scheduler, even an unfair one.

$\mathcal{SSAU}$ is implemented using $\alpha + K$ states per process. Its *stabilization time*, *i.e.*, the maximum time to reach a legitimate configuration starting from any arbitrary configuration, is $O(n + \alpha)$ rounds.[1] Hence, by choosing $\alpha = O(n)$, $\alpha \geq n - 2$, $K = O(n)$, and $K \geq n + 1$, $\mathcal{SSAU}$ self-stabilizes in $O(n)$ rounds using $O(n)$ states per process.

Two other unison algorithms for arbitrary anonymous networks and using a bounded number of states per process have been proposed in the literature: [5, 7]. However, both need $\Omega(n^2)$ states per process. Moreover, the stabilization time of the algorithm [5] is proven in [3] to be in $O(\mathcal{D}n)$ rounds ($\mathcal{D}$ is the diameter of the network), while no round complexity is available for the one in [7]. Note also that no step complexity analysis is available for these two algorithms.[2]

Therefore, $\mathcal{SSAU}$ is currently the most efficient asynchronous unison for anonymous networks existing in the literature. $\mathcal{SSAU}$ being proven assuming an unfair daemon, its stabilization time in steps is finite. However, the step complexity of its stabilization time was, until now, left as an open question.

Note that the stabilization time in steps of a self-stabilizing algorithm is not necessarily bounded, *e.g.*, a self-stabilizing algorithm working under a weakly fair scheduler has a finite stabilization time in terms of rounds, but some rounds may not be bounded in terms of steps.

The step complexity of $\mathcal{SSAU}$ is of special interest, in particular if we use it to transform a self-stabilizing algorithm assuming a weakly fair scheduler into one working under an unfair scheduler. Indeed, the step complexity (and so the efficiency) of the transformed protocol will depend, in part, on the stabilization time in steps of $\mathcal{SSAU}$.

Here, we answer the open question on the stabilization time of $\mathcal{SSAU}$ by showing that it is less or equal to $2\mathcal{D}n^3 + (\alpha + 1)n^2 + (\alpha - 2\mathcal{D})n$ steps.

[1]The round complexity captures the execution rate of the slowest processor in any execution.
[2]A step is any transition from a configuration to another in an execution.

Roadmap. The rest of the paper is organized as follows: In the next section, we define the computational model and some basic notions used in the paper. In Section 3, we present Algorithm $\mathcal{SSAU}$. In Section 4, we propose its step complexity analysis. We make concluding remarks and perspectives in Section 4.3.

2 Preliminaries

Distributed Systems We consider distributed systems made of n processes. Each process can directly communicate with a subset of other processes, called *neighbors*. Communications are assumed to be *bidirectional*, that is, the neighboring relation is symmetric. Hence, we model a distributed system as a simple undirected connected graph $G = (V, E)$, where V is the set of processes and E is a set of edges representing (direct) communication relations. Every processor p can distinguish all its neighbors using a local labeling, but the network is otherwise not identified. All labels of p'neighbors are stored into the set $\mathcal{N}eig_p$. By abuse of language, we indifferently use p to designate the process p itself or its label in the code any other process.

We assume that communications are carried out using locally shared variables (henceforth called variables). Each process has a finite set of variables. A process can only write to its own variables, but can read its variables and that of its neighbors. A distributed algorithm is a collection of n programs, each one operating on a single process. The program of a process consists of a finite set of actions $\langle label \rangle$ :: $\langle guard \rangle \rightarrow \langle statement \rangle$. The *label* of an action in the program of p is used to identify the action in the reasoning. The *guard* of an action in the program of p is a predicate over the variables of p and its neighbors. The *statement* of an action in the program of p is a set of assignments on the variables of p. An action can be executed only if its guard evaluates to $true$. We consider the composite read/write atomicity, that is, actions are executed asynchronously but the evaluation of a guard and the execution of the corresponding statement, if executed, are done in one atomic step.

The *state* of a process is defined by the values of its variables. The *configuration* of a (distributed) system is the product of the states of all its processes. We denote by $\mathcal{C}$ the set of (possible) configurations. An action is *enabled* in a configuration γ if its guard evaluates to $true$ in γ. By extension, a process is said to be *enabled* in γ if at least one of its actions is enabled in γ. We denote by $Enabled(\gamma)$ the subset of processes that are enabled in γ.

A distributed algorithm $\mathcal{P}$ induces a binary relation, noted $\mapsto$, on $\mathcal{C}$: Let $\gamma, \gamma' \in \mathcal{C}$, $\gamma \mapsto \gamma'$ if and only if there exists $S \subseteq Enabled(\gamma)$ such that (1) $S \neq \emptyset$ and (2) γ' is the result of the atomic execution one enabled action per process of S on γ. An execution of $\mathcal{P}$ is a maximal sequence $e = \gamma_0, \ldots \gamma_i, \ldots$ such that $\forall i > 0$, $\gamma_{i-1} \mapsto \gamma_i$. By *maximal*, we mean that either e is infinite or e ends by a *terminal configuration*, where no process is enabled. Each transition from a configuration to another is called a *step* and driven by a *daemon* (or *scheduler*), that is, a daemon is a predicate over executions that defines a subset of admissible executions. We consider the most general daemon: the *(distributed) unfair daemon*, where every execution is admissible (hence, throughout this paper we will always omit the term admissible). This implies that if one or more processes are enabled in some configuration, then at least one enabled process (possibly more) execute one of its enabled action in the following transition. However, the unfair daemon can prevent forever a continuously enabled process from executing an action, unless it is the only enabled process.

Self-Stabilization Let $\mathcal{P}$ be a distributed algorithm. Let SP be a predicate over executions of $\mathcal{P}$. $\mathcal{P}$ is *self-stabilizing w.r.t.* SP if and only if there exists a non-empty subset of configurations $\mathcal{S}$ such that:
Closure. Every execution of $\mathcal{P}$ starting from any configuration of $\mathcal{S}$ (always) satisfies SP.
Convergence. Every execution of $\mathcal{P}$, starting from an arbitrary configuration, contains a configuration of $\mathcal{S}$.
The configurations of $\mathcal{S}$ are called the *legitimate configurations*. Conversely, all other configurations are said *illegitimate*.

