Maintaining a Spanning Forest in Highly Dynamic Networks: The synchronous case

M. Barjon, A. Casteigts, S. Chaumette, C. Johnen, Y. M. Neggaz

University of Bordeaux

OPODIS'14
Cortina, Italy

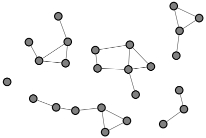
Dynamic networks

Highly dynamic networks.

Ex: 

How changes are perceived?

- ~~Faults and Failures?~~
- Nature of the system. Change is normal.
- The network is partitioned most of the time.



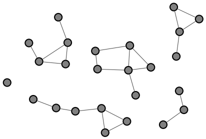
Dynamic networks

Highly dynamic networks.

Ex: 

How changes are perceived?

- ~~Faults and Failures?~~
- Nature of the system. Change is normal.
- The network is partitioned most of the time.



Example scenario
(Wireless mobile robots)

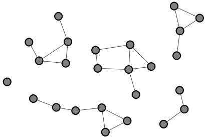
Dynamic networks

Highly dynamic networks.

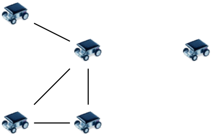
Ex: 

How changes are perceived?

- ~~Faults and Failures?~~
- Nature of the system. Change is normal.
- The network is partitioned most of the time.



Example scenario
(Wireless mobile robots)



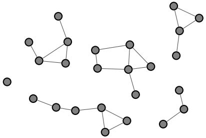
Dynamic networks

Highly dynamic networks.

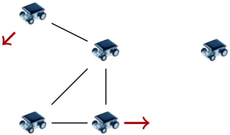
Ex: 

How changes are perceived?

- ~~Faults and Failures?~~
- Nature of the system. Change is normal.
- The network is partitioned most of the time.



Example scenario
(Wireless mobile robots)



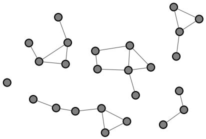
Dynamic networks

Highly dynamic networks.

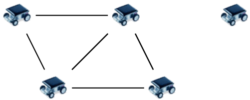
Ex: 

How changes are perceived?

- ~~Faults and Failures?~~
- Nature of the system. Change is normal.
- The network is partitioned most of the time.



Example scenario
(Wireless mobile robots)



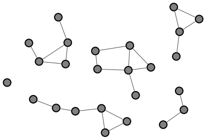
Dynamic networks

Highly dynamic networks.

Ex: 

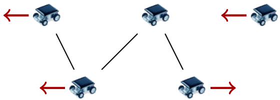
How changes are perceived?

- ~~Faults and Failures?~~
- Nature of the system. Change is normal.
- The network is partitioned most of the time.



Example scenario

(Wireless mobile robots)



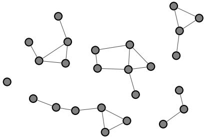
Dynamic networks

Highly dynamic networks.

Ex: 

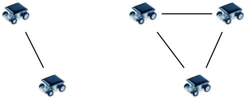
How changes are perceived?

- ~~Faults and Failures?~~
- Nature of the system. Change is normal.
- The network is partitioned most of the time.



Example scenario

(Wireless mobile robots)



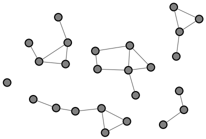
Dynamic networks

Highly dynamic networks.

Ex: 

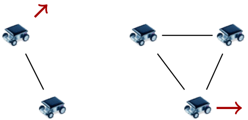
How changes are perceived?

- ~~Faults and Failures?~~
- Nature of the system. Change is normal.
- The network is partitioned most of the time.



Example scenario

(Wireless mobile robots)



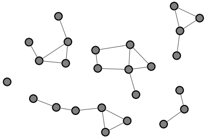
Dynamic networks

Highly dynamic networks.

Ex: 

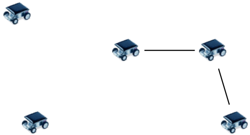
How changes are perceived?

- ~~Faults and Failures?~~
- Nature of the system. Change is normal.
- The network is partitioned most of the time.



