

Self-Stabilizing Disconnected Components Detection and Rooted Shortest-Path Tree Maintenance in Polynomial Steps

Stéphane Devismes¹ David Ilcinkas² Colette Johnen²

¹Univ. Grenoble Alpes, France

²CNRS & Univ. Bordeaux, France

GT DALGO at LIF, Marseille
October 26, 2017

Self-Stabilizing Disconnected Components Detection and **Rooted Shortest-Path Tree** Maintenance in Polynomial Steps

Stéphane Devismes¹ David Ilcinkas² Colette Johnen²

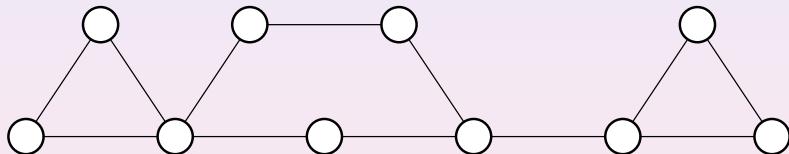
¹Univ. Grenoble Alpes, France

²CNRS & Univ. Bordeaux, France

GT DALGO at LIF, Marseille
October 26, 2017

Rooted shortest-path tree

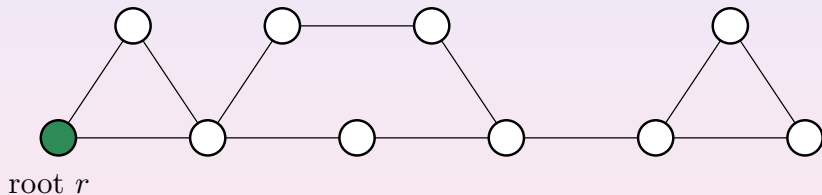
(Framework: distributed computing in networks.)



Basically, local knowledge of a shortest path to a fixed node.

Rooted shortest-path tree

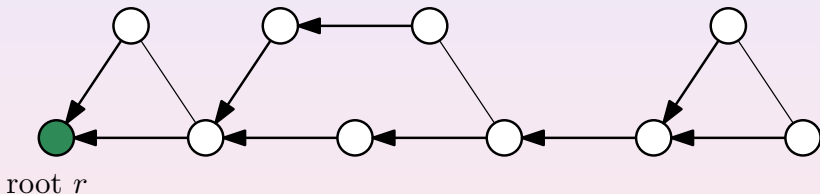
(Framework: distributed computing in networks.)



Basically, local knowledge of a shortest path to a fixed node.

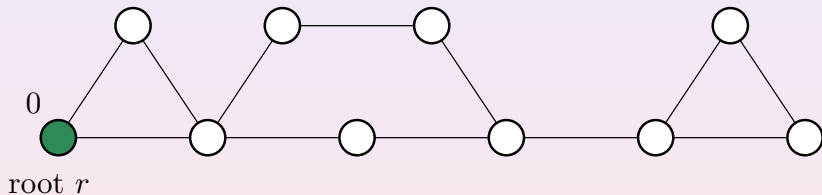
Rooted shortest-path tree

(Framework: distributed computing in networks.)



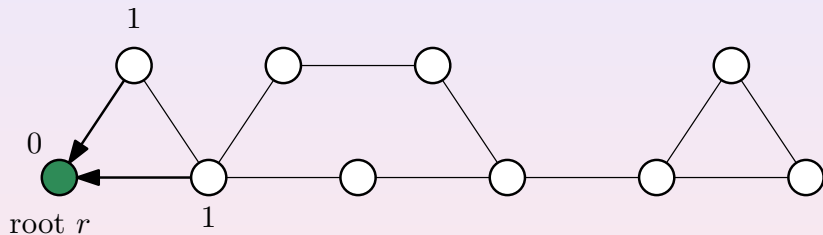
Basically, local knowledge of a **shortest** path to a **fixed** node.

Bellman-Ford Algorithm



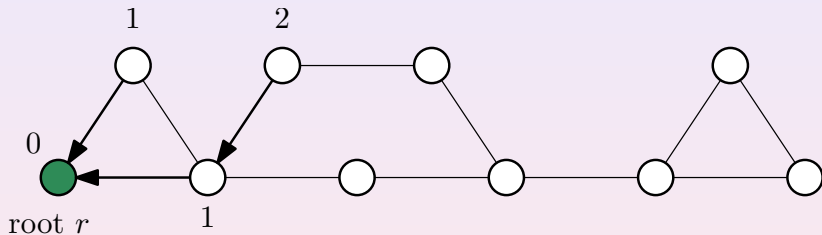
Distributed propagation of distances from the root.

Bellman-Ford Algorithm



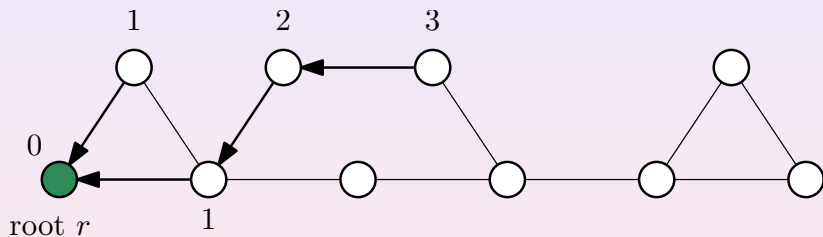
Distributed propagation of distances from the root.

Bellman-Ford Algorithm



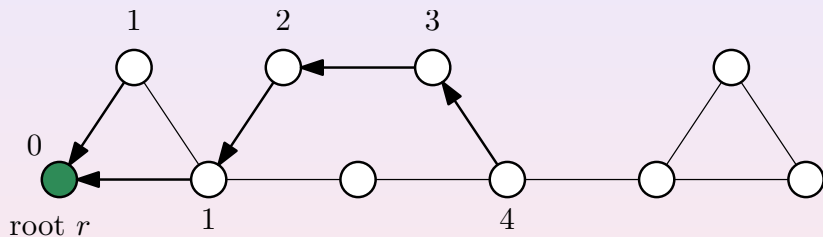
Distributed propagation of distances from the root.

Bellman-Ford Algorithm



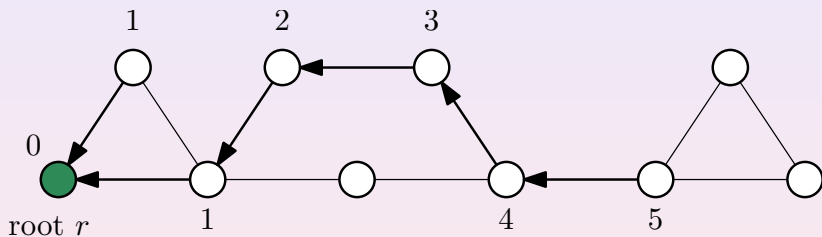
Distributed propagation of distances from the root.