Finite Incrementing System and Reset Let a be an integer. Denote $\overline{a}$ the unique element in $[0, K-1]$ such that $a = \overline{a} \bmod K$. The *distance* $d_K(a, b) = \inf(\overline{a-b}, \overline{b-a})$ on $[0, K-1]$. Two integers a and b are said to be *locally comparable* if and only if $d_K(a, b) \leq 1$. We then define the *local order relation* $\leq_l$ as follows: $a \leq_l b \overset{\text{def}}{\Leftrightarrow} 0 \leq \overline{b-a} \leq 1$. Let us define $\mathcal{X} = \{-\alpha, \ldots, 0, \ldots, K-1\}$, where α is a non-negative integer and $K \geq 2$. Let φ be a function from $\mathcal{X}$ to $\mathcal{X}$ defined by:

$$\varphi : x \rightarrow \begin{cases} (x+1) & \text{if } x < 0 \\ (x+1) \bmod K & \text{otherwise} \end{cases}$$

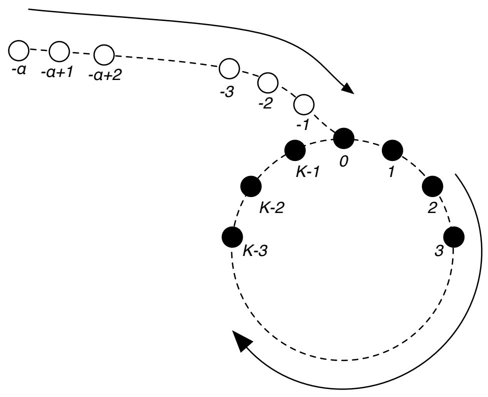


Figure 1: Finite incrementing system

Recall that $\varphi^i(x) = x$ if $i = 0$, and $\varphi^i(x) = \varphi(\varphi^{i-1}(x))$ otherwise. The pair $(\mathcal{X}, \varphi)$ is called a *finite incrementing system*, see Figure 1. The value $-\alpha$ is the *initial value* of $(\mathcal{X}, \varphi)$. A *reset* on $\mathcal{X}$ consists of an operation replacing any value of $\mathcal{X} \setminus \{-\alpha\}$ by $-\alpha$. Let $init_\varphi = \{-\alpha, \dots, 0\}$ and $stab_\varphi = \{0, \dots, K-1\}$ be the sets of *initial values* and *correct values*, respectively. The set $init^\star_\varphi$ is equal to $init_\varphi \setminus \{0\}$. We denote by $\leq_{init}$ the usual total order on $init_\varphi$.

Graph Definitions Let $G = (V, E)$ be an unoriented graph. A *path* of G is a sequence $P = p_0, \dots, p_k$ of nodes in V such that $\forall i \in [0..k[$, $(p_i, p_{i+1}) \in E$. The *length* of P is k. We denote by $\|p, q\|$ the length of the shortest path for p to q in G. The diameter of G, noted $\mathcal{D}$, is the length of the longest shortest path of G, *i.e.*, $\mathcal{D} = \max_{p,q \in V} \|p, q\|$. P is *elementary* if only if $\forall i, j \in [0..k]$, $p_i = p_j \Rightarrow i = j$. Recall that an *elementary cycle* is a path $C = c_0, \dots, c_k$ with $k > 2$ such that $c_0 = c_k$ and $c_0, \dots, c_{k-1}$ is an elementary path. A *chord* in C is a pair (c_i, c_j) such that $0 \leq i < j - 1 < k$ or $0 \leq j < i - 1 < k$ and $(c_i, c_j) \in E$. Any chordless elementary cycle is called a *hole*. Below, we recall a graph property:

Property 1 *If $p_0, p_1, \dots, p_k$ is an elementary cycle of G, then there exists a pair $i, j \in [0..k]$ such that $i < j - 1$ and $p_i, p_{i+1}, \dots, p_j, p_i$ is a hole of G.*

3 Algorithm $\mathcal{SSAU}$

The formal code of Algorithm $\mathcal{SSAU}$ is given in Algorithm 1. $\mathcal{SSAU}$ consists of a single incrementing variable $p.r$ and three mutually exclusive actions for each process p. Actually, $p.r$ is the logical clock local at p. In a legitimate configuration (where $AllCorrect$ is true at every process), a process increments its clock modulus K using action NA, when $\forall q \in \mathcal{N}eig_p : p.r \leq_l q.r$.

Algorithm 1 Algorithm $\mathcal{SSAU}(\alpha, K)$ for any process $p \in V$

```
Input: $\mathcal{N}eig_p$
Variable: $p.r \in \mathcal{X}$
Predicates:
    $C_p(x,y)$             $\equiv$   $x.r \in stab_\varphi \wedge y.r \in stab_\varphi$
    $Correct_p(q)$         $\equiv$   $C_p(p,q) \wedge d_K(p.r, q.r) \leq 1$
    $AllCorrect_p$         $\equiv$   $\forall q \in \mathcal{N}eig_p : Correct_p(q)$
    $NormalStep_p$         $\equiv$   $AllCorrect_p \wedge (\forall q \in \mathcal{N}eig_p : p.r \leq_l q.r)$
    $ConvergeStep_p$       $\equiv$   $p.r \in init^\star_\varphi \wedge$
                                      $(\forall q \in \mathcal{N}eig_p : q.r \in init_\varphi \wedge p.r \leq_{init} q.r)$
    $ResetInit_p$          $\equiv$   $\neg AllCorrect \wedge p.r \notin init_\varphi$
Actions:
    $NA$   $NormalStep_p$     ::   $p.r \leftarrow \varphi(p.r)$
    $CA$   $ConvergeStep_p$   ::   $p.r \leftarrow \varphi(p.r)$
    $RA$   $ResetInit_p$      ::   $p.r \leftarrow -\alpha$
```