Example scenario

(Wireless mobile robots)



Redefinition of problems in highly dynamic networks

Ex. with two classical problems : ELECTION, SPANNINGTREE

Usual definition (static networks)

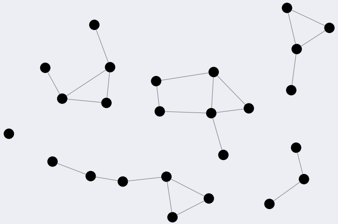
Leader election — Distinguishing one node among all.

Spanning tree — Selecting a cycle-free set of edges that interconnects all nodes.

Redefinition of problems in highly dynamic networks

Ex. with two classical problems : LEADERELECTION, SPANNINGTREE

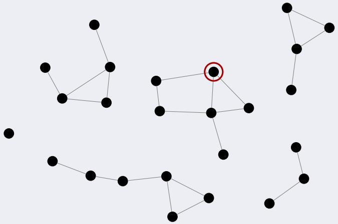
In highly dynamic networks



How to define them?

Ex. with two classical problems : LEADERELECTION, SPANNINGTREE

In highly dynamic networks



How to define them?

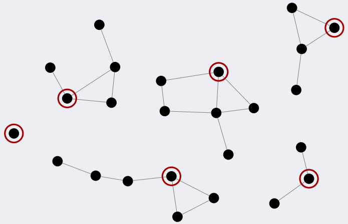
→ Two options:

1. Global solution, which holds over time (computed once)

Redefinition of problems in highly dynamic networks

Ex. with two classical problems : LEADERELECTION, SPANNINGTREE

In highly dynamic networks



How to define them?

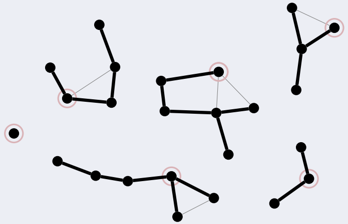
→ Two options:

1. Global solution, which holds over time (computed once)
2. Component solutions, contemporaneous (constantly maintained)

Redefinition of problems in highly dynamic networks

Ex. with two classical problems : LEADERELECTION, SPANNINGTREE

In highly dynamic networks



How to define them?

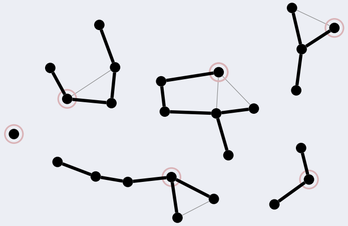
→ Two options:

1. Global solution, which holds over time (computed once)
2. Component solutions, contemporaneous (constantly maintained)

Redefinition of problems in highly dynamic networks

Ex. with two classical problems : LEADERELECTION, SPANNINGTREE

In highly dynamic networks



How to define them?

→ Two options:

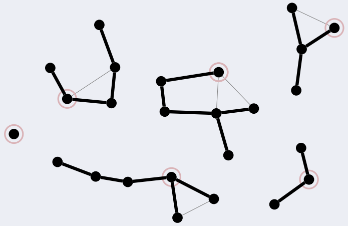
1. Global solution, which holds over time (computed once)
2. Component solutions, contemporaneous (constantly maintained)

→ This line of work : Maintaining a forest of spanning trees (and a leader, incidently).

Redefinition of problems in highly dynamic networks

Ex. with two classical problems : LEADERELECTION, SPANNINGTREE

In highly dynamic networks



How to define them?

→ Two options:

1. Global solution, which holds over time (computed once)
2. Component solutions, contemporaneous (constantly maintained)

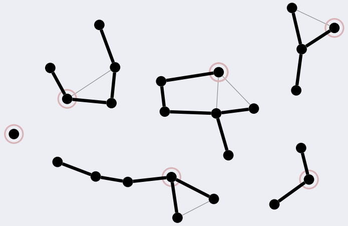
→ This line of work : Maintaining a forest of spanning trees (and a leader, incidently).

What assumptions?

Redefinition of problems in highly dynamic networks

Ex. with two classical problems : LEADERELECTION, SPANNINGTREE

In highly dynamic networks



How to define them?