Bellman-Ford Algorithm



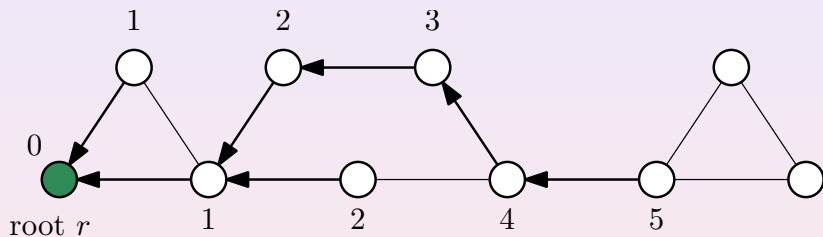
Distributed propagation of distances from the root.

Bellman-Ford Algorithm



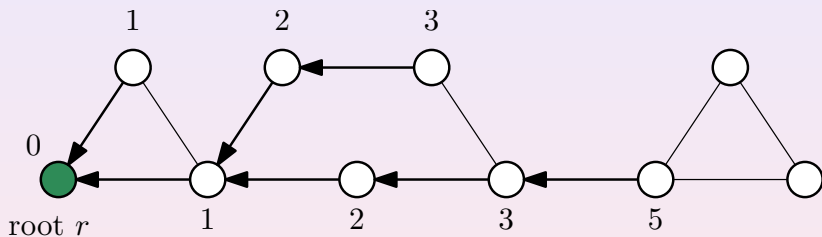
Distributed propagation of distances from the root.

Bellman-Ford Algorithm



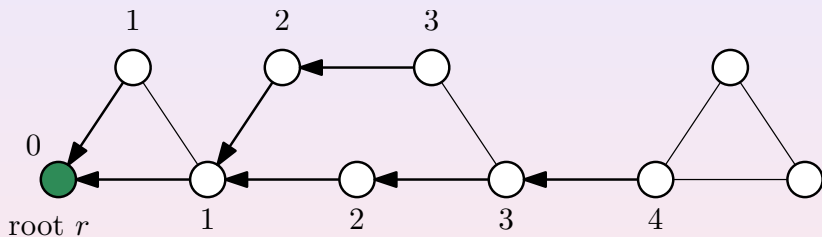
Distributed propagation of distances from the root.

Bellman-Ford Algorithm



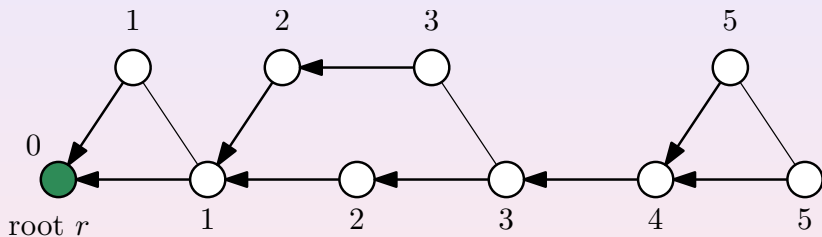
Distributed propagation of distances from the root.

Bellman-Ford Algorithm



Distributed propagation of distances from the root.

Bellman-Ford Algorithm



Distributed propagation of distances from the root.

Motivations

Classical building block for **routing in networks** ...

- In shortest-path routing schemes (like BGP)
- In compact routing schemes

... and in **distributed algorithms** in general.

In practice

- The network may change over time (faults, updates, etc.)
- The network may be even disconnected

Motivations

Classical building block for **routing in networks** ...

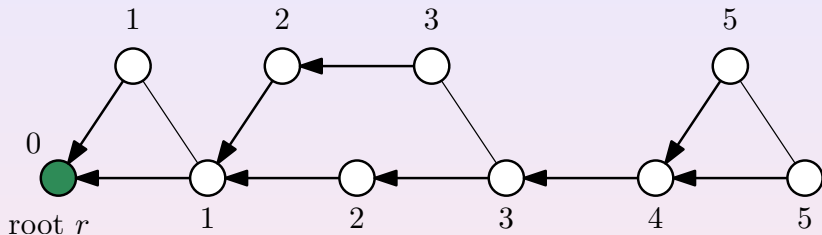
- In shortest-path routing schemes (like BGP)
- In compact routing schemes

... and in **distributed algorithms** in general.

In practice

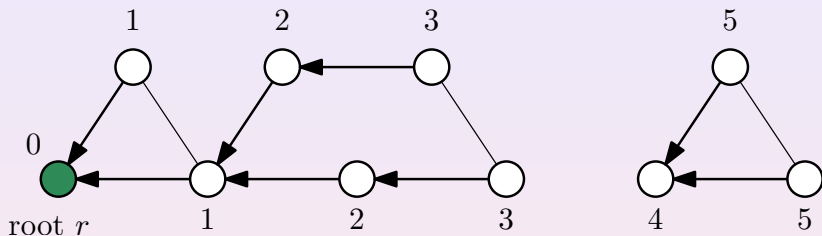
- The network may **change over time** (faults, updates, etc.)
- The network may be even **disconnected**

Count-to-infinity problem



- Bandwidth is uselessly consumed.
- Basic update mechanism may fail
⇒ useless control messages.

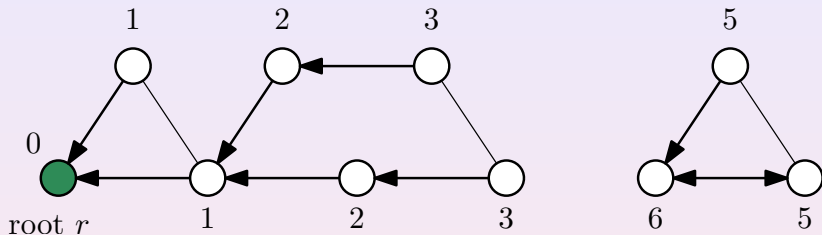
Count-to-infinity problem



1 Bandwidth is uselessly consumed.

Basic update mechanism may fail
⇒ useless control messages.