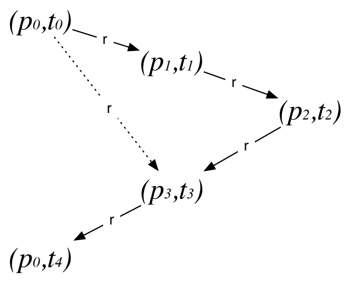


Figure 2: An helix and a stutter

When the configuration is illegitimate, the goal is to propagate a reset using Action RA so that the system re-synchronizes and eventually recovers a legitimate configuration as follows. If $AllCorrect_p$ is false but the value $p.r \notin init_\varphi$, then $p.r$ is reset to $-\alpha$ by Action RA. If either Action RA is enabled or $p.r \in init^\star_\varphi$, then p is said to be in the *reset phase*. The aim of the reset phase is to re-synchronizes p with its neighbors from any value in $init^\star_\varphi$ to 0. This is the purpose of the third action, CA.

4 Stabilization Time in Steps

In the next subsection, we define some notions used in the proofs hereafter, and prove some of their properties. In Subsection 4.2, we focus on the number of resets a process can execute. Indeed, the resets are central in the step complexity. Finally, we conclude in Subsection 4.3 by proving a bound on the stabilization time in steps.

4.1 Resets and Reset Generations

Let $e = \gamma_0\gamma_1 \ldots \gamma_k \ldots$ be an execution of $\mathcal{SSAU}$. A *reset* is a pair (p, t), where p is a process and t is a positive integer such that p executes Action RA in $\gamma_{t-1} \mapsto \gamma_t$. For every reset pair (p, t), we say that *p resets at time t*. If p resets at time t, then $p.r \notin init_\varphi$ (in particular, $p.r \neq -\alpha$) in γ_{t-1} and $p.r = -\alpha$ in γ_t.

We now give some definitions to characterize the relations between resets. Let (p_0, t_0) and (p_1, t_1) be two resets. We say that (p_0, t_0) *generates* (p_1, t_1) — denoted by $(p_0, t_0) \overset{r}{\rightsquigarrow} (p_1, t_1)$ — if and only if the following three conditions hold:

(1) $t_0 < t_1$ and $p_1 \in \mathcal{N}eig_{p_0}$,

(2) $\forall t \in [t_0..t_1 - 1]$: $p_1.r \notin init_\varphi$ in γ_t, and

(3) $p_1.r = -\alpha$ in γ_{t_1}, *i.e.*, p_1 is reset at time t_1.

Since $(p_0, t_0) \overset{r}{\rightsquigarrow} (p_1, t_1)$ implies $t_0 < t_1$, the relation $\overset{r}{\rightsquigarrow}$ defines a Directed Acyclic Graph (DAG), called $reset\ DAG$. If (p_0, t_0) is not generated by any other reset, (p_0, t_0) is said to be an *initial reset*. Below, we borrow some results from [4] (Lemmas 1 and 2).

Lemma 1 *In any execution of $\mathcal{SSAU}$, for every process p, there exists at most one positive integer t such that (p, γ_t) is an initial reset.*

A *stutter* is a path of a $reset\ DAG$ of the following form: (p_0, t_0), (p_1, t_1), (p_0, t_2). Figure 2 gives an example of stutter: (p_0, t_0), (p_3, t_3), (p_0, t_4).

Lemma 2 *A $reset\ DAG$ is without any stutter.*

A *Reset Helix* is a path (p_0, t_0), (p_1, t_1), …, (p_k, t_k) on a $reset\ DAG$ such that its projection $p_0, p_1, \ldots, p_k$ on G is an elementary cycle. An example of reset helix is given in Figure 2: (p_0, t_0), (p_1, t_1), (p_2, t_2), (p_3, t_3), (p_0, t_4). A $reset\ DAG$ is *hole-transitive* if and only if: $(p_0, t_0) \overset{r}{\rightsquigarrow} (p_1, t_1) \overset{r}{\rightsquigarrow} \ldots \overset{r}{\rightsquigarrow} (p_k, t_k)$ and $p_0, p_1, \ldots, p_k, p_0$ is a hole of G implies $(p_0, t_0) \overset{r}{\rightsquigarrow} (p_k, t_k)$. In Figure 2, we apply the notion of hole-transitivity on the hole given in Figure 3.

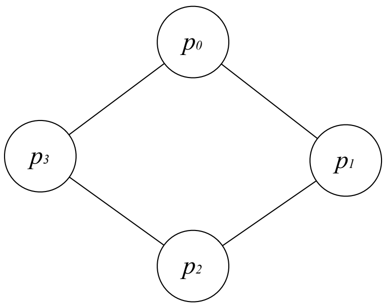
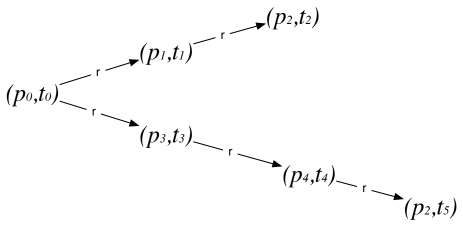


Figure 3: A hole

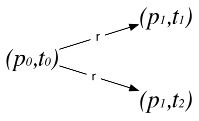
Theorem 1 *A hole-transitive $reset\ DAG$ contains no reset helix.*

Proof: Similar to the proof of Theorem 5.7 in [4]. The whole proof is given in Appendix A. □

We call *v-graph* of the $reset\ DAG\ D$, any subgraph D' of D consisting of a pair of reset paths $[(p_0,t_0),\ldots,(p_x,t_x);\ (q_0,t'_0),\ \ldots,\ (q_y,t'_y)]$ such that $(p_0,t_0) = (q_0,t'_0)$, $p_0 \neq p_x$, $p_x = q_y$, and $t_x \neq t'_y$. The size of the v-graph D' is equal to $x+y$. Note that, by definition, the size of any v-graph is at least 2. We call *small-v-graph* any v-graph of size 2, *i.e.* a v-graph of the form $[(p,t),(q,t');(p,t),(q,t'')]$. An example of v-graph of size 5 is given in Figure 4a: $[(p_0,t_0),\ (p_1,t_1),\ (p_2,t_2);\ (p_0,t_0),\ (p_3,t_3),\ (p_4,t_4),\ (p_2,t_5)]$. An example of small-v-graph is given in Figure 4b: $[(p_0,t_0),\ (p_1,t_1);\ (p_0,t_0),\ (p_1,t_2)]$.



