Acyclic Strategy for Silent Self-Stabilization in Spanning Forest

Karine Altisen[1] Stéphane Devismes[1] Anaïs Durand[2]

[1] Univ. Grenoble Alpes, CNRS, Grenoble INP, VERIMAG, 38000 Grenoble, France
[2] IRISA, Université de Rennes, 35042 Rennes, France

SSS'2018, November 5th, Tokyo (Japan)

Roadmap

Introduction

Trees/Forests: Ubiquitous in Self-Stabilization

Composition is a popular way to design self-stabilizing algorithms (modular approach, simplicity of the design and proofs)

Numerous self-stabilizing algorithms [Arora et al., 1990, Blin et al., 2010, Datta et al., 2016] are made as **a composition of**

- **a spanning directed treelike construction** and
- some other **algorithms specifically designed for directed tree/forest topologies**.

Many solutions are silent

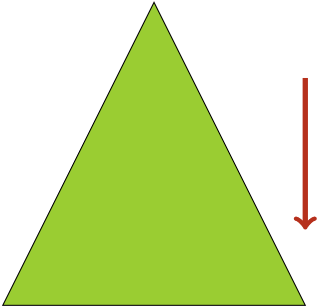
Spanning Directed Treelike Constructions

Many (silent) self-stabilizing spanning tree constructions are available, *e.g.*:

- **Arbitrary Tree:** [Chen et al., 1991]
- **DFS:** [Collin and Dolev, 1994]
- **BFS:** [Cournier et al., 2009, Cournier et al., 2011]
- **Shortest-Path:** [Glacet et al., 2014]
- ...
- **(Efficient) General Scheme:** [Devismes et al., 2019]

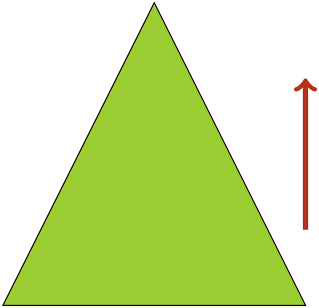
Self-stabilizing Algorithms for Directed Spanning Tree/Forest

Classical design pattern based on top-down (broadcast) and bottom-up (convergecast) computations:



Top-Down

Computations are propagated from parents to nodes

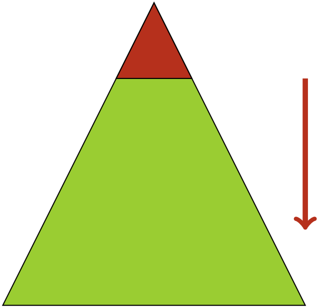


Bottom-Up

Computations are propagated from children to nodes

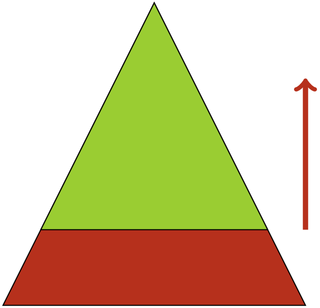
Self-stabilizing Algorithms for Directed Spanning Tree/Forest

Classical design pattern based on top-down (broadcast) and bottom-up (convergecast) computations:



Top-Down

Computations are propagated from parents to nodes

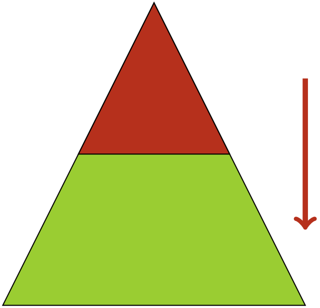


Bottom-Up

Computations are propagated from children to nodes

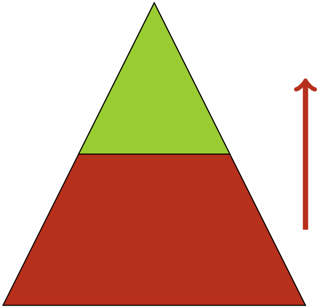
Self-stabilizing Algorithms for Directed Spanning Tree/Forest

Classical design pattern based on top-down (broadcast) and bottom-up (convergecast) computations:



Top-Down

Computations are propagated from parents to nodes

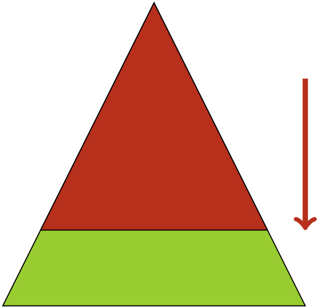


Bottom-Up

Computations are propagated from children to nodes

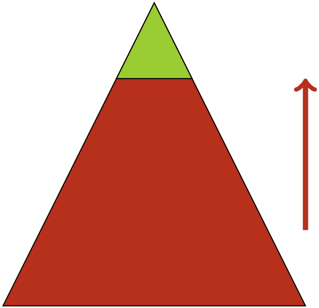
Self-stabilizing Algorithms for Directed Spanning Tree/Forest

