# Schéma général auto-stabilisant et silencieux de constructions de type arbres couvrants

Stéphane Devismes, David Ilcinkas, Colette Johnen

# *Schéma général auto-stabilisant et silencieux de constructions de type arbres couvrants* †

Stéphane Devismes[1], David Ilcinkas[2] et Colette Johnen[2]

[1] *Université Grenoble Alpes, VERIMAG UMR 514, Grenoble, France*
[2] *CNRS & Univ. Bordeaux, LaBRI, UMR 5800, F-33400 Talence, France*

Nous proposons un schéma général, appelé Scheme, qui calcule des structures de données de type arbres couvrants dans des réseaux quelconques. Scheme est auto-stabilisant, silencieux et malgré sa généralité, est aussi efficace. Il est écrit dans le modèle à mémoires localement partagées avec atomicité composite, et suppose un démon distribué inéquitable, l'hypothèse la plus faible concernant l'ordonnancement dans ce modèle. Son temps de stabilisation est d'au plus $4n_{\texttt{maxCC}}$ rondes, où $n_{\texttt{maxCC}}$ est le nombre maximum de processus dans une composante connexe. Nous montrons également des bornes supérieures polynomiales sur le temps de stabilisation en nombre de pas et de mouvements pour de grandes classes d'instances de l'algorithme Scheme. Nous illustrons la souplesse de notre approche en décrivant de telles instances résolvant des problèmes classiques tels que l'élection de leader et la construction d'arbres couvrants.

**Mots-clefs :** Algorithmes distribués, auto-stabilisation, arbre couvrant, élection de leader, plus courts chemins.

## 1. Introduction

A *self-stabilizing algorithm* is able to recover a correct behavior (defined by a set of *legitimate* configurations) in finite time, regardless of the *arbitrary* initial configuration of the system, and therefore also after a finite number of transient faults. Among the vast self-stabilizing literature, many works (see [Gär03] for a survey) focus on *spanning-tree-like constructions*, *i.e.* constructions of specific distributed spanning tree- or forest- shaped data structures. Most of these constructions achieve an additional property called *silence*: a silent self-stabilizing algorithm converges within finite time to a configuration from which the values of the communication registers used by the algorithm remain fixed. Such a configuration is called *terminal*. Silence is a desirable property, as it facilitates composition of different algorithms and may utilize less communication operations and communication bandwidth. We consider the locally shared memory model with composite atomicity, which is the most commonly used model in self-stabilization. In this model, $n$ processes communicate according to a given communication network using a finite number of locally shared registers, called *variables*. Each process can read its own variables and those of its neighbors, but can write only to its own variables. In this model, executions proceed in atomic *steps* and the asynchrony of the system is captured by the notion of *daemon*. The weakest (*i.e.*, the most general) daemon is the *distributed unfair daemon*, meaning that while the configuration is not terminal, the daemon should select at least one enabled process, maybe more. Hence, solutions stabilizing under such an assumption are highly desirable, because they work under any other daemon assumption. Moreover, under an unfair daemon, the *stabilization time* can also be bounded in terms of steps (and moves, *i.e.*, local state updates), which capture the execution time according to the fastest process, and not only in terms of *rounds*, which capture the execution time according to the slowest process. Note that if the number of moves (and thus steps) is unbounded, this means that there are processes whose moves do not make the system progress, hence wasting resources. There are many self-stabilizing algorithms proven under the distributed unfair daemon. However, analyses of the stabilization time in steps (or moves) is rather unusual and this may be an important issue. Indeed, recently, several self-stabilizing algorithms which work under a distributed unfair daemon have been shown to have an exponential stabilization time in steps in the worst case [DJ16, ACD+17].

† This study was partially supported by the ANR project DESCARTES: ANR-16-CE40-0023 and ANR project ESTATE: ANR-16-CE25-0009-03. A complete version of this work can be found in the technical report https://hal.archives-ouvertes.fr/hal-01667863.