(a) A v-graph of size 5



(b) A small v-graph

Figure 4: V-graphs

Lemma 3 *A $reset\ DAG$ is without small-v-graph.*

Proof: Consider the execution $\gamma_0,\ldots,\gamma_i,\ldots,\gamma_k$ of $\mathcal{SSAU}$ and its associated $reset\ DAG\ D$. Assume that D contains a small-v-graph $[(p,t_0),(q,t_1);(p,t_0),(q,t_2)]$. Without loss of generality, assume that $t_2 < t_1$. As $(p,t_0) \overset{r}{\rightsquigarrow} (q,t_1)$, $\forall t \in [t_0..t_1-1]$, $q.r \notin init_\varphi$ in γ_t. Now, $(p,t_0) \overset{r}{\rightsquigarrow} (q,t_2)$ implies that $q.r = -\alpha \in init_\varphi$ in γ_{t_2}. So, as $t_2 \in [t_0..t_1-1]$, we have a contradiction. □

4.2 Bound on the Number of Resets

We now show that if $\alpha \geq T_G - 2$, any process resets at most n times during any execution (Corollary 3). The main idea is to show that if $\alpha \geq T_G - 2$, any process resets at most as many time as there are initial resets (Corollary 2), the number of these latter being bounded by n (Lemma 1). To see this, we study the structure of the reset DAG. We already know that it contains no stutter (Lemma 2) and no small v-graph (Lemma 3). Then, we show that if $\alpha \geq T_G - 2$, it contains no reset helix, thank to a previous theorem (Theorem 1) and Theorem 2. Finally, we show in Lemma 7, that if $\alpha \geq T_G - 2$, the reset DAG contains no v-graph.

The following technical lemma is proven in [3]. However, we will mainly used its corollary given hereafter.

Lemma 4 *Let $(p_0,t_0),(p_1,t_1),\ldots,(p_k,t_k)$ be a path in the $reset\ DAG$ of execution $\gamma_0,\ldots\gamma_i,\ldots$ of $\mathcal{SSAU}$, where k is a positive integer. Then, $\forall t \in]t_{k-1}..t_k]$: $p_0.r \in \{\varphi^j(-\alpha), j \in [0..k-1]\}$ in configuration γ_t.*

Corollary 1 *Let $(p_0,t_0),(p_1,t_1),\ldots,(p_k,t_k)$ be a path in the $reset\ DAG$ of execution $\gamma_0,\ldots\gamma_i,\ldots$ of $\mathcal{SSAU}$, where k is a positive integer. Then, $\forall t\in[t_0..t_k]$: $p_0.r\in\{\varphi^j(-\alpha),j\in[0..k-1]\}$ in configuration γ_t.*

Proof: First, as p_0 resets at time t_0, we have $p_0.r=-\alpha=\varphi^0(-\alpha)$ in γ_{t_0}. Then, the corollary follows by applying Lemma 4 on every reset path $(p_0,t_0),(p_1,t_1),\ldots,(p_i,t_i)$ with $i\in[1..k]$. □

Theorem 2 *If $\alpha\geq T_G-2$, every $reset\ DAG$ is hole-transitive.*

Proof: Similar to the proof of Theorem 5.10 in [4]. The whole proof is given in Appendix B. □

Lemma 5 *Let $\gamma_0,\ldots,\gamma_i,\ldots,\gamma_k$ be an execution of $\mathcal{SSAU}$ and D be its associated $reset\ DAG$. If $(p_0,t_0)\overset{r}{\rightsquigarrow}(p_2,t_2)$ and (p_1,t_1) exists in D with $p_1\in\mathcal{N}eig_{p_2}$ and $t_1\in[t_0..t_2-1]$, then $(p_1,t_1)\overset{r}{\rightsquigarrow}(p_2,t_2)$ in D.*

Proof: First, by hypothesis:

(1) $t_1<t_2$ and $p_1\in\mathcal{N}eig_{p_2}$.

Then, $(p_0,t_0)\overset{r}{\rightsquigarrow}(p_2,t_2)$ implies that $\forall t\in[t_0..t_2-1]$, $p_2.r\notin init_\varphi$ in γ_t. So, as $t_1\in[t_0..t_2-1]$, we have:

(2) $\forall t\in[t_1..t_2-1]$, $p_2.r\notin init_\varphi$ in γ_t.

Finally, p_2 resets in γ_{t_2}. So:

(3) $p_2.r=-\alpha$ in γ_{t_2}.

Hence, $(p_1,t_1)\overset{r}{\rightsquigarrow}(p_2,t_2)$ in D. □

Lemma 6 *Let $(p_0,t_0),(p_1,t_1),\ldots,(p_k,t_k)$ be a path in the $reset\ DAG$ of execution $\gamma_0,\ldots\gamma_i,\ldots$ of $\mathcal{SSAU}$, where k is an integer. Let $t_z\geq t_k$. Assume that $\forall t\in[t_0..t_z]$, $\beta_t<_{init}0$, where β_t is the value of $p_0.r$ in γ_t.*

Then, $\forall t\in[t_k..t_z]$, if $\varphi^k(\beta_t)\leq_{init}0$, $p_k.r\leq_{init}\varphi^k(\beta_t)$ in γ_t.