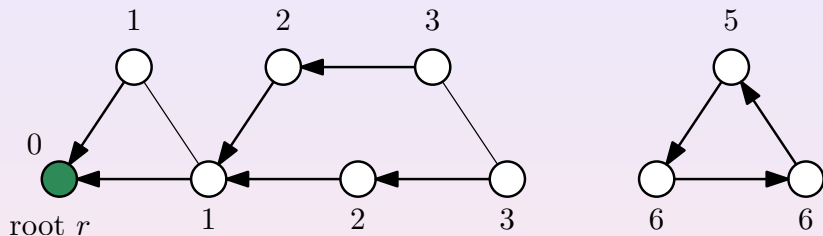
Count-to-infinity problem



1 Bandwidth is uselessly consumed.

2 Basic update mechanism may fail
⇒ useless control messages.

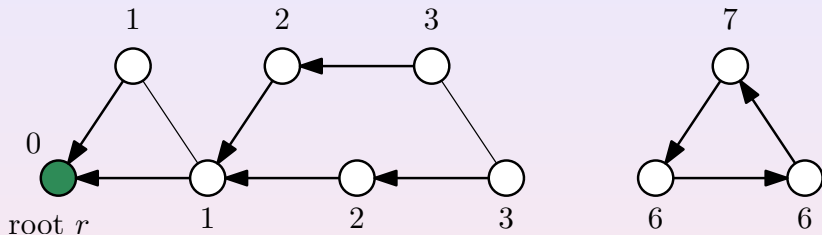
Count-to-infinity problem



1 Bandwidth is uselessly consumed.

Basic update mechanism may fail
⇒ useless control messages.

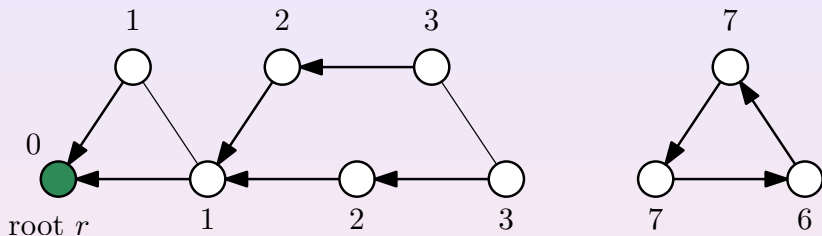
Count-to-infinity problem



1 Bandwidth is uselessly consumed.

2 Basic update mechanism may fail
⇒ useless control messages.

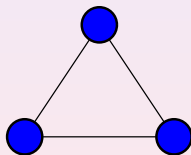
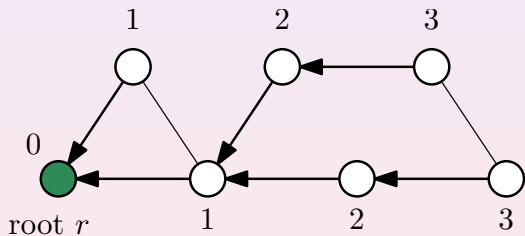
Count-to-infinity problem



- 1 Bandwidth is uselessly consumed.
- 2 Basic update mechanism may fail
⇒ useless control messages.

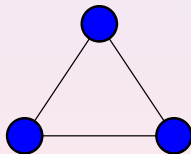
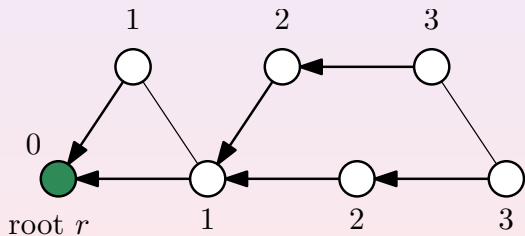
The problem at hand

Disconnected Components Detection and
rooted Shortest-Path tree Maintenance (DCDSPM)



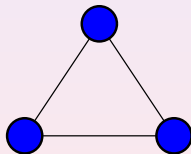
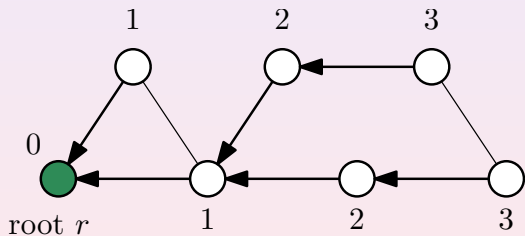
The problem at hand

Disconnected Components Detection and
rooted Shortest-Path tree Maintenance (DCDSPM)



The problem at hand

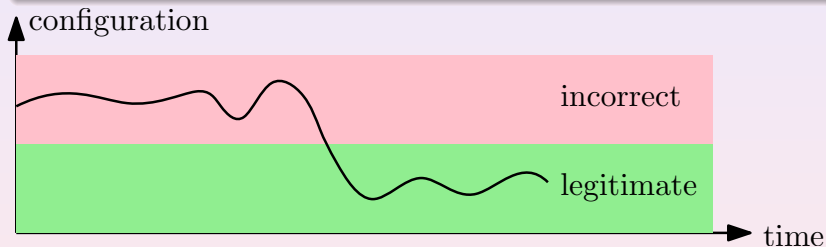
Disconnected Components Detection and
rooted Shortest-Path tree Maintenance (DCDSPM)



Self-Stabilization

Definition

A **self-stabilizing** distributed system will **eventually behave correctly** no matter its **initialization**.



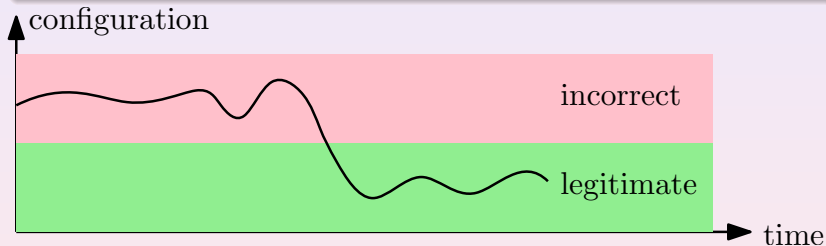
Context of use

- Transient faults
- Low-rate dynamics

Self-Stabilization

Definition

A **self-stabilizing** distributed system will **eventually behave correctly** no matter its **initialization**.



Context of use

- Transient faults
- Low-rate dynamics

Model (1/2)

Classical model, introduced by Dijkstra [Commun. ACM, 1974]

Locally shared memory model. . .

Each node u has **local variables** that can be

- read by u and its neighbors,
- written only by u

with composite atomicity

The state of a node is updated based on the local neighborhood in an atomic step.

Model (1/2)

Classical model, introduced by Dijkstra [Commun. ACM, 1974]

Locally shared memory model. . .

Each node u has **local variables** that can be

- read by u and its neighbors,
- written only by u

. . . with composite atomicity

The state of a node is **updated** based on the local neighborhood in an **atomic step**.

Model (2/2)

Program of a node

List of rules **Guard** \rightarrow **Action** for node u

- Guard: function of the variables of u and its neighbors
- Action: update of the local variables

Enabled

- A rule is enabled if its guard evaluates to true.
- A node is enabled if at least one of its rules is enabled.

Step and daemon (asynchrony)

At each step, the daemon must select at least one enabled node to be executed.