**Contribution.** We propose a general scheme, called Algorithm Scheme, to compute spanning-tree-like data structures on bidirectional weighted networks of arbitrary (not necessarily connected) topology. Scheme is self-stabilizing and silent. It is written in the locally shared memory model with composite atomicity, assuming the distributed unfair daemon. Despite its versatility, Scheme is efficient. Indeed, its stabilization time is at most $4n_{\texttt{maxCC}}$ rounds, where $n_{\texttt{maxCC}}$ is the maximum number of processes in a connected component. Moreover, its stabilization time in moves is polynomial in usual cases. Precisely, we exhibit polynomial upper bounds on its stabilization time in moves that depend on the particular problems we consider. To illustrate the versatility of our approach, we propose instantiations of Scheme solving classical spanning-tree-like problems. Assuming an input set of roots but no identifiers, we propose two instantiations to compute a spanning forest of unconstrained, resp. shortest-path, trees, with non-rooted components detection. ‡ The first instantiation stabilizes in $O(n_{\texttt{maxCC}}n)$ moves, which matches the best known step complexity for spanning tree construction [Cou09] with explicit parent pointers. The second instantiation stabilizes in $O(\mathbb{W}_{\max}{n_{\texttt{maxCC}}}^3 n)$ moves ($\mathbb{W}_{\max}$ is the maximum weight of an edge). This move complexity also matches the best known move complexity for this problem [DIJ16]. Then, assuming the network is identified (*i.e.*, processes have distinct IDs), we propose two instantiations of Scheme for electing a leader in each connected component and building a spanning tree rooted at each leader. The first instantiation stabilizes in $O({n_{\texttt{maxCC}}}^2 n)$ moves, matching the best known step complexity for leader election [ACD+17]. The second instantation stabilizes in $O({n_{\texttt{maxCC}}}^3 n)$ moves but the built spanning tree is a Breadth first search tree. From these various examples, one can easily derive other silent self-stabilizing spanning-tree-like constructions.

## 2. Algorithm Scheme

**Algorithm 1:** Algorithm Scheme, code for any process $u$

**Inputs**

- $canBeRoot_u$: a boolean value; it is true if $u$ can be a root
- $pname_u$: name of $u$, that belongs to $IDs = \mathbb{N} \cup \{\perp\}$

**Variables**

- $st_u \in \{I, C, EB, EF\}$: the status of $u$
- $parent_u \in \{\perp\} \cup Lbl$: a pointer to a neighbor or $\perp$
- $d_u$: the distance value associated to $u$

**Predicates**

- $P\_root(u) \equiv canBeRoot_u \wedge st_u = C \wedge parent_u = \perp \wedge d_u = distRoot(u)$
- $P\_abnormalRoot(u) \equiv \neg P\_root(u) \wedge st_u \neq I \wedge$
  $[parent_u \notin \Gamma(u) \vee st_{parent_u} = I \vee d_u \prec d_{parent_u} \oplus \omega_u(parent_u) \vee (st_u \neq st_{parent_u} \wedge st_{parent_u} \neq EB)]$
- $P\_reset(u) \equiv st_u = EF \wedge P\_abnormalRoot(u)$
- $P\_updateNode(u) \equiv (\exists v \in \Gamma(u) \mid st_v = C \wedge d_v \oplus \omega_u(v) \prec d_u)$
- $P\_updateRoot(u) \equiv canBeRoot_u \wedge distRoot(u) \prec d_u$
- $P\_nodeImp(u)$ is problem dependent. If $P\_nodeImp(u)$, then $P\_updateNode(u) \vee P\_updateRoot(u)$; if $P\_updateRoot(u)$ then $P\_nodeImp(u)$. $P\_nodeImp(u)$ only depends on the values of $st_u$, $d_u$, $P\_updateRoot(u)$, and $\min_{(v \in \Gamma(u) \wedge st_v = C)}(d_v \oplus \omega_u(v))$.

**Functions**

- $beRoot(u)$: $st_u := C$; $parent_u := \perp$; $d_u := distRoot(u)$;
- $computePath(u)$: $st_u := C$; $parent_u := \text{argmin}_{(v \in \Gamma(u) \wedge st_v = C)}(d_v \oplus \omega_u(v))$; $d_u := d_{parent_u} \oplus \omega_u(parent_u)$; **if** $P\_updateRoot(u)$ **then** $beRoot(u)$;
- $Children(u) : \{v \in \Gamma(u) \mid st_u \neq I \wedge st_v \neq I \wedge parent_v = u \wedge d_v \succeq d_u \oplus \omega_v(u) \wedge (st_u = st_v \vee st_u = EB)\}$.