Proof: We prove this lemma by induction on the length of the path. If $i=0$, the lemma holds by definition. Assume that the lemma is true for every path of length less or equal to i.

Consider a path $(p_0,t_0),(p_1,t_1),\ldots,(p_i,t_i),(p_{i+1},t_{i+1})$. Assume that $t_z\geq t_{i+1}$ and $\forall t\in[t_0..t_z]$, $\beta_t<_{init}0$.

By definition, $p_i.r=-\alpha$ in γ_{t_i} and $\forall t\in[t_i..t_{i+1}-1]$, $p_{i+1}.r\notin init_\varphi$. Consequently, $\forall t\in[t_i..t_{i+1}]$, $p_i.r=-\alpha$ (p_i cannot execute CA because of the value of $p_{i+1}.r$). Hence, $p_i.r=p_{i+1}.r=-\alpha$ in $\gamma_{t_{i+1}}$, and from that point,

(*) while $p_i.r\in init_\varphi^\star$, we have $p_{i+1}.r\leq_{init}\varphi(p_i.r)$ (see the guard of action CA).

So, $\forall t\in[t_i..t_z]$, if $\varphi^{i+1}(\beta_t)\leq_{init}0$, then $\varphi^i(\beta_t)<_{init}0$ and, by induction hypothesis, $p_i.r\leq_{init}\varphi^i(\beta_t)$ in γ_t and by (*) $p_{i+1}.r\leq_{init}\varphi(\varphi^i(\beta_t))$, *i.e.*, $p_{i+1}.r\leq_{init}\varphi^{i+1}(\beta_t)\leq_{init}0$, and we are done. □

Lemma 7 *If $\alpha\geq T_G-2$, every $reset\ DAG$ contains no v-graph.*

Proof: Assume that $\alpha\geq T_G-2$ and consider the execution $\gamma_0,\ldots,\gamma_i,\ldots,\gamma_k$ of $\mathcal{SSAU}$ and its associated $reset\ DAG$ D. Assume, by contradiction, that D contains a v-graph. Consider a v-graph of D, $\mathcal{V}=[(p_0,t_0),\ldots,(p_x,t_x);(q_0,t'_0),\ldots,(q_y,t'_y)]$ of minimum size. Without loss of generality, let $t_x>t'_y$.

First, $\forall i\in[0..x-1],\forall j\in[i+1..x]$, $p_i\neq p_j$ because otherwise D would contain a reset helix or a stutter, a contradiction (by Theorem 1 and 2, D contains no helix and by Lemma 2, D contains no stutter). Similarly, $\forall i\in[0..y-1],\forall j\in[i+1..y]$, $q_i\neq q_j$. Finally, $\forall i\in[1..x-1],\forall j\in[1..y-1]$, $p_i\neq q_j$ because otherwise D would contain a v-graph of size strictly smaller than $\mathcal{V}$, a contradiction. Hence, $p_0,\ldots,p_x,q_{y-1},\ldots,q_0$ is an elementary cycle of G, and consider the two following cases:

(1) *$p_0,\ldots,p_x,q_{y-1},\ldots q_0$ is a hole.* Then, $x+y\leq T_G$ and, by definition, $x>0$ and $y>0$. Consider now, the two following cases:

(a) $t'_{y-1} \geq t_{x-1}$. So, as $t'_y < t_x$, $t'_{y-1} \in [t_{x-1}..t_x - 1]$. Now, $q_y = p_x$. So, $q_{y-1} \in \mathcal{N}eig_{p_x}$ and, by Lemma 5, $(p_{x-1}, t_{x-1}) \overset{r}{\rightsquigarrow} (p_x, t_x)$ implies that $(q_{y-1}, t'_{y-1}) \overset{r}{\rightsquigarrow} (p_x, t_x)$, *i.e.*, $(q_{y-1}, t'_{y-1}) \overset{r}{\rightsquigarrow} (q_y, t_x)$. Now, $t_x \neq t'_y$. So, $[(q_{y-1}, t'_{y-1}), (q_y, t_x); (q_{y-1}, t'_{y-1}), (q_y, t'_y)]$ is a small-v-graph, a contradiction to Lemma 3.

(b) $t'_{y-1} < t_{x-1}$.
If $t'_y > t_{x-1}$, then $t_{x-1} \in]t'_{y-1}..t'_y - 1]$. Now, $q_y = p_x$. So, $p_{x-1} \in \mathcal{N}eig_{q_y}$ and, by Lemma 5, $(q_{y-1}, t'_{y-1}) \overset{r}{\rightsquigarrow} (q_y, t'_y)$ implies that $(p_{x-1}, t_{x-1}) \overset{r}{\rightsquigarrow} (q_y, t'_y)$, *i.e.*, $(p_{x-1}, t_{x-1}) \overset{r}{\rightsquigarrow} (p_x, t'_y)$. Now $t_x \neq t'_y$. So, $[(p_{x-1}, t_{x-1}), (p_x, t_x); (p_{x-1}, t_{x-1}), (p_x, t'_y)]$ is a small-v-graph in D, a contradiction to Lemma 3.
So, $t'_y \leq t_{x-1}$. If $x = 1$, then $t_{x-1} = t_0 = t'_0$ and $t'_y \leq t'_0$, a contradiction (by definition of the reset generation, $t'_y > t'_0$). So, $x \geq 2$. Then, by applying Corollary 1 on $(p_0, t_0), (p_1, t_1), \ldots, (p_{x-1}, t_{x-1})$, we obtain $\forall t \in [t_0..t_{x-1}]$, $p_0.r \in \{\varphi^j(-\alpha), j \in [0..x-2]\}$ in γ_t. Now $x - 2 < T_G - 2 \leq \alpha$. So, $\forall t \in [t_0..t_{x-1}]$, $p_0.r <_{init} 0$ in γ_t. Moreover, $x + y - 2 \leq T_G - 2 \leq \alpha$ and $t'_y \in [t_0..t_{x-1}]$. So, by Lemma 6, $q_y.r \leq_{init} 0$ in $\gamma_{t_{x-1}}$. As $q_y = p_x$, $p_x.r \leq_{init} 0$ in $\gamma_{t_{x-1}}$, *i.e.*, $p_x.r \in init_\varphi$ in $\gamma_{t_{x-1}}$. Now, this contradicts the fact that $(p_{x-1}, t_{x-1}) \overset{r}{\rightsquigarrow} (p_x, t_x)$.