Classical design pattern based on top-down (broadcast) and bottom-up (convergecast) computations:



Top-Down

Computations are propagated from parents to nodes

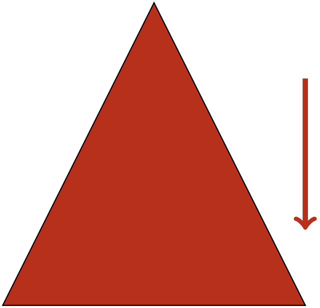


Bottom-Up

Computations are propagated from children to nodes

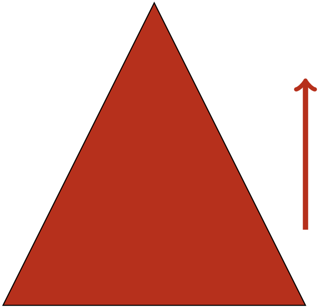
Self-stabilizing Algorithms for Directed Spanning Tree/Forest

Classical design pattern based on top-down (broadcast) and bottom-up (convergecast) computations:



Top-Down

Computations are propagated from parents to nodes



Bottom-Up

Computations are propagated from children to nodes

Our Goal

Define a class of algorithms for networks endowed with a spanning forest (*e.g.*, a spanning tree) based the notions of **top-down** and **bottom-up** computations.

- The definition should be **simple** to check (*i.e.*, quasi-syntactic)
- Algorithms of the class should be **(silent) self-stabilizing**
- Algorithms of the class should be **efficient** in stabilization time

Challenge: Trade-off efficiency/versatility

Context

- **Locally shared memory model** with **composite atomicity**
  - Distributed unfair daemon
    (the most general scheduling assumption)
- **Silent self-stabilizing** algorithms
- **Sense of direction** defining spanning forest
  - *p.par*: *p.par* is either a neighbor (its *parent*), or $\perp$ (for a root).
  - *p.chldrn*: the set of *children*
- Time complexity in **moves** and **rounds**

Contribution

Acyclic Strategy

Definition 1.

A distributed algorithm $\mathcal{A}$ follows an **acyclic strategy** if

- it is well-formed,
- its graph of actions' causality **GC** is (directed) acyclic, and
- for every A_i in its families' partition, A_i is
  - correct-alone and
  - either bottom-up or top-down.

syntactic condition / semantic condition

(An illustrative example in few slides ...)

Main Result

Theorem 1.

Let $\mathcal{A}$ be a distributed algorithm. If

- *$\mathcal{A}$ follows an acyclic strategy,*
- *every terminal configuration of $\mathcal{A}$ satisfies SP*
- *$\mathcal{A}$ is locally mutually exclusive*

then

- *$\mathcal{A}$ is silent and self-stabilizing for SP in G under* **the distributed unfair daemon**
- **its stabilization time** *is at most $(1 + \mathbf{d} \cdot (1 + \Delta))^{\mathfrak{H}} \cdot k \cdot n^{\mathfrak{H}+2}$ moves*
- **its stabilization time** *is at most $(\mathfrak{H} + 1) \cdot (H + 1)$ rounds*

(Δ is the degree of the network, k is the number of families of $\mathcal{A}$, $\mathbf{d}$ is the in-degree of **GC**, $\mathfrak{H}$ the height of **GC**, H is the height of the spanning forest)

(typically, k, $\mathbf{d}$, and $\mathfrak{H}$ are constants)

Main Result

Theorem 1.

Let $\mathcal{A}$ be a distributed algorithm. If

- *$\mathcal{A}$ follows an acyclic strategy,*
- *every terminal configuration of $\mathcal{A}$ satisfies SP*
- *$\mathcal{A}$ is locally mutually exclusive*

then

- *$\mathcal{A}$ is silent and self-stabilizing for SP in G under* **the distributed unfair daemon**
- **its stabilization time** *is at most $(1 + \mathbf{d} \cdot (1 + \Delta))^{\mathfrak{H}} \cdot k \cdot n^{\mathfrak{H}+2}$ moves*
- **its stabilization time** *is at most $(\mathfrak{H} + 1) \cdot (H + 1)$ rounds*

(Δ is the degree of the network, k is the number of families of $\mathcal{A}$, $\mathbf{d}$ is the in-degree of **GC**, $\mathfrak{H}$ the height of **GC**, H is the height of the spanning forest)

(typically, k, $\mathbf{d}$, and $\mathfrak{H}$ are constants)

Acyclic Strategy

Toy Example

Compute a sum of inputs and broadcast the result

- Each process p holds a constant integer input $p.in \in \mathbb{N}$
- The network is a directed tree rooted at r

Toy Example

Compute a sum of inputs and broadcast the result

- Each process p holds a constant integer input $p.in \in \mathbb{N}$
- The network is a directed tree rooted at r
- Every process p has two variables:
  - $p.sub \in \mathbb{N}$ (to compute the sum of input values in the subtree of p)
  - $p.res \in \mathbb{N}$ (to broadcast the result).