**Rules**

| | | | |
|---|---|---|---|
| $\mathbf{R_U}(u)$: | $st_u = C \wedge P\_nodeImp(u)$ | $\rightarrow$ | $computePath(u)$; |
| $\mathbf{R_{EB}}(u)$: | $st_u = C \wedge \neg P\_nodeImp(u) \wedge (P\_abnormalRoot(u) \vee st_{parent_u} = EB)$ | $\rightarrow$ | $st_u := EB$; |
| $\mathbf{R_{EF}}(u)$: | $st_u = EB \wedge (\forall v \in Children(u) \mid st_v = EF)$ | $\rightarrow$ | $st_u := EF$; |
| $\mathbf{R_I}(u)$: | $P\_reset(u) \wedge \neg canBeRoot_u \wedge (\forall v \in \Gamma(u) \mid st_v \neq C)$ | $\rightarrow$ | $st_u := I$; |
| $\mathbf{R_R}(u)$: | $(P\_reset(u) \vee st_u = I) \wedge [canBeRoot_u \vee (\exists v \in \Gamma(u) \mid st_v = C)]$ | $\rightarrow$ | $computePath(u)$; |

‡. By non-rooted components detection, we mean that every process in a connected component that does not contain the root should eventually take a special state notifying that it detects the absence of a root.

According to the specific problem we consider, we may want to minimize the weight of the trees using some kind of distance. So, we assume that each edge $\{u,v\}$ has two *weights*: $\omega_u(v)$ denotes the weight of the arc $(u,v)$ and $\omega_v(u)$ denotes the weight of the arc $(v,u)$. Both values belong to the domain *DistSet*. Let $(DistSet, \oplus, \prec)$ be an ordered magma, *i.e.*, $\oplus$ is a closed binary operation on *DistSet* and $\prec$ is a total order on this set. The definition of $(DistSet, \oplus, \prec)$ is problem dependent. We assume that, for every edge $\{u,v\}$ of $E$ and for every value $d$ of *DistSet*, we have $d \prec d \oplus \omega_u(v)$ and $d \prec d \oplus \omega_v(u)$.

The silent self-stabilizing algorithm Scheme (see Algorithm 1 for its code), converges to a terminal configuration where a specified spanning forest (maybe a single spanning tree) is distributedly defined. Each process $u$ uses as input a name $pname_u$ ($pname_u = \perp$, for every process $u$ if the network is anonymous), a constant boolean value $canBeRoot_u$, which is true if $u$ is allowed to be root of a tree, and in this latter case a problem dependent distance $distRoot(u)$, used when $u$ is a root. Our scheme also uses a problem dependent predicate $P\_nodeImp(u)$, with specific properties, that indicates to $u$ whether its current estimated distance to the root (variable $d_u$) can be improved (decreased). Then, a legitimate configuration is defined as follows.

**Definition 1 (Legitimate configuration)** *A* legitimate configuration *of* Scheme *is a configuration where every process u is in a legitimate state,* i.e.*, u satisfies* $\neg P\_nodeImp(u)$ *and one of the following conditions:*

1. $P\_root(u)$*;*
2. *there is a process satisfying canBeRoot in the connected component* $V_u$ *containing u,* $st_u = C$ *(for* Correct*), and* $u \in Children(parent_u)$*;*
3. *there is no process satisfying canBeRoot in* $V_u$ *and* $st_u = I$ *(for* Isolated*).*