Model (2/2)

Program of a node

List of rules **Guard** \rightarrow **Action** for node u

- Guard: function of the variables of u and its neighbors
- Action: update of the local variables

Enabled

- A **rule** is **enabled** if its guard evaluates to true.
- A **node** is **enabled** if at least one of its rules is enabled.

Step and daemon (asynchrony)

At each step, the daemon must select at least one enabled node to be executed.

Model (2/2)

Program of a node

List of rules **Guard** \rightarrow **Action** for node u

- Guard: function of the variables of u and its neighbors
- Action: update of the local variables

Enabled

- A **rule** is **enabled** if its guard evaluates to true.
- A **node** is **enabled** if at least one of its rules is enabled.

Step and daemon (asynchrony)

At each **step**, the **daemon** must **select at least one** enabled node to be **executed**.

Our results

Main contribution

We propose and prove a **self-stabilizing** algorithm solving the **DCDSPM** problem with the following properties

- It works under *any daemon* (the unfair daemon).
- The algorithm is *silent*: eventually, no states are changed.
- Edges may have arbitrary *positive weights*.
- No *a priori* knowledge on global parameters (like n or D).
- *Linear in rounds*: $3n_{\max CC} + D$.
- *Polynomial in steps*: $O(W_{\max} n_{\max CC}^3 n)$.

- n : total number of nodes
- D : (hop-)diameter of the network
- $n_{\max CC}$: max. number of nodes in a connected component
- W_{\max} : maximum integer weight

Our results

Main contribution

We propose and prove a **self-stabilizing** algorithm solving the **DCDSPM** problem with the following properties

- It works under **any daemon** (the unfair daemon).
 - The algorithm is silent: eventually, no states are changed.
 - Edges may have arbitrary positive weights.
 - No a priori knowledge on global parameters (like n or D).
 - Linear in rounds: $3n_{\max CC} + D$.
 - Polynomial in steps: $O(W_{\max} n_{\max CC}^3 n)$.
-
- n : total number of nodes
 - D : (hop-)diameter of the network
 - $n_{\max CC}$: max. number of nodes in a connected component
 - W_{\max} : maximum integer weight

Our results

Main contribution

We propose and prove a **self-stabilizing** algorithm solving the **DCDSPM** problem with the following properties

- It works under **any daemon** (the unfair daemon).
- The algorithm is **silent**: eventually, no states are changed.
- Edges may have arbitrary positive weights.
- No a priori knowledge on global parameters (like n or D).
- Linear in rounds: $3n_{\max\text{CC}} + D$.
- Polynomial in steps: $O(W_{\max} n_{\max\text{CC}}^3 n)$.
- n : total number of nodes
- D : (hop-)diameter of the network
- $n_{\max\text{CC}}$: max. number of nodes in a connected component
- W_{\max} : maximum integer weight

Our results

Main contribution

We propose and prove a **self-stabilizing** algorithm solving the **DCDSPM** problem with the following properties

- It works under **any daemon** (the unfair daemon).
- The algorithm is **silent**: eventually, no states are changed.
- Edges may have arbitrary **positive weights**.

• No a priori knowledge on global parameters (like n or D).

• Linear in rounds: $3n_{\max CC} + D$.

• Polynomial in steps: $O(W_{\max} n_{\max CC}^3 n)$.

• n : total number of nodes

• D : (hop-)diameter of the network

• $n_{\max CC}$: max. number of nodes in a connected component

• W_{\max} : maximum integer weight

Our results

Main contribution

We propose and prove a **self-stabilizing** algorithm solving the **DCDSPM** problem with the following properties

- It works under **any daemon** (the unfair daemon).
- The algorithm is **silent**: eventually, no states are changed.
- Edges may have arbitrary **positive weights**.
- **No a priori knowledge** on global parameters (like n or D).

• Linear in rounds: $3n_{\max CC} + D$.

• Polynomial in steps: $O(W_{\max} n_{\max CC}^3 n)$.

- n : total number of nodes
- D : (hop-)diameter of the network
- $n_{\max CC}$: max. number of nodes in a connected component
- W_{\max} : maximum integer weight

Our results

Main contribution

We propose and prove a **self-stabilizing** algorithm solving the **DCDSPM** problem with the following properties

- It works under **any daemon** (the unfair daemon).
- The algorithm is **silent**: eventually, no states are changed.
- Edges may have arbitrary **positive weights**.
- **No a priori knowledge** on global parameters (like n or D).
- **Linear in rounds**: $3n_{\max\text{CC}} + D$.

• Polynomial in steps: $O(W_{\max} n_{\max\text{CC}}^3 n)$.

- n : total number of nodes
- D : (hop-)diameter of the network
- $n_{\max\text{CC}}$: max. number of nodes in a connected component

• W_{\max} : maximum integer weight

Our results

Main contribution

We propose and prove a **self-stabilizing** algorithm solving the **DCDSPM** problem with the following properties

- It works under **any daemon** (the unfair daemon).
- The algorithm is **silent**: eventually, no states are changed.
- Edges may have arbitrary **positive weights**.
- **No a priori knowledge** on global parameters (like n or D).
- **Linear in rounds**: $3n_{\max\text{CC}} + D$.
- **Polynomial in steps**: $O(W_{\max} n_{\max\text{CC}}^3 n)$.

- n : total number of nodes
- D : (hop-)diameter of the network
- $n_{\max\text{CC}}$: max. number of nodes in a connected component
- W_{\max} : maximum integer weight

DCDSPM

- [Glacet, Hanusse, Ilcinkas, Johnen, 2014]
 $2n_{\max CC} + D$ rounds but **exponential #steps**

Shortest-path spanning tree (no analyses for #steps)

- [Many papers]

Disjunction

- [Datta, Devismes, Larmore, 2012]
 \approx DCDSPM with uniform weights but **exponential #steps**

BFS-tree (restricted to polynomial #steps)

- [Cournier, Devismes, Villain, 2009]
Not silent, $O(\Delta n^3)$ steps
- [Cournier, Rovedakis, Villain, 2011]
 $O(D^2)$ rounds, $O(n^6)$ steps

Related work

DCDSPM

- [Glacet, Hanusse, Ilcinkas, Johnen, 2014]
 $2n_{\max CC} + D$ rounds but exponential #steps

Shortest-path spanning tree (no analyses for #steps)

- [Many papers]

Disjunction

- [Datta, Devismes, Larmore, 2012]
 \approx DCDSPM with uniform weights but exponential #steps

BFS-tree (restricted to polynomial #steps)

- [Cournier, Devismes, Villain, 2009]
Not silent, $O(\Delta n^3)$ steps
- [Cournier, Rovedakis, Villain, 2011]
 $O(D^2)$ rounds, $O(n^6)$ steps