Toy Example

Compute a sum of inputs and broadcast the result

- Each process p holds a constant integer input $p.in \in \mathbb{N}$
- The network is a directed tree rooted at r
- Every process p has two variables:
  - $p.sub \in \mathbb{N}$ (to compute the sum of input values in the subtree of p)
  - $p.res \in \mathbb{N}$ (to broadcast the result).
- Legitimacy predicate: $\texttt{SumOfInputs} \equiv \forall p \in V, p.res = \sum_{q \in V} q.in$

Toy Example

Algorithm $\mathcal{TE}$

For every process p

$S(p) :: p.sub \neq (\sum_{q \in p.chldrn} q.sub) + p.in \rightarrow p.sub \leftarrow (\sum_{q \in p.chldrn} q.sub) + p.in$

Toy Example

Algorithm $\mathcal{TE}$

For every process p

$S(p) :: p.sub \neq (\sum_{q \in p.chldrn} q.sub) + p.in \rightarrow p.sub \leftarrow (\sum_{q \in p.chldrn} q.sub) + p.in$

For process r

$R(r) :: r.res \neq r.sub \rightarrow r.res \leftarrow r.sub$

For every process $p \neq r$

$R(p) :: p.res \neq \max(p.par.res, p.sub) \rightarrow p.res \leftarrow \max(p.par.res, p.sub)$

Families and Well-Formedness

Definition 1.

A distributed algorithm $\mathcal{A}$ follows an **acyclic strategy** if

- it is well-formed,
- its graph of actions' causality C is (directed) acyclic, and
- for every A_i in its families' partition, A_i is
  - correct-alone and
  - either bottom-up or top-down.

- **Family of actions:** set of n actions, one per process
- **Well-Formed:** Actions partitioned into families s.t. **each variable is written in exactly one family**

(Well-formedness is rather a guideline to simplify the analysis.)

Families and Well-Formedness

Definition 1.

A distributed algorithm $\mathcal{A}$ follows an **acyclic strategy** if

- it is well-formed,
- its graph of actions' causality C is (directed) acyclic, and
- for every A_i in its families' partition, A_i is
  - correct-alone and
  - either bottom-up or top-down.

- **Family of actions:** set of n actions, one per process
- **Well-Formed:** Actions partitioned into families s.t. **each variable is written in exactly one family**

(Well-formedness is rather a guideline to simplify the analysis.)

$\mathcal{TE}$ **is well-formed:** two families S and R

- $S = \{S(p) \ : \ p \in V\}$
- $R = \{R(p) \ : \ p \in V\}$

$$S(p) :: p.sub \neq (\sum_{q \in p.chldrn} q.sub) + p.in \rightarrow \mathbf{p.sub} \leftarrow (\sum_{q \in p.chldrn} q.sub) + p.in$$

$$R(r) :: r.res \neq r.sub \rightarrow \mathbf{r.res} \leftarrow r.sub$$

$$R(p) :: p.res \neq \max(p.par.res, p.sub) \rightarrow \mathbf{p.res} \leftarrow \max(p.par.res, p.sub)$$

Acyclicity of the graph of actions' causality

Definition 1.

A distributed algorithm $\mathcal{A}$ follows an **acyclic strategy** if

- it is well-formed,
- its graph of actions' causality **GC** is (directed) acyclic, and
- for every A_i in its families' partition, A_i is
  - correct-alone and
  - either bottom-up or top-down.

- $A_1, \ldots, A_k$: families' partition of $\mathcal{A}$.
- $A_j \prec_{\mathcal{A}} A_i$ iff
  - $i \neq j$ and
  - $\exists p, q$ s.t. $A_j(p)$ writes in variables "read" by $A_i(q)$.
- $\mathbf{GC} = (\{A_1, \ldots, A_k\}, \{(A_j, A_i), A_j \prec_{\mathcal{A}} A_i\})$

Acyclicity of the graph of actions' causality

Definition 1.

A distributed algorithm $\mathcal{A}$ follows an **acyclic strategy** if

- it is well-formed,
- its graph of actions' causality **GC** is (directed) acyclic, and
- for every A_i in its families' partition, A_i is
  - correct-alone and
  - either bottom-up or top-down.

- $A_1, ..., A_k$: families' partition of $\mathcal{A}$.
- $A_j \prec_{\mathcal{A}} A_i$ iff
  - $i \neq j$ and
  - $\exists p, q$ s.t. $A_j(p)$ writes in variables "read" by $A_i(q)$.
- $\mathbf{GC} = (\{A_1, ..., A_k\}, \{(A_j, A_i), A_j \prec_{\mathcal{A}} A_i\})$

$$S(p) :: p.sub \neq (\sum_{q \in p.chldrn} q.sub) + p.in \rightarrow \mathbf{p.sub} \leftarrow (\sum_{q \in p.chldrn} q.sub) + p.in$$

$$R(r) :: r.res \neq \mathbf{r.sub} \rightarrow r.res \leftarrow r.sub$$

$$R(p) :: p.res \neq \max(p.par.res, \mathbf{p.sub}) \rightarrow p.res \leftarrow \max(p.par.res, p.sub)$$

Acyclicity of the graph of actions' causality

Definition 1.