→ Two options:

1. Global solution, which holds over time (computed once)
2. Component solutions, contemporaneous (constantly maintained)

→ This line of work : Maintaining a forest of spanning trees (and a leader, incidently).

What assumptions?

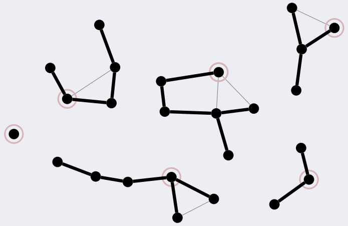
- No stability period
- No restriction on the rate of events

$\Bigg\}\implies$ No recomputation from scratch.
& Decision should be purely local!

Redefinition of problems in highly dynamic networks

Ex. with two classical problems : LEADERELECTION, SPANNINGTREE

In highly dynamic networks



How to define them?

→ Two options:

1. Global solution, which holds over time (computed once)
2. Component solutions, contemporaneous (constantly maintained)

→ This line of work : Maintaining a forest of spanning trees (and a leader, incidently).

What assumptions?

- No stability period
- No restriction on the rate of events

$\Longrightarrow$ No recomputation from scratch.
& Decision should be purely local!

What can we still expect in such a setting?

Related works (1) – Weak dynamism

- Occasional failures (self-stabilization):

  Ex: *[Burman and Kutten, 2007]* ... *[Gërtner, 2013]* (Survey)

  - *Minimum* spanning tree
  - When the graph changes:
    $\rightarrow$ Reset and recompute everything from scratch

Related works (1) – Weak dynamism

- Occasional failures (self-stabilization):

  Ex: *[Burman and Kutten, 2007]* ... *[Gërtner, 2013]* (Survey)

  - *Minimum* spanning tree
  - When the graph changes:
    $\rightarrow$ Reset and recompute everything from scratch

- A bit more "dynamic": *[Abbas et al. 2003]*, *[Baala et al. 2006]* (based on coalescing random walks)
  - *Non-minimum* (rooted) spanning tree
  - When the tree is impacted by a graph change
    $\Longrightarrow$ Reset and recomputes the *orphan* part
  - If there is no token left:
    $\Longrightarrow$ Regenerates one based on expected cover time ($O(n^3)$ + n is known)

Related works (2) – Mild dynamism

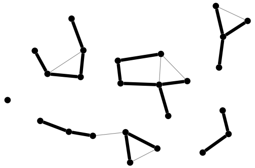
- Mild dynamism: *[Bernard et al. 2013]*
  - Same coalescing process as before, but then..
  - ..the tree keeps being redefined continuously as the token(s) move
    $\Longrightarrow$ Tolerates slow dynamics
  - If there is no token left:
    $\Longrightarrow$ regenerates one based on expected cover time ($O(n^3)$ + n is known)

Related works (2) – Mild dynamism

- Mild dynamism: *[Bernard et al. 2013]*
  - Same coalescing process as before, but then..
  - ..the tree keeps being redefined continuously as the token(s) move
    $\Longrightarrow$ Tolerates slow dynamics
  - If there is no token left:
    $\Longrightarrow$ regenerates one based on expected cover time ($O(n^3)$ + n is known)
- Strong(er) dynamicity: *[Awerbuch et al. 2008]*
  - *Minimum* spanning tree
  - When the graph changes:
    $\rightarrow$ *updates* the previous solution in $O(n)$ time & message
  - If high rate of change:
    $\Longrightarrow$ events are queued and processed one after another
    ($\rightarrow$ Implicitely assumes that strong dynamism is episodical.)