Related work

DCDSPM

- [Glacet, Hanusse, Ilcinkas, Johnen, 2014]
 $2n_{\max CC} + D$ rounds but **exponential #steps**

Shortest-path spanning tree (no analyses for #steps)

- [Many papers]

Disjunction

- [Datta, Devismes, Larmore, 2012]
 \approx DCDSPM with uniform weights but **exponential #steps**

BFS-tree (restricted to polynomial #steps)

- [Cournier, Devismes, Villain, 2009]
Not silent, $O(\Delta n^3)$ steps
- [Cournier, Rovedakis, Villain, 2011]
 $O(D^2)$ rounds, $O(n^6)$ steps

Related work

DCDSPM

- [Glacet, Hanusse, Ilcinkas, Johnen, 2014]
 $2n_{\max CC} + D$ rounds but **exponential #steps**

Shortest-path spanning tree (no analyses for #steps)

- [Many papers]

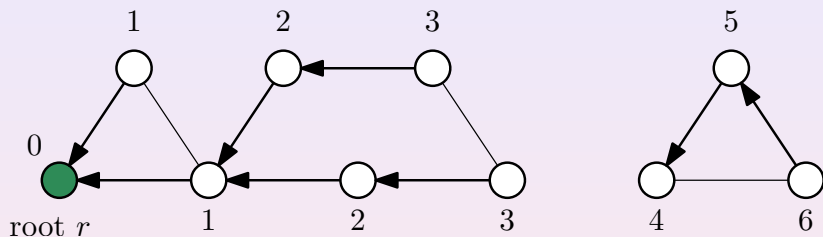
Disjunction

- [Datta, Devismes, Larmore, 2012]
 \approx DCDSPM with uniform weights but **exponential #steps**

BFS-tree (restricted to polynomial #steps)

- [Cournier, Devismes, Villain, 2009]
Not silent, $O(\Delta n^3)$ steps
- [Cournier, Rovedakis, Villain, 2011]
 $O(D^2)$ rounds, $O(n^6)$ steps

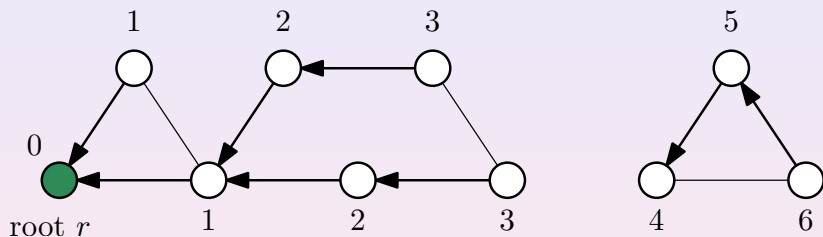
Ideas behind the algorithm



To be pretty cautious...

... but not too much.

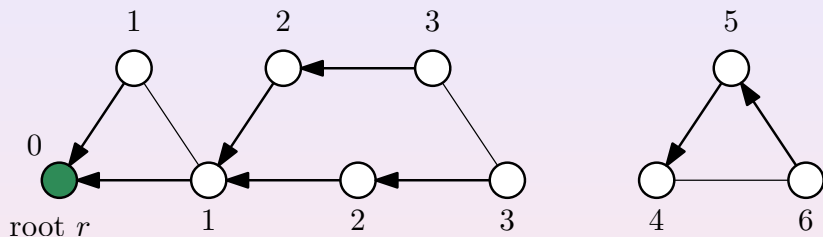
Ideas behind the algorithm



To be pretty cautious...

... but not too much.

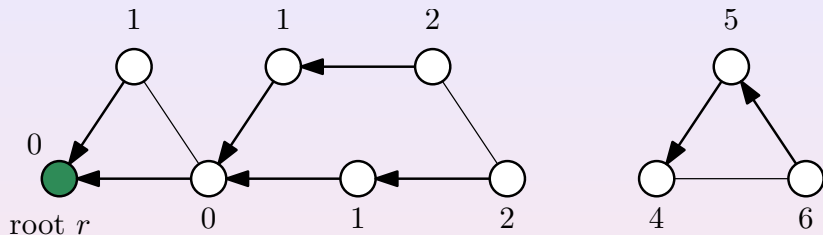
Ideas behind the algorithm



To be pretty cautious...

... but not too much.

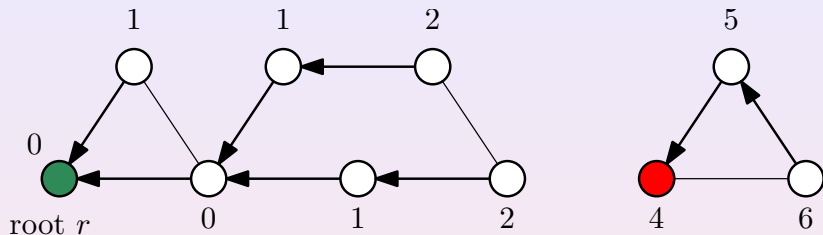
Pretty cautious...



1 wave (1 error state) gives a correct but exponential algorithm
[Glacet, Hanusse, Ilcinkas, Johnen, 2014]

Two waves allow a polynomial number of steps.
But $n_{\max CC}$ additional rounds.

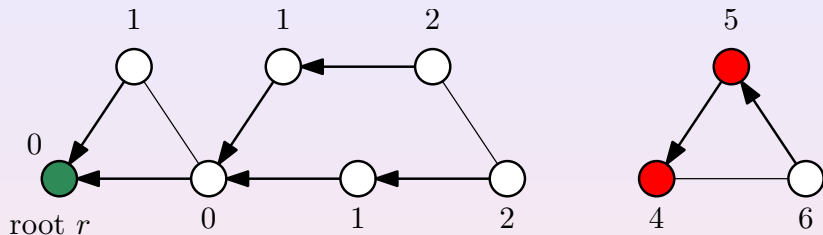
Pretty cautious...



1 wave (1 error state) gives a correct but exponential algorithm
[Glacet, Hanusse, Ilcinkas, Johnen, 2014]

Two waves allow a polynomial number of steps.
But $n_{\max CC}$ additional rounds.

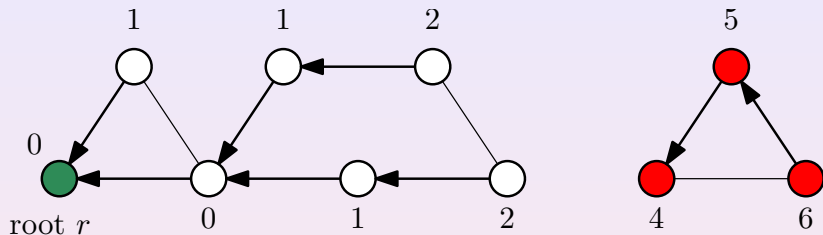
Pretty cautious...