A distributed algorithm $\mathcal{A}$ follows an **acyclic strategy** if

- it is well-formed,
- its graph of actions' causality **GC** is (directed) acyclic, and
- for every A_i in its families' partition, A_i is
  - correct-alone and
  - either bottom-up or top-down.

- $A_1, ..., A_k$: families' partition of $\mathcal{A}$.
- $A_j \prec_{\mathcal{A}} A_i$ iff
  - $i \neq j$ and
  - $\exists p, q$ s.t. $A_j(p)$ writes in variables "read" by $A_i(q)$.
- $\mathbf{GC} = (\{A_1, ..., A_k\}, \{(A_j, A_i), A_j \prec_{\mathcal{A}} A_i\})$

$$S(p) :: p.sub \neq (\sum_{q \in p.chldrn} q.sub) + p.in \rightarrow \mathbf{p.sub} \leftarrow (\sum_{q \in p.chldrn} q.sub) + p.in$$

$$R(r) :: r.res \neq \mathbf{r.sub} \rightarrow r.res \leftarrow r.sub$$

$$R(p) :: p.res \neq \max(p.par.res, \mathbf{p.sub}) \rightarrow p.res \leftarrow \max(p.par.res, p.sub)$$

$$S \prec_{\mathcal{TE}} R$$

$\mathbf{GC} : S \longrightarrow R$ is acyclic

All families are correct-alone

Definition 1.

A distributed algorithm $\mathcal{A}$ follows an **acyclic strategy** if

- it is well-formed,
- its graph of actions' causality **GC** is (directed) acyclic, and
- for every A_i in its families' partition, A_i is
  - correct-alone and
  - either bottom-up or top-down.

- $A_1, ..., A_k$: families' partition of $\mathcal{A}$.
- A_i is correct-alone if $\forall p$, $A_i(p)$ becomes disabled whenever variables "read" by $A_i(p)$ are only written by $A_i(p)$ in a step.

All families are correct-alone

Definition 1.

A distributed algorithm $\mathcal{A}$ follows an **acyclic strategy** if

- it is well-formed,
- its graph of actions' causality **GC** is (directed) acyclic, and
- for every A_i in its families' partition, A_i is
  - correct-alone and
  - either bottom-up or top-down.

- $A_1, ..., A_k$: families' partition of $\mathcal{A}$.
- A_i is correct-alone if $\forall p$, $A_i(p)$ becomes disabled whenever variables "read" by $A_i(p)$ are only written by $A_i(p)$ in a step.

S and R are correct-alone

$$S(p) :: p.sub \neq (\sum_{q \in p.chldrn} q.sub) + p.in \rightarrow p.sub \leftarrow (\sum_{q \in p.chldrn} q.sub) + p.in$$

$$R(r) :: r.res \neq r.sub \rightarrow r.res \leftarrow r.sub$$

$$R(p) :: p.res \neq \max(p.par.res, p.sub) \rightarrow p.res \leftarrow \max(p.par.res, p.sub)$$

Every family is either bottom-up or top-down

Definition 1.

A distributed algorithm $\mathcal{A}$ follows an **acyclic strategy** if

- it is well-formed,
- its graph of actions' causality **GC** is (directed) acyclic, and
- for every A_i in its families' partition, A_i is
  - correct-alone and
  - either bottom-up or top-down.

- A_i is top-down: $\forall p$, any variable "read" by $A_i(p)$ is written by $A_i(q)$ only if $q = p$ or $q = p.par$.
- A_i is bottom-up: $\forall p$, any variable "read" by $A_i(p)$ is written by $A_i(q)$ only if $q = p$ or $q \in p.chldrn$.

Every family is either bottom-up or top-down

Definition 1.

A distributed algorithm $\mathcal{A}$ follows an **acyclic strategy** if

- it is well-formed,
- its graph of actions' causality **GC** is (directed) acyclic, and
- for every A_i in its families' partition, A_i is
  - correct-alone and
  - either bottom-up or top-down.

- A_i is top-down: $\forall p$, any variable "read" by $A_i(p)$ is written by $A_i(q)$ only if $q = p$ or $q = p.par$.
- A_i is bottom-up: $\forall p$, any variable "read" by $A_i(p)$ is written by $A_i(q)$ only if $q = p$ or $q \in p.chldrn$.

S is bottom-up

$$S(p) :: p.sub \neq (\sum_{q \in p.chldrn} q.sub) + p.in \rightarrow p.sub \leftarrow (\sum_{q \in p.chldrn} q.sub) + p.in$$

R is top-down

$$R(r) :: r.res \neq r.sub \rightarrow r.res \leftarrow r.sub$$

$$R(p) :: p.res \neq \max(p.par.res, p.sub) \rightarrow p.res \leftarrow \max(p.par.res, p.sub)$$

Every family is either bottom-up or top-down

Definition 1.