In any given configuration, every process $u$ satisfies exactly one of the following cases: (1) $u$ is isolated, *i.e.* it has status $I$; (2) $u$ is a normal root, *i.e.*, $P\_root(u)$ holds; (3) $u$ points to some neighbor and the state of $u$ is coherent *w.r.t.* the state of its parent, *i.e.*, $u \in Children(parent_u)$; (4) $u$ is an abnormal root, *i.e.*, $P\_abnormalRoot(u)$ holds. In that latter case, we want to correct the state of $u$ while avoiding the following situation: $u$ leaves its abnormal tree $T$; this removal creates some new abnormal trees, each of those being rooted at a previous child of $u$; and later $u$ joins one of those (created) abnormal trees. (This issue is sometimes referred to as the count-to-infinity problem.) Hence, the idea is to freeze $T$, before removing any node from it. This is done as in a "Propagation of Information with Feedback": From an abnormal root, the status $EB$, for *Error Broadcast*, is broadcast down in the tree using rule $\mathbf{R_{EB}}$. Then, once the $EB$-wave reaches a leaf, the leaf initiates a convergecast $EF$-wave (*Error Feedback*) using rule $\mathbf{R_{EF}}$. Once the abnormal root gets status $EF$, the tree is frozen and can be safely deleted from its abnormal root toward its leaves. At this point, an abnormal root $u$ can either become the root of a new normal tree or join another tree, via rule $\mathbf{R_R}(u)$, depending on which option gives it the smaller distance, or $u$ becomes isolated via rule $\mathbf{R_I}(u)$ if $\neg canBeRoot_u$ and $u$ has no neighbor with status $C$. In parallel, rules $\mathbf{R_U}$ are executed to reduce the weight of the trees, if necessary, *i.e.*, if $P\_nodeImp(u)$ holds. A detailed analysis of our algorithm allows us to prove the following result.

**Theorem 1** *Any terminal configuration of Algorithm* Scheme *is legitimate, and vice versa. Moreover, Algorithm* Scheme *is silent self-stabilizing under the distributed unfair daemon, has a bounded move (and step) complexity, and stabilizes in at most* $4n_{maxCC}$ *rounds from any configuration.*

Roughly speaking, we define a *GC*-segment, for any connected component *GC*, as a part of execution between two removals of a non-frozen abnormal tree. A key property of our algorithm is that non-frozen abnormal trees are never created. Combined with other properties, this allows us to prove that there are at most $n_{\mathtt{maxCC}} + 1$ *GC*-segments. The sequence of rules executed by a process $u$ of *GC* during a *GC*-segment belongs to the following language: $(\mathbf{R_I} + \varepsilon)(\mathbf{R_R} + \varepsilon)(\mathbf{R_U})^*(\mathbf{R_{EB}} + \varepsilon)(\mathbf{R_{EF}} + \varepsilon)$. This further leads to the two following key results.

**Theorem 2** *If the number of* $\mathbf{R_U}$ *executions during a GC-segment by any process of GC is bounded by* nb_U*, then the total number of moves (and steps) in any execution is bounded by* $(\mathrm{nb\_U} + 4)(n_{maxCC} + 1)n$.

**Theorem 3** *When all weights are strictly positive integers bounded by* $\mathbb{W}_{\max}$*, and* $\oplus$ *is the addition operator, the stabilization time of* Scheme *in moves (and steps) is at most* $(\mathbb{W}_{\max}(n_{maxCC} - 1)^2 + 5)(n_{maxCC} + 1)n$.

**Unconstrained and Shortest-Path Spanning Forest.** Given an input set of processes *rootSet*, and assuming (strictly) positive integer weights for each edge, Algorithms Forest and SPF are the instantiations of Scheme with the parameters given in Algorithm 2.

**Algorithm 2:** Parameters for any process $u$ in Algorithms Forest and SPF

**Inputs:**
(1) $canBeRoot_u$ is true if and only if $u \in rootSet$, (2) $pname_u$ is $\perp$, and (3) $\omega_u(v) = \omega_v(u) \in \mathbb{N}^*$, for every $v \in \Gamma(u)$.

**Ordered Magma:** (1) $DistSet = \mathbb{N}$, (2) $i1 \oplus i2 = i1 + i2$, (3) $i1 \prec i2 \equiv i1 < i2$, and (4) $distRoot(u) = 0$.

**Predicate:**
Forest: $P\_nodeImp(u) \equiv P\_updateRoot(u)$
SPF: $P\_nodeImp(u) \equiv P\_updateNode(u) \vee P\_updateRoot(u)$

Algorithm Forest (resp. SPF) computes (in a self-stabilizing manner) an unconstrained (resp. shortest-path) spanning forest in each connected component of $G$ containing at least one process of *rootSet*. The forest consists of trees rooted at each process of *rootSet*. Moreover, in any component containing no process of *rootSet*, the processes eventually detect the absence of roots by taking the status $I$ (Isolated).