(2) $p_0, \ldots, p_x, q_{y-1}, \ldots q_0$ *is not a hole.* There are three cases by Property 1:

(a) $\exists i \in [0..x-1]$, $\exists j \in [i..x]$ *such that* $p_i, \ldots, p_j, p_i$ *is a hole.* As D is hole-transitive (Theorem 2), $(p_i, t_i) \overset{r}{\rightsquigarrow} (p_j, t_j)$, and consequently, $[(p_0, t_0), \ldots, (p_i, t_i), (p_j, t_j) \ldots, (p_x, t_x); (q_0, t'_0), \ldots, (q_y, t'_y)]$ is a v-graph of size strictly smaller than $\mathcal{V}$, a contradiction.

(b) $\exists i \in [0..y-1]$, $\exists j \in [i..y]$ *such that* $q_i, \ldots, q_j, q_i$ *is a hole.* Similarly to the previous case, we obtain a contradiction.

(c) $\exists i \in [1..x-1]$, $\exists j \in [1..y-1]$ *such that* $p_i, \ldots, p_x, q_{y-1}, \ldots, q_j, p_i$ *is a hole.* Let consider the following three cases:

(i) $t_i < t'_j$. Then, $t'_j \in [t_i + 1..t_x]$. From the hypothesis, we know that q_j and p_i are neighbors. Now, $(x - i) + (y - j) + 1 \leq T_G$ and, as $y - j \geq 1$, $x - i \leq T_G - 2$. So, by applying Corollary 1 on $(p_i, t_i), \ldots, (p_x, t_x)$, we obtain $\forall t \in [t_i..t_x]$, $p_i.r \in \{\varphi^z(-\alpha), z \in [0..x-i-1]\} \subseteq init^\star_\varphi$ in γ_t. Let β be the value of $q_j.r$ in $\gamma_{t'_j - 1}$. By definition, $\beta \notin init_\varphi$. Moreover, q_j cannot execute action NA during $[t_i..t'_j - 1]$ so that $q_j.r$ takes value β because of p_i, so $\forall t \in [t_i..t'_j - 1]$, $q_j.r = \beta \notin init_\varphi$ in γ_t. Finally, $q_j.r = -\alpha$ in t'_j. So, $(p_i, t_i) \overset{r}{\rightsquigarrow} (q_j, t'_j)$, and consequently $[(p_i, t_i) \ldots, (p_x, t_x);(p_i, t_i)(q_j, t'_j), \ldots, (q_y, t'_y)]$ is a v-graph of size strictly smaller than $\mathcal{V}$, a contradiction.

(ii) $t'_j < t_i$. Similarly to the previous case, we obtain a contradiction.

(iii) $t_i = t'_j$. Then, $p_i.r = q_j.r = -\alpha$ in γ_{t_i}, and from that point, we have:

(*) while $p_i.r \in init^\star_\varphi$, $q_j.r \leq_{init} \varphi(p_i.r)$ because p_i and q_j are neighbors (see the guard of action CA).

Moreover, we have:

(**) $(x - i) + (y - j) + 1 \leq T_G$ with $(x - i) \geq 1$ and $(y - j) \geq 1$.

Then, as $(p_{x-1}, t_{x-1}) \overset{r}{\rightsquigarrow} (p_x, t_x)$ and $p_x = q_y$, we have $\forall t \in [t_{x-1}..t_x - 1] : q_y.r \notin init_\varphi$ in γ_t, which in particular means that $q_y.r \neq -\alpha$ in γ_t. Now, q_y resets at time t'_y with $t'_y < t_x$ (by hypothesis). So, $q_y.r = -\alpha$ in $\gamma_{t'_y}$ with $t'_y < t_x$, and consequently:

(***) $t'_y < t_{x-1}$.

Then, by applying Corollary 1 on $(p_i, t_i) \ldots, (p_{x-1}, t_{x-1})$, we obtain $\forall t \in [t_i..t_{x-1}] : p_i.r \in \{\varphi^z(-\alpha), z \in [0..x-2-i]$ in configuration γ_t. By (**), $x-2-i < T_G-2$ and, consequently $\forall t \in [t_i..t_{x-1}] : p_i.r \in init^\star_\varphi$ in configuration γ_t. Then, by (*) $\forall t \in [t_i..t_{x-1}] : q_j.r \in \{\varphi^z(-\alpha), z \in [0..x-1-i]\}$ in configuration γ_t. By (**) again, $x - 1 - i < T_G - 2$ and, consequently $\forall t \in [t_i..t_{x-1}] : q_j.r \in init^\star_\varphi$ in configuration γ_t. By (**) again, $x - 1 - i + (y - j) \leq T_G - 2$ and, as $t_i = t'_j$ (by hypothesis) and $t'_y < t_{x-1}$ (by (***)), we can apply Lemma 6, $\forall t \in [t'_y..t_{x-1}] : q_y.r \in \{\varphi^z(-\alpha), z \in [0..x-1-i+(y-j)]$ in configuration γ_t, *i.e.*, $\forall t \in [t'_y..t_{x-1}] : q_y.r \in init_\varphi$. In particular, $q_y.r \in init_\varphi$ in $\gamma_{t_{x-1}}$. Now, as $(p_{x-1}, t_{x-1}) \overset{r}{\rightsquigarrow} (p_x, t_x)$ and $p_x = q_y$, we should have, in particular, $q_y.r \notin init_\varphi$ in $\gamma_{t_{x-1}}$, a contradiction.