A distributed algorithm $\mathcal{A}$ follows an **acyclic strategy** if

- it is well-formed,
- its graph of actions' causality **GC** is (directed) acyclic, and
- for every A_i in its families' partition, A_i is
  - correct-alone and
  - either bottom-up or top-down.

- A_i is top-down: $\forall p$, any variable "read" by $A_i(p)$ is written by $A_i(q)$ only if $q = p$ or $q = p.par$.
- A_i is bottom-up: $\forall p$, any variable "read" by $A_i(p)$ is written by $A_i(q)$ only if $q = p$ or $q \in p.chldrn$.

S is bottom-up

$$S(p) :: p.sub \neq (\sum_{q \in p.chldrn} q.sub) + p.in \rightarrow \mathbf{p.sub} \leftarrow (\sum_{q \in p.chldrn} q.sub) + p.in$$

R is top-down

$$R(r) :: r.res \neq r.sub \rightarrow \mathbf{r.res} \leftarrow r.sub$$

$$R(p) :: p.res \neq \max(p.par.res, p.sub) \rightarrow \mathbf{p.res} \leftarrow \max(p.par.res, p.sub)$$

Every family is either bottom-up or top-down

Definition 1.

A distributed algorithm $\mathcal{A}$ follows an **acyclic strategy** if

- it is well-formed,
- its graph of actions' causality **GC** is (directed) acyclic, and
- for every A_i in its families' partition, A_i is
  - correct-alone and
  - either bottom-up or top-down.

- A_i is top-down: $\forall p$, any variable "read" by $A_i(p)$ is written by $A_i(q)$ only if $q = p$ or $q = p.par$.
- A_i is bottom-up: $\forall p$, any variable "read" by $A_i(p)$ is written by $A_i(q)$ only if $q = p$ or $q \in p.chldrn$.

S is bottom-up

$$S(p) :: p.sub \neq (\sum_{q \in p.chldrn} q.sub) + p.in \rightarrow \mathbf{p.sub} \leftarrow (\sum_{q \in p.chldrn} q.sub) + p.in$$

R is top-down

$$R(r) :: r.res \neq r.sub \rightarrow \mathbf{r.res} \leftarrow r.sub$$

$$R(p) :: p.res \neq \max(p.par.res, p.sub) \rightarrow \mathbf{p.res} \leftarrow \max(p.par.res, p.sub)$$

Result for $\mathcal{TE}$ (1/2)

Theorem 1.

Let $\mathcal{A}$ be a distributed algorithm. If

- $\mathcal{A}$ follows an acyclic strategy,
- every terminal configuration of $\mathcal{A}$ satisfies SP
- $\mathcal{A}$ is locally mutually exclusive

then

- $\mathcal{A}$ is silent and self-stabilizing for SP in G under **the distributed unfair daemon**
- **its stabilization time** is at most $\Big(1 + \mathbf{d} \cdot (1 + \Delta)\Big)^{\mathfrak{H}} \cdot k \cdot n^{\mathfrak{H}+2}$ moves
- **its stabilization time** is at most $(\mathfrak{H} + 1) \cdot (H + 1)$ rounds

- $\mathcal{TE}$ follows an acyclic strategy
- Every terminal configuration of $\mathcal{TE}$ satisfies `SumOfInputs` (a trivial induction)

Result for $\mathcal{TE}$ (1/2)

Theorem 1.

Let $\mathcal{A}$ be a distributed algorithm. If

- $\mathcal{A}$ follows an acyclic strategy,
- every terminal configuration of $\mathcal{A}$ satisfies SP
- $\mathcal{A}$ is locally mutually exclusive

then

- $\mathcal{A}$ is silent and self-stabilizing for SP in G under **the distributed unfair daemon**
- **its stabilization time** is at most $\left(1 + \mathbf{d} \cdot (1 + \Delta)\right)^{\mathfrak{H}} \cdot k \cdot n^{\mathfrak{H}+2}$ moves
- **its stabilization time** is at most $(\mathfrak{H} + 1) \cdot (H + 1)$ rounds

- $\mathcal{TE}$ follows an acyclic strategy
- Every terminal configuration of $\mathcal{TE}$ satisfies `SumOfInputs` (a trivial induction)

- $\mathcal{TE}$ is silent and self-stabilizing for `SumOfInputs` in G under **the distributed unfair daemon**

Result for $\mathcal{TE}$ (1/2)

Theorem 1.