1 wave (1 error state) gives a correct but exponential algorithm
[Glacet, Hanusse, Ilcinkas, Johnen, 2014]

Two waves allow a polynomial number of steps.
But $n_{\max CC}$ additional rounds.

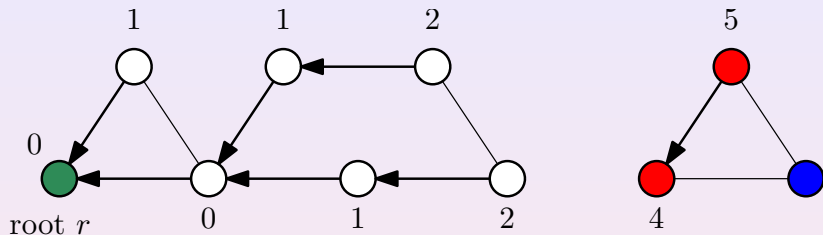
Pretty cautious...



1 wave (1 error state) gives a correct but exponential algorithm
[Glacet, Hanusse, Ilcinkas, Johnen, 2014]

Two waves allow a polynomial number of steps.
But $n_{\max CC}$ additional rounds.

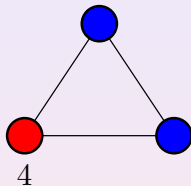
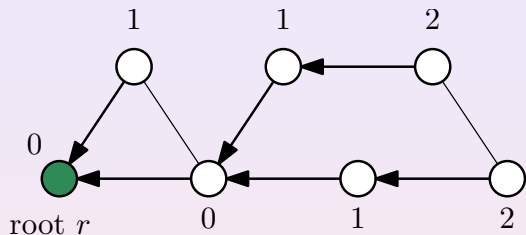
Pretty cautious...



1 wave (1 error state) gives a correct but exponential algorithm
[Glacet, Hanusse, Ilcinkas, Johnen, 2014]

Two waves allow a polynomial number of steps.
But $n_{\max CC}$ additional rounds.

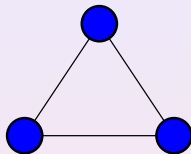
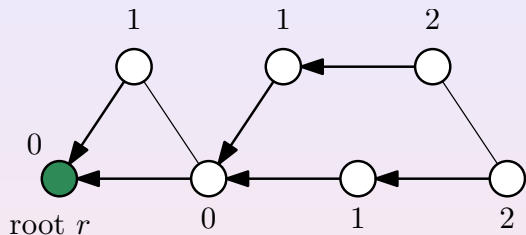
Pretty cautious...



1 wave (1 error state) gives a correct but exponential algorithm
[Glacet, Hanusse, Ilcinkas, Johnen, 2014]

Two waves allow a polynomial number of steps.
But $n_{\max CC}$ additional rounds.

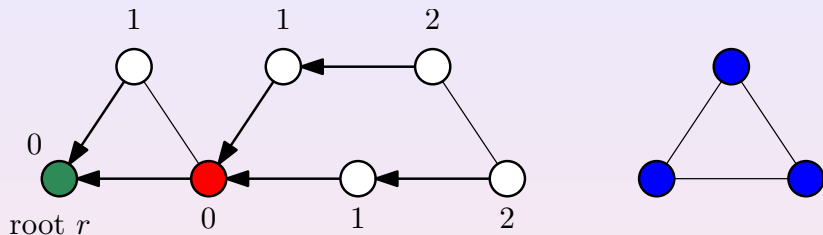
Pretty cautious...



1 wave (1 error state) gives a correct but exponential algorithm
[Glacet, Hanusse, Ilcinkas, Johnen, 2014]

Two waves allow a polynomial number of steps.
But $n_{\max CC}$ additional rounds.

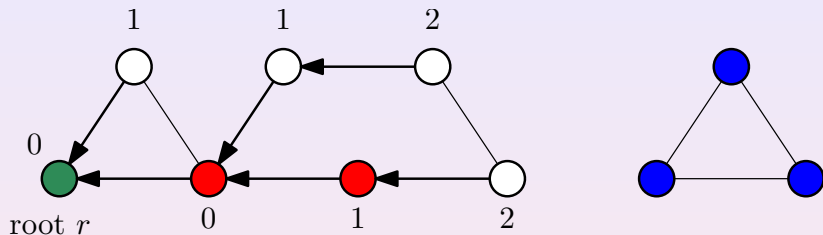
Pretty cautious...



1 wave (1 error state) gives a **correct but exponential** algorithm
[Glacet, Hanusse, Ilcinkas, Johnen, 2014]

Two waves allow a polynomial number of steps.
But $n_{\max CC}$ additional rounds.

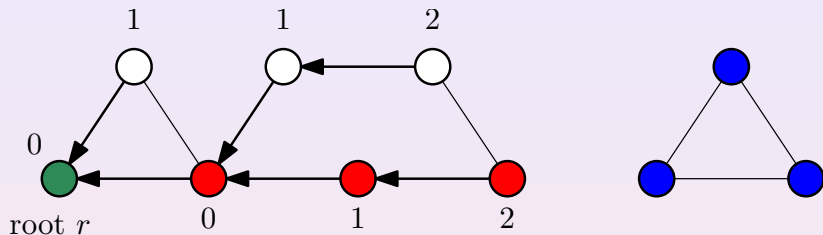
Pretty cautious...



1 wave (1 error state) gives a **correct but exponential** algorithm
[Glacet, Hanusse, Ilcinkas, Johnen, 2014]

Two waves allow a polynomial number of steps.
But $n_{\max CC}$ additional rounds.

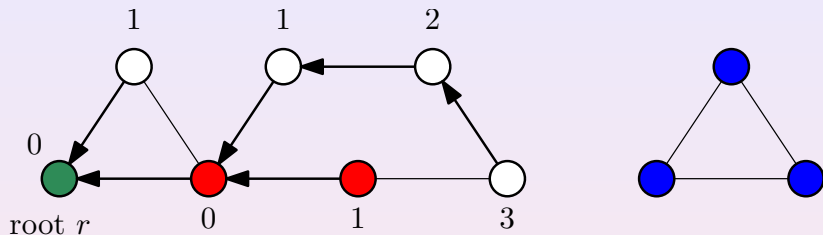
Pretty cautious...



1 wave (1 error state) gives a **correct but exponential** algorithm
[Glacet, Hanusse, Ilcinkas, Johnen, 2014]

Two waves allow a polynomial number of steps.
But $n_{\max CC}$ additional rounds.