□

Corollary 2 *If $\alpha \geq T_G - 2$, then for every reset DAG D, for every initial reset (p, t) in D, and for every process q, there exists at most one non-negative integer t' such that there is a reset path from (p, t) to (q, t') in D.*

Proof: This a consequence of the fact that D contains no helix (by Theorem 2, D is hole-transitive, and so contains no reset helix by Theorem 1), no stutter (Lemma 2), and no v-graph (Lemma 7). □

Corollary 3 *If $\alpha \geq T_G - 2$, every process executes Action RA at most n times.*

Proof: By Corollary 2, a process executes Action RA at most as many times as there are initial resets. Now, there are at most n initial resets by Lemma 1. □

4.3 Stabilization Time in Steps

We now bound the number of Actions CA and NA each process executes before the system reaches a legitimate configuration. These bounds are related to the number of resets the process executes, as shown below.

Corollary 4 *If $\alpha \geq T_G - 2$, every process executes Action CA at most $(n + 1)\alpha$ times.*

Proof: Consider any process p. If $p.r <_{init} 0$ initially, then p executes CA at most α times so that $p.r$ becomes equal to 0. Then, p executes CA α times after each reset. Hence, the corollary follows from Corollary 3. □

Let $\gamma_0, \ldots, \gamma_k$ be an execution of Algorithm $\mathcal{SSAU}$. We said that a process p *recovers* at time t if and only if $p.r \in init^\star_\varphi$ in γ_{t-1} and $p.r \in stab_\varphi$ in γ_t. Clearly, any process that recovers at time t executed CA during $\gamma_{t-1} \to \gamma_t$ to switches $p.r$ from -1 to 0.

Lemma 8 *Let $\gamma_0, \ldots, \gamma_k$ be an execution of Algorithm $\mathcal{SSAU}$. $\forall i, j \in [0..k]$, with $j > i$, if γ_j is illegitimate and all processes neither reset nor recover during $\gamma_i, \ldots, \gamma_j$, then at most $(n-1)2\mathcal{D}$ Actions NA are executed during $\gamma_i, \ldots, \gamma_j$.*

Proof: First, $\forall t \in [i..j]$, γ_t is illegitimate. Then, let $Correct$ be the set of processes p, such that $AllCorrect_p = true$ in γ_i. Let $Incorrect = V \setminus Correct$. Clearly, these two sets are invariant during $\gamma_i, \ldots, \gamma_j$ and only processes in $Correct$ can execute Action NA during $\gamma_i, \ldots, \gamma_j$. Note that $|Correct| \leq n-1$. Let p be a process in $Correct$. Let $DistIncorrect(p)$ be the distance between p and to the nearest element of $Incorrect$. To prove the lemma, we show by induction on $DistIncorrect(p)$ that p can execute Action NA at most $2 \times DistIncorrect(p)$ times during $\gamma_i, \ldots, \gamma_j$.

Let p be a process of $Correct$ such that $DistIncorrect(p) = 1$. Then, a neighbor q of p is in $Incorrect$ and either $q.r$ is always (strictly) negative during $\gamma_i, \ldots, \gamma_j$ or $q.r$ is set to some fixed value $\beta \in stab_\varphi$ (and Action RA is enabled at q) during $\gamma_i, \ldots, \gamma_j$. In the former case, p cannot execute NA during $\gamma_i, \ldots, \gamma_j$. In the latter case, p satisfies $AllCorrect_p$, $d_K(p.r, q.r) \leq 1$ and consequently, p can execute NA at most 2 times during $\gamma_i, \ldots, \gamma_j$.

Assume now that any process p such that $DistIncorrect(p) = k$ can execute NA at most $2k$ times during $\gamma_i, \ldots, \gamma_j$.

Let q be a process such that $DistIncorrect(p) = k + 1$. Then, there is a neighbor q' of q such that $DistIncorrect(q) = k$ and, by induction hypothesis, q' can execute NA at most $2k$ times during $\gamma_i, \ldots, \gamma_j$. Now, through out $\gamma_i, \ldots, \gamma_j$, p always satisfies $AllCorrect_p$ and consequently $d_K(q.r, q'.r) \leq 1$ always. So, q can execute only 2 times more than q', *i.e.*, $2k + 2 = 2(k + 1)$, and the induction holds. □

A process p can recover only once per reset and one more time if $p.r \in init^\star_\varphi$ initially. Hence, following Corollary 3 and Lemma 8, we have the next corollary:

Corollary 5 *If $\alpha \geq T_G - 2$, at most $(n^3 - n)2\mathcal{D}$ Actions NA are executed before the system reaches a legitimate configuration, where every process p satisfies $AllCorrect_p$.*

From Corollaries 3-5, follows:

Theorem 3 *If $\alpha \geq T_G - 2$, Algorithm $\mathcal{SSAU}$ reaches in at most $2\mathcal{D}n^3 + (\alpha+1)n^2 + (\alpha - 2\mathcal{D})n$ steps a legitimate configuration, where every process p satisfies $AllCorrect_p$.*

Corollary 6 *If $\alpha \geq T_G - 2$ and $K > C_G$, Algorithm $\mathcal{SSAU}$ is a self-stabilizing unison algorithm under an unfair daemon and its stabilization time is at most $2\mathcal{D}n^3 + (\alpha + 1)n^2 + (\alpha - 2\mathcal{D})n$ steps.*

sectionConclusion

$\mathcal{SSAU}$ is both simple (one variable and three actions per process) and efficient. The analysis of its stabilization time in steps allows a better understanding of its internal mechanics. More precisely, we showed that the stabilization time of $\mathcal{SSAU}$ is at most $2\mathcal{D}n^3 + (\alpha + 1)n^2 + (\alpha - 2\mathcal{D})n$ steps, provided that $\alpha \geq T_G - 2$ and $K > C_G$. Hence, by choosing $\alpha = O(n)$, $\alpha \geq n - 2$, $K = O(n)$, and $K \geq n + 1$, $\mathcal{SSAU}$ self-stabilizes in $O(\mathcal{D}n^3)$ steps in any arbitrary network.