Let $\mathcal{A}$ be a distributed algorithm. If

- $\mathcal{A}$ follows an acyclic strategy,
- every terminal configuration of $\mathcal{A}$ satisfies SP
- $\mathcal{A}$ is locally mutually exclusive

then

- $\mathcal{A}$ is silent and self-stabilizing for SP in G under **the distributed unfair daemon**
- **its stabilization time** is at most $\left(1 + \mathbf{d} \cdot (1 + \Delta)\right)^{\mathfrak{H}} \cdot k \cdot n^{\mathfrak{H}+2}$ moves
- **its stabilization time** is at most $(\mathfrak{H} + 1) \cdot (H + 1)$ rounds

- $\mathcal{TE}$ follows an acyclic strategy
- Every terminal configuration of $\mathcal{TE}$ satisfies `SumOfInputs` (a trivial induction)
-
  - $\mathbf{d} = 1$ (in-degree of **GC**)
  - $k = 2$ (number of families)
  - $\mathfrak{H} = 1$ (height of **GC**)

- $\mathcal{TE}$ is silent and self-stabilizing for `SumOfInputs` in G under **the distributed unfair daemon**
- **its stabilization time** is at most $(4 + 2\Delta) \cdot n^3$ moves, where Δ is the degree of G and n the number of processes

Result for $\mathcal{TE}$ (2/2)

The complexity of $\mathcal{TE}$ can be refined to at most $n^2(3+2H)$ moves where n the number of processes and H is the height of the spanning tree using the following technical lemma

Lemma 1.

Let A_i be a family of actions and p be a process. For every execution e of the algorithm $\mathcal{A}$ on G, $\#m(e, A_i, p) \leq \left(n \cdot \left(1 + \mathbf{d} \cdot \left(1 + maxO(A_i)\right)\right)\right)^{\mathfrak{H}(A_i)} \cdot |Z(p, A_i)|$.

Result for $\mathcal{TE}$ (2/2)

The complexity of $\mathcal{TE}$ can be refined to at most $n^2(3 + 2H)$ moves where n the number of processes and H is the height of the spanning tree using the following technical lemma

Lemma 1.

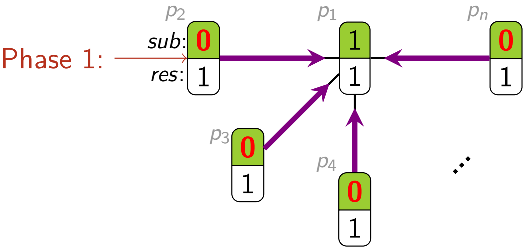
Let A_i be a family of actions and p be a process. For every execution e of the algorithm $\mathcal{A}$ on G, $\#m(e, A_i, p) \leq \left(n \cdot \left(1 + \mathbf{d} \cdot \left(1 + maxO(A_i)\right)\right)\right)^{\mathfrak{H}(A_i)} \cdot |Z(p, A_i)|$.

This bound is tight: there is an execution of $\mathcal{TE}$ containing $O(H \cdot n^2)$ moves

Round Complexity

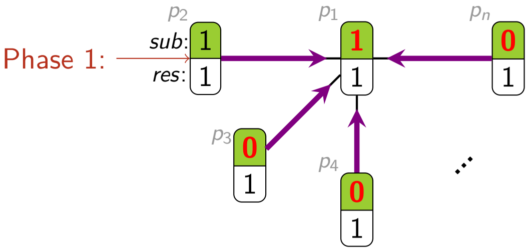
Lower Bound in Rounds for Algorithm $\mathcal{TE}$

$\forall i \in \{1, \ldots, n\},\ p_i.in = 1$



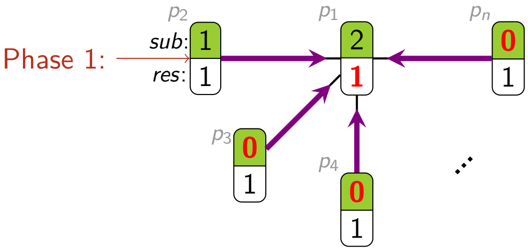
Lower Bound in Rounds for Algorithm $\mathcal{TE}$

$\forall i \in \{1, \ldots, n\}$, $p_i.in = 1$

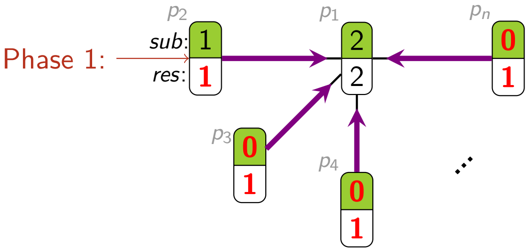


Lower Bound in Rounds for Algorithm $\mathcal{TE}$

$\forall i \in \{1, \ldots, n\},\ p_i.in = 1$

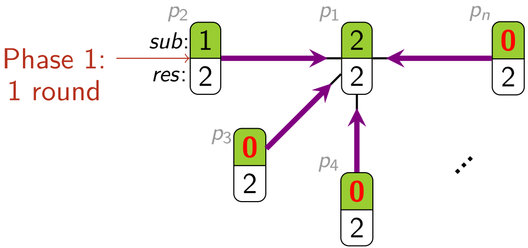


$\forall i \in \{1, \ldots, n\},\ p_i.in = 1$



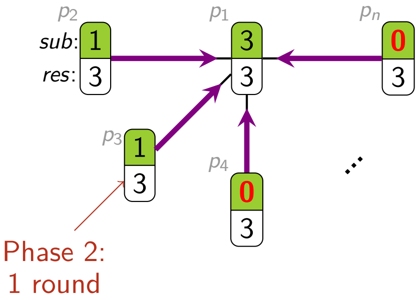
Lower Bound in Rounds for Algorithm $\mathcal{TE}$