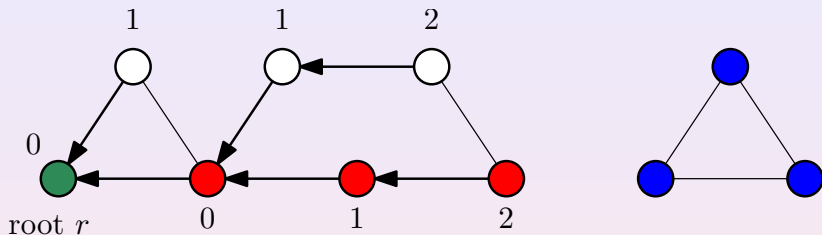
Pretty cautious...



1 wave (1 error state) gives a **correct but exponential** algorithm
[Glacet, Hanusse, Ilcinkas, Johnen, 2014]

Two waves allow a polynomial number of steps.
But $n_{\max CC}$ additional rounds.

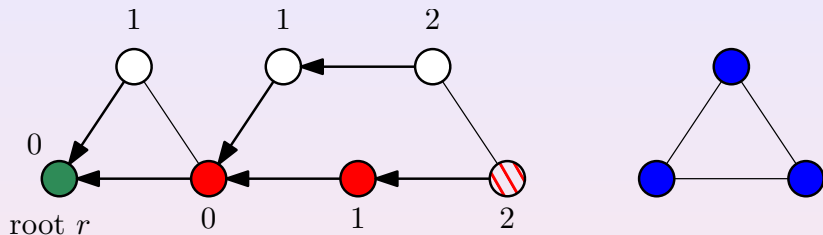
Pretty cautious...



1 wave (1 error state) gives a **correct but exponential** algorithm
[Glacet, Hanusse, Ilcinkas, Johnen, 2014]

Two waves allow a **polynomial number of steps**.
But $n_{\max CC}$ additional rounds.

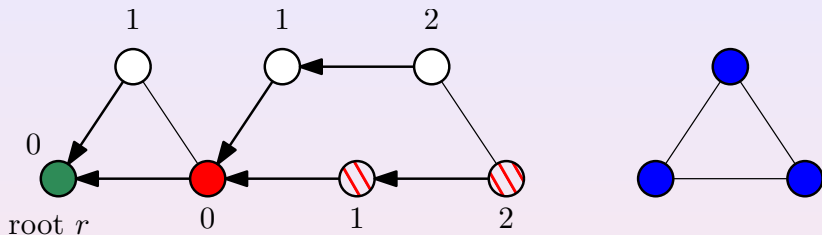
Pretty cautious...



1 wave (1 error state) gives a **correct but exponential** algorithm
[Glacet, Hanusse, Ilcinkas, Johnen, 2014]

Two waves allow a **polynomial number of steps**.
But $n_{\max CC}$ additional rounds.

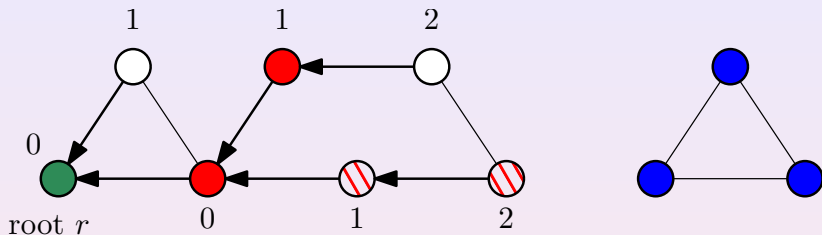
Pretty cautious...



1 wave (1 error state) gives a **correct but exponential** algorithm
[Glacet, Hanusse, Ilcinkas, Johnen, 2014]

Two waves allow a **polynomial number of steps**.
But $n_{\max CC}$ additional rounds.

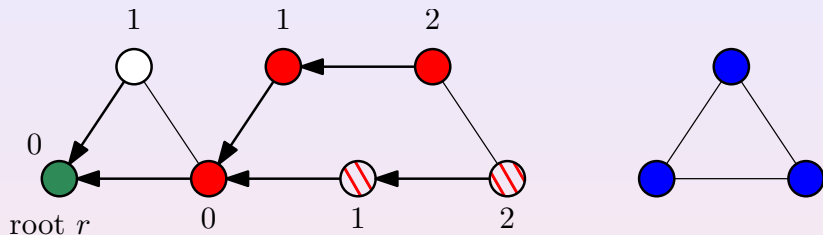
Pretty cautious...



1 wave (1 error state) gives a **correct but exponential** algorithm
[Glacet, Hanusse, Ilcinkas, Johnen, 2014]

Two waves allow a **polynomial number of steps**.
But $n_{\max CC}$ additional rounds.

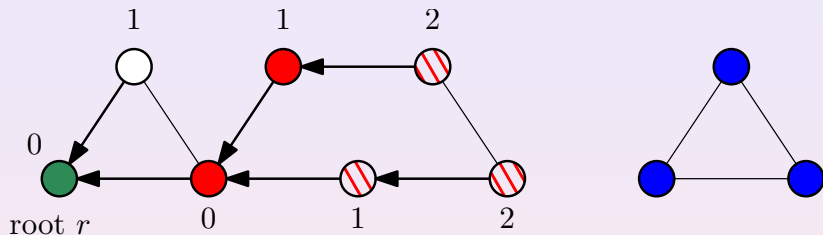
Pretty cautious...



1 wave (1 error state) gives a **correct but exponential** algorithm
[Glacet, Hanusse, Ilcinkas, Johnen, 2014]

Two waves allow a **polynomial number of steps**.
But $n_{\max CC}$ additional rounds.

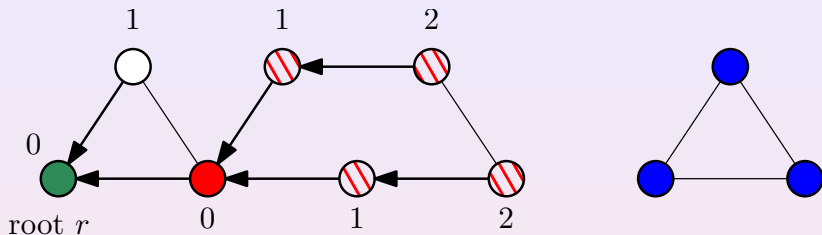
Pretty cautious...



1 wave (1 error state) gives a **correct but exponential** algorithm
[Glacet, Hanusse, Ilcinkas, Johnen, 2014]

Two waves allow a **polynomial number of steps**.
But $n_{\max CC}$ additional rounds.