An immediate perspective of this work would be to see if the bounds can be refined, or otherwise exhibiting a worst case that matches these bounds.

Finally, it is worth investigating if it is possible to implement a self-stabilizing unison in our settings (*i.e.*, asynchronous and anonymous environment, bounded memory per process, and arbitrary topology), whose stabilization time is in $O(\mathcal{D})$ rounds.

Acknowledgement: We are grateful to Antoine Gerbaud for his useful comments and advice.

References

[1] Baruch Awerbuch. Complexity of network synchronization. *J. ACM*, 32(4):804–823, 1985. 1

[2] Joffroy Beauquier, Maria Gradinariu, and Colette Johnen. Cross-over composition - enforcement of fairness under unfair adversary. In *WSS*, pages 19–34, 2001. 1

[3] Christian Boulinier. *L'Unisson*. PhD thesis, Université de Picardie Jules Verne, Amiens, October 2007. Dissertation in french. 1, 4.2

[4] Christian Boulinier, Franck Petit, and Vincent Villain. When graph theory helps self-stabilization. In *PODC*, pages 150–159, 2004. 1, 4.1, 4.1, 4.2

[5] Jean-Michel Couvreur, Nissim Francez, and Mohamed G. Gouda. Asynchronous unison (extended abstract). In *ICDCS*, pages 486–493, 1992. 1

[6] Edsger W. Dijkstra. Self-stabilizing systems in spite of distributed control. *Commun. ACM*, 17:643–644, 1974. 1

[7] Shlomi Dolev. *Self-stabilization*. MIT Press, 2000. 1, 1

[8] Adrian Kosowski and Lukasz Kuszner. Energy optimisation in resilient self-stabilizing processes. In *symposium on Parallel Computing in Electrical Engineering*, pages 105–110, 2006. 1

A Proof of Theorem 1

Assume, by contradiction, the existence of a reset helix on a hole-transitive $reset\ DAG$.

Consider a reset helix $(p_0, t_0), (p_1, t_1), \ldots, (p_k, t_k)$ of minimum length. By definition, $p_0, p_1, \ldots, p_k$ is cycle of G (in particular, $p_0 = p_k$). Consider the two following cases:

1. $p_0, p_1, \ldots, p_k$ *is not a hole.* Then, there exists $i, j \in [0..k]$, such that $i < j - 1$ and $p_i, p_{i+1}, \ldots, p_j, p_i$ is a hole of G by Property 1. By hole-transitivity, $(p_i, t_i) \overset{r}{\rightsquigarrow} (p_j, t_j)$. Consequently, $(p_0, t_0), (p_1, t_1), \ldots, (p_i, t_i)(p_j, t_j), \ldots, (p_k, t_k)$ is a reset helix of length strictly smaller than $(p_0, t_0), (p_1, t_1), \ldots, (p_k, t_k)$, a contradiction.

2. $p_0, p_1, \ldots, p_k$ *is a hole.* By hole-transitivity, $(p_0, t_0) \overset{r}{\rightsquigarrow} (p_{k-1}, t_{k-1})$. Now, by definition of the reset helix, $(p_{k-1}, t_{k-1}) \overset{r}{\rightsquigarrow} (p_k, t_k)$ and $p_k = p_0$. Hence, there exists a stutter: (p_0, t_0), (p_{k-1}, t_{k-1}), (p_0, t_k), which contradicts Lemma 2.

So, every hole-transitive $reset\ DAG$ contains no reset helix. $\square$

B Proof of Theorem 2

If G is a tree, then by definition, any $reset\ DAG$ is hole-transitive.

So, assume that G is not a tree. Let $(p_0, t_0), (p_1, t_1), \ldots, (p_k, t_k)$ be a path in the $reset\ DAG$ of execution $\gamma_0, \ldots \gamma_i, \ldots$ of $\mathcal{SSAU}$ such that $p_0, p_1, \ldots, p_k, p_0$ is a hole of G. First, by definition, we have:

(1) $t_0 < t_k$ and $p_k \in \mathcal{N}eig_{p_0}$.

Then, let β be the value of $p_k.r$ in configuration $\gamma_{t_{k-1}}$. Since, $(p_{k-1}, t_{k-1}) \overset{r}{\rightsquigarrow} (p_k, t_k)$, $\forall t \in [t_{k-1}..t_k - 1]$, $p_k.r \notin init_\varphi$ in γ_t, and so $\beta > 0$.

Now, since $k < T_G$ and $k \geq 2$, we have: $0 \leq k - 2 < T_G - 2$. Then, by applying Corollary 1 on $(p_0, t_0), (p_1, t_1), \ldots, (p_{k-1}, t_{k-1})$, we obtain $\forall t \in [t_0..t_{k-1}]$, $p_0.r \in \{\varphi^j(-\alpha), j \in [0..k-2]\} \subseteq init^\star_\varphi$ in γ_t and consequently p_k, which is neighbor of p_0, cannot execute NA during that period to take value β. So, $\forall t \in [t_0..t_{k-1}]$, $p_k.r = \beta \notin init_\varphi$ in γ_t. So,

(2) $\forall t \in [t_0..t_k - 1]$, $p_k.r \notin init_\varphi$ in γ_t.

Finally, by hypothesis:

(3) p_k resets at time t_k, so $p_k.r = -\alpha$ in γ_{t_k}.

Hence, $(p_0, t_0) \overset{r}{\rightsquigarrow} (p_k, t_k)$, which implies that G is hole-transitive. $\square$