$\forall i \in \{1, \ldots, n\}$, $p_i.in = 1$



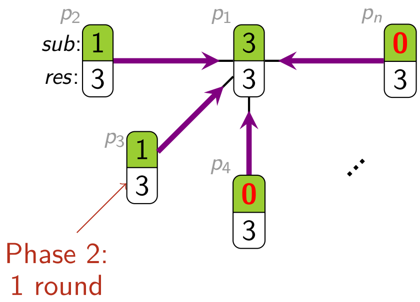
Lower Bound in Rounds for Algorithm $\mathcal{TE}$

$\forall i \in \{1, \ldots, n\},\ p_i.in = 1$



Lower Bound in Rounds for Algorithm $\mathcal{TE}$

$\forall i \in \{1, \ldots, n\}, \ p_i.in = 1$



$n - 1$ phases $\Rightarrow \Omega(n)$ rounds

Condition for a Stabilization Time in $O(H)$ Rounds

Round complexity of $\mathcal{TE} = \Omega(n)$ rounds $\Rightarrow$ **not optimal!**

Why?

Condition for a Stabilization Time in $O(H)$ Rounds

Round complexity of $\mathcal{TE} = \Omega(n)$ rounds $\Rightarrow$ **not optimal!**

Why? S and R of $\mathcal{TE}$ are not **mutually exclusive**

Condition for a Stabilization Time in $O(H)$ Rounds

Round complexity of $\mathcal{TE} = \Omega(n)$ rounds $\Rightarrow$ **not optimal!**

Why? S and R of $\mathcal{TE}$ are not **mutually exclusive**

Theorem 2.

Let $\mathcal{A}$ be a distributed algorithm. If $\mathcal{A}$

- *follows an acyclic strategy and*
- *is locally mutually exclusive*

then every execution of A reaches a terminal configuration within at most at most $(\mathfrak{H} + 1) \cdot (H + 1)$ rounds

($\mathfrak{H}$ is the height of **GC** and H is the height of the spanning forest)

(typically, $\mathfrak{H}$ is a constant)

Transformation into Locally Mutually Exclusive Algorithm

- **Idea:** use a strict total order compatible with the partial order $\prec_{\mathcal{A}}$ to implement priorities on actions locally at each process

Transformation into Locally Mutually Exclusive Algorithm

- **Idea:** use a strict total order compatible with the partial order $\prec_{\mathcal{A}}$ to implement priorities on actions locally at each process

- **Application on the Toy Example:**

$\mathcal{TE}$

- For every process p:
  $S(p) :: p.sub \neq (\sum_{q \in p.chldrn} q.sub) + p.in$
  $\quad \rightarrow p.sub \leftarrow (\sum_{q \in p.chldrn} q.sub) + p.in$
- For process r:
  $R(r) :: r.res \neq r.sub$
  $\quad \rightarrow r.res \leftarrow r.sub$
- For every process $p \neq r$:
  $R(p) :: p.res \neq \max(p.par.res, p.sub)$
  $\quad \rightarrow p.res \leftarrow \max(p.par.res, p.sub)$

Transformation into Locally Mutually Exclusive Algorithm

- **Idea:** use a strict total order compatible with the partial order $\prec_{\mathcal{A}}$ to implement priorities on actions locally at each process

- **Application on the Toy Example:**

$\mathtt{T}(\mathcal{TE})$ using $S < R$, *i.e.* $S(p)$ as priority over $R(p)$, $\forall p$

- For every process p:
  $S(p) :: p.sub \neq (\sum_{q \in p.chldrn} q.sub) + p.in$
  $\rightarrow p.sub \leftarrow (\sum_{q \in p.chldrn} q.sub) + p.in$
- For process r:
  $R(r) :: (r.sub = (\sum_{q \in r.chldrn} q.sub) + r.in) \wedge r.res \neq r.sub$
  $\rightarrow r.res \leftarrow r.sub$
- For every process $p \neq r$:
  $R(p) :: (p.sub = (\sum_{q \in p.chldrn} q.sub) + p.in) \wedge p.res \neq \max(p.par.res, p.sub)$
  $\rightarrow p.res \leftarrow \max(p.par.res, p.sub)$

Transformation into Locally Mutually Exclusive Algorithm

Theorem 3.