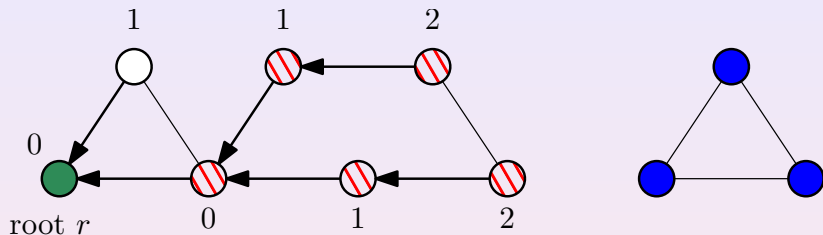
Pretty cautious...



1 wave (1 error state) gives a **correct but exponential** algorithm
[Glacet, Hanusse, Ilcinkas, Johnen, 2014]

Two waves allow a **polynomial number of steps**.
But $n_{\max CC}$ additional rounds.

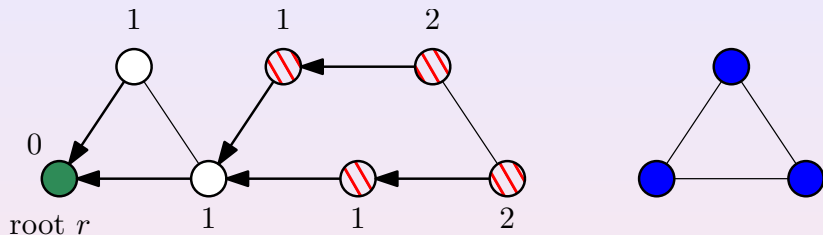
Pretty cautious...



1 wave (1 error state) gives a **correct but exponential** algorithm
[Glacet, Hanusse, Ilcinkas, Johnen, 2014]

Two waves allow a **polynomial number of steps**.
But $n_{\max CC}$ additional rounds.

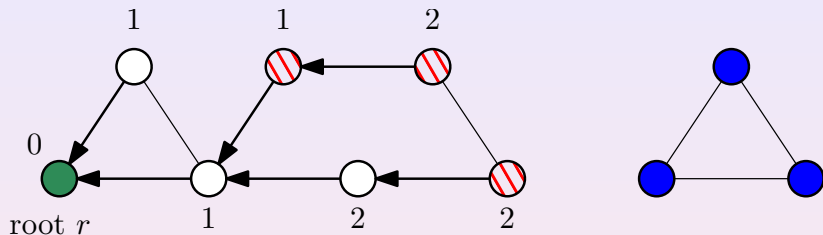
Pretty cautious...



1 wave (1 error state) gives a **correct but exponential** algorithm
[Glacet, Hanusse, Ilcinkas, Johnen, 2014]

Two waves allow a **polynomial number of steps**.
But $n_{\max CC}$ additional rounds.

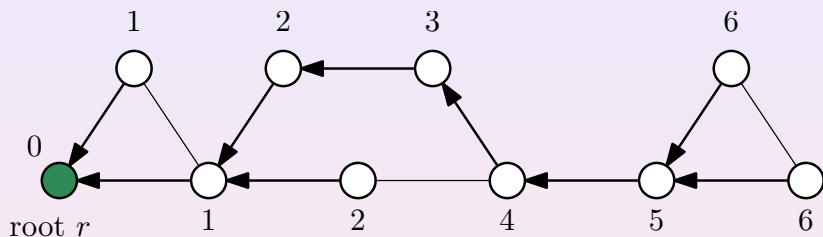
Pretty cautious...



1 wave (1 error state) gives a **correct but exponential** algorithm
[Glacet, Hanusse, Ilcinkas, Johnen, 2014]

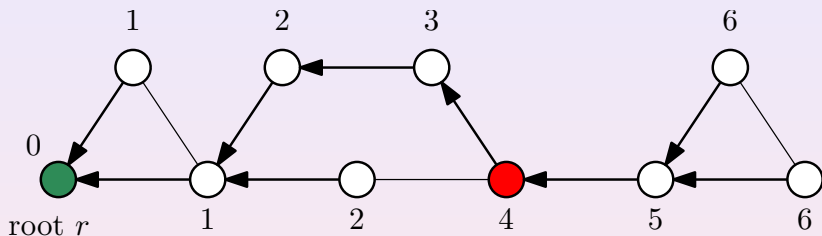
Two waves allow a **polynomial number of steps**.
But $n_{\max CC}$ additional rounds.

... but not too much



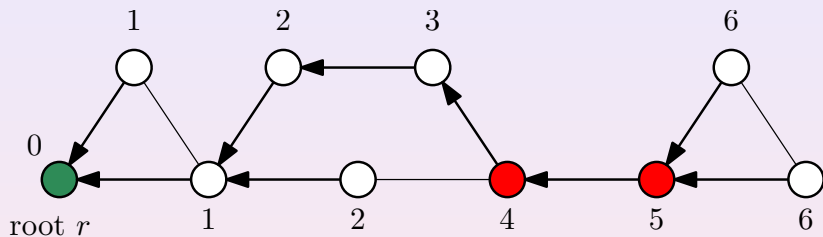
Not directly improving distances leads to $\Omega(D^2)$ rounds.

... but not too much



Not directly improving distances leads to $\Omega(D^2)$ rounds.

... but not too much



Not directly improving distances leads to $\Omega(D^2)$ rounds.

Formal algorithm

Algorithm 2: Code of RSP for any process $u \neq r$

Variables:

$st_u \in \{I, C, EB, EF\}$

$par_u \in Lbl$

$d_u \in \mathbb{N}^*$

Predicates:

$P_reset(u) \equiv st_u = EF \wedge abRoot(u)$

$P_correction(u) \equiv (\exists v \in \Gamma(u) \mid st_v = C \wedge d_v + \omega(u, v) < d_u)$

Macro:

$computePath(u) : par_u := \operatorname{argmin}_{(v \in \Gamma(u) \wedge st_v = C)} (d_v + \omega(u, v));$
 $d_u := d_{par_u} + \omega(u, par_u);$
 $st_u := C$

Rules

$\mathbf{R}_C(u) : st_u = C \wedge P_correction(u) \rightarrow computePath(u)$

$\mathbf{R}_{EB}(u) : st_u = C \wedge \neg P_correction(u) \wedge (abRoot(u) \vee st_{par_u} = EB) \rightarrow st_u := EB$

$\mathbf{R}_{EF}(u) : st_u = EB \wedge (\forall v \in children(u) \mid st_v = EF) \rightarrow st_u := EF$

$\mathbf{R}_I(u) : P_reset(u) \wedge (\forall v \in \Gamma(u) \mid st_v \neq C) \rightarrow st_u := I$

$\mathbf{R}_R(u) : (P_reset(u) \vee st_u = I) \wedge (\exists v \in \Gamma(u) \mid st_v = C) \rightarrow computePath(u)$

Thank you
for your attention