By Theorem 2 (resp. Theorem 3), Algorithms Forest and SPF self-stabilize to a terminal legitimate configuration in at most $O(n_{\texttt{maxCC}}n)$ (resp. $O(\mathbb{W}_{\max}{n_{\texttt{maxCC}}}^3 n)$) moves, where $\mathbb{W}_{\max}$ is the largest edge weight.

**Leader Election Algorithms.** Assuming the network is identified (each node has a unique identifier), Algorithm LEM and LEM_BFS are the instantiations of Scheme with the parameters given in Algorithm 3.

**Algorithm 3:** Parameters for any process $u$ in Algorithm LEM and LEM_BFS

**Inputs:** (1) $canBeRoot_u$ is true for any process, (2) $pname_u$ is the identifier of $u$ (*n.b.*, $pname_u \in \mathbb{N}$)
(3) $\omega_u(v) = (\perp, 1)$ for every $v \in \Gamma(u)$

**Ordered Magma:** (1) $DistSet = IDs \times \mathbb{N}$; for every $d = (a,b) \in DistSet$, we let $d.id = a$ and $d.h = b$;
(2) $(id_1,i_1) \oplus (id_2,i_2) = (id_1, i_1 + i_2)$; (3) $(id_1,i_1) \prec (id_2,i_2) \equiv (id_1 < id_2) \vee [(id_1 = id_2) \wedge (i_1 < i_2)]$;
(4) $distRoot(u) = (pname_u, 0)$

**Predicate:**
LEM: $P\_nodeImp(u) \equiv ((\exists v \in \Gamma(u) \mid st_v = C \wedge d_v.id < d_u.id)) \vee P\_updateRoot(u)$
LEM_BFS: $P\_nodeImp(u) \equiv P\_updateNode(u) \vee P\_updateRoot(u)$

In each connected component, Algorithm LEM and LEM_BFS elect the process $u$ (*i.e.*, $P\_leader(u)$ holds) of smallest identifier and builds a tree rooted at $u$ that spans the whole connected component. Algorithm LEM builds a tree of arbitrary topology; algorithm LEM_BFS builds a breadth-first-search tree.

By Theorem 3, Algorithm LEM_BFS, self-stabilizes to a terminal legitimate configuration in at most $O({n_{\texttt{maxCC}}}^3 n)$ moves. By Theorem 2, Algorithm LEM, self-stabilizes to a terminal legitimate configuration in at most $(2n_{\texttt{maxCC}} + 4)(n_{\texttt{maxCC}} + 1)n$ moves (*i.e.* $O({n_{\texttt{maxCC}}}^2 n)$ moves) since during a *GC*-segment, a process can only execute $\mathbf{R}_{\mathrm{U}}$ to improve its ID.

# References


[ACD+17] K. Altisen, A. Cournier, S. Devismes, A. Durand, and F. Petit. Self-stabilizing leader election in polynomial steps. *Information and Computation*, 254:330 – 366, 2017.

[Cou09] A. Cournier. A new polynomial silent stabilizing spanning-tree construction algorithm. In *Int. Col. on Struct. Inf. and Comm. Complexity, SIROCCO'09, LNCS 5869*, pages 141–153, 2009.

[DIJ16] S. Devismes, D. Ilcinkas, and C. Johnen. Self-stabilizing disconnected components detection and rooted shortest-path tree maintenance in polynomial steps. In *20th Int. Conf. on Principles of Distributed Systems, OPODIS'16*, volume 70 of *LIPIcs*, pages 10:1–10:16, 2016.

[DJ16] Stéphane Devismes and Colette Johnen. Silent self-stabilizing BFS tree algorithms revisited. *Journal of Parallel and Distributed Computing*, 97:11 – 23, 2016.

[Gär03] F. C. Gärtner. A survey of self-stabilizing spanning-tree construction algorithms. Technical report, Swiss Federal Institute of Technolog (EPFL), 2003.