Let $\mathcal{A}$ be a distributed algorithm. If

- *$\mathcal{A}$ follows an acyclic strategy and*
- *$\mathcal{A}$ is silent and self-stabilizing for SP in G under* **the distributed unfair daemon**

then

- *$\mathcal{T}(\mathcal{A})$ is silent and self-stabilizing for SP in G under* **the distributed unfair daemon**
- **its stabilization time** *is at most $(\mathfrak{H}+1)\cdot(H+1)$ rounds*
- **its stabilization time in moves** *is less than or equal to the one of $\mathcal{A}$*

($\mathfrak{H}$ the height of **GC** and H is the height of the spanning forest)

(typically, $\mathfrak{H}$ is a constant)

Result for $\mathcal{TE}$

Since $\mathfrak{H} = 1$, $(\mathfrak{H} + 1) \cdot (H + 1)$ gives

$$2H + 2$$

- $\mathtt{T}(\mathcal{TE})$ is silent and self-stabilizing for `SumOfInputs` in G under **the distributed unfair daemon**
- **its stabilization time** is at most $2H + 2$ rounds (asymptotically optimal)
- **its stabilization time in moves** is at most $O(H \cdot n^3)$ moves

Conclusion

Application to the Literature

- ★ (blue) Same results
- ★ (green) More general daemon
- ★ (purple) New/better complexity

- [Turau and Köhler, 2015] ★ (purple)
- [Chaudhuri and Thompson, 2005] ★ (green) ★ (purple)
- [Chaudhuri and Thompson, 2011] ★ (blue)
- [Chaudhuri, 1999a] ★ (purple)
- [Chaudhuri, 1999b] ★ (purple)
- [Karaata, 1999] ★ (green) ★ (purple)
- [Karaata and Chaudhuri, 1999] ★ (purple)
- [Devismes, 2005] ★ (blue)

Conclusion

- **Contribution:** General scheme to prove and analyze silent self-stabilizing algorithms designed for networks endowed with a spanning forest.

- **Future work:** How to compose those algorithms **carefully** with (silent) self-stabilizing spanning tree construction?

  i.e., to obtain efficient composite algorithms

Thank you for your attention

Questions?

Arora, A., Gouda, M., and Herman, T. (1990).
Composite routing protocols.
In SPDP'90, pages 70–78.

Blin, L., Potop-Butucaru, M., Rovedakis, S., and Tixeuil, S. (2010).
Loop-free super-stabilizing spanning tree construction.
In SSS'10, pages 50–64.

Chaudhuri, P. (1999a).
An $O(n^2)$ Self-Stabilizing Algorithm for Computing Bridge-Connected Components.
Computing, 62(1):55–67.

Chaudhuri, P. (1999b).
A note on self-stabilizing articulation point detection.
Journal of Systems Architecture, 45(14):1249–1252.

Chaudhuri, P. and Thompson, H. (2005).
Self-stabilizing tree ranking.
Int. J. Comput. Math., 82(5):529–539.

Chaudhuri, P. and Thompson, H. (2011).
Improved self-stabilizing algorithms for l(2, 1)-labeling tree networks.
Mathematics in Computer Science, 5(1):27–39.

Chen, N., Yu, H., and Huang, S. (1991).
A self-stabilizing algorithm for constructing spanning trees.
Information Processing Letters, 39:147–151.

Collin, Z. and Dolev, S. (1994).
Self-stabilizing depth-first search.
Inf. Process. Lett., 49(6):297–301.

Cournier, A., Devismes, S., and Villain, V. (2009).
Light enabling snap-stabilization of fundamental protocols.
ACM Transactions on Autonomous and Adaptive Systems, 4(1).

Cournier, A., Rovedakis, S., and Villain, V. (2011).
The first fully polynomial stabilizing algorithm for BFS tree construction.
In the 15th International Conference on Principles of Distributed Systems (OPODIS'11), Springer LNCS 7109, pages 159–174.

Datta, A. K., Devismes, S., Heurtefeux, K., Larmore, L. L., and Rivierre, Y. (2016).
Competitive self-stabilizing *k*-clustering.
TCS, 626:110–133.

Devismes, S. (2005).
A silent self-stabilizing algorithm for finding cut-nodes and bridges.
Parallel Processing Letters, 15(1-2):183–198.

Devismes, S., Ilcinkas, D., and Johnen, C. (2019).
Silent self-stabilizing scheme for spanning-tree-like constructions.
In 20th International Conference on Distributed Computing and Networking (ICDCN 2019), Bangalore, India. ACM. to appear.

Glacet, C., Hanusse, N., Ilcinkas, D., and Johnen, C. (2014).
Disconnected components detection and rooted shortest-path tree maintenance in networks.
In SSS'14, pages 120–134.

Karaata, M. H. (1999).
A self-stabilizing algorithm for finding articulation points.
Int. J. Found. Comput. Sci., 10(1):33–46.

Karaata, M. H. and Chaudhuri, P. (1999).
A self-stabilizing algorithm for bridge finding.
Dist. Comp., 12(1):47–53.

Turau, V. and Köhler, S. (2015).
A distributed algorithm for minimum distance-k domination in trees.
J. Graph Algorithms Appl., 19(1):223–242.