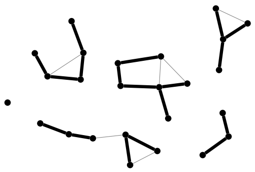
Related works (3) – Unrestricted dynamism

The spanning forest principle *[C. et al. 2013]*



- *Non-minimum* (rooted) spanning trees
- Purely localized, instant decision
- No restriction on the dynamics

The spanning forest principle *[C. et al. 2013]*

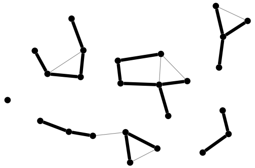


- *Non-minimum* (rooted) spanning trees
- Purely localized, instant decision
- No restriction on the dynamics

$\rightarrow$ Coarse-grain model (atomic pairwise interaction)
inspired from graph relabellings systems *[Litovsky et al., 1999]*
($\sim$ *population protocols*)

Related works (3) – Unrestricted dynamism

The spanning forest principle *[C. et al. 2013]*

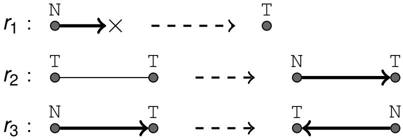


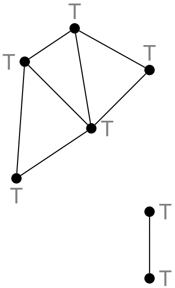
- *Non-minimum* (rooted) spanning trees
- Purely localized, instant decision
- No restriction on the dynamics

→ Coarse-grain model (atomic pairwise interaction)
inspired from graph relabellings systems *[Litovsky et al., 1999]*
(∼ *population protocols*)

Can be seen as a general (i.e. abstract) principle, explained next.

3 rules :





initial states:

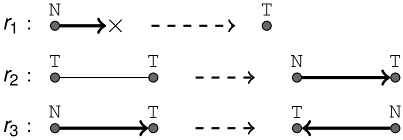
- T for every node,

meaning of the states:

- T: a token is on this node
- N: no token is on this node
- $\rightarrow$: relation from child to parent

General principle [C. et al. 2013]

3 rules :

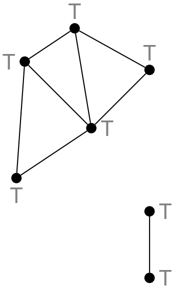


initial states:

- T for every node,

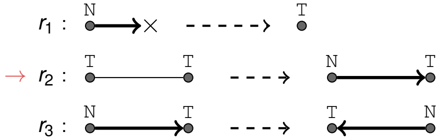
meaning of the states:

- T: a token is on this node
- N: no token is on this node
- $\rightarrow$: relation from child to parent



General principle *[C. et al. 2013]*

3 rules :

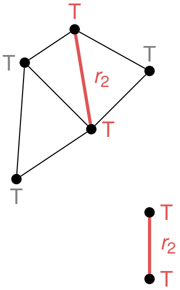


<u>initial states:</u>

- T for every node,

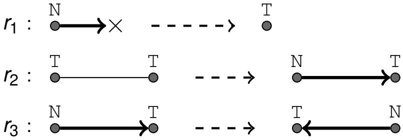
<u>meaning of the states:</u>

- T: a token is on this node
- N: no token is on this node
- → : relation from child to parent



General principle [C. et al. 2013]

3 rules :

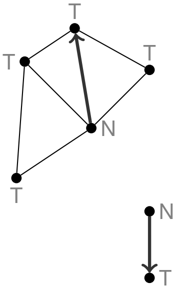


<u>initial states:</u>

- T for every node,

<u>meaning of the states:</u>

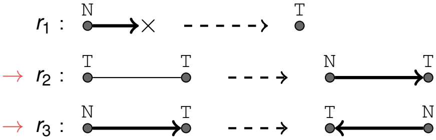
- T: a token is on this node
- N: no token is on this node
- $\rightarrow$: relation from child to parent

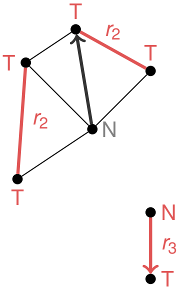


General principle

[C. et al. 2013]

3 rules :





initial states:

- T for every node,

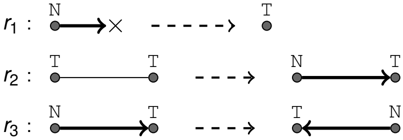
meaning of the states:

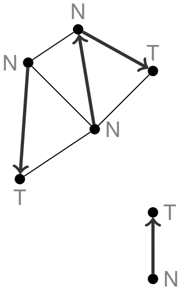
- T: a token is on this node
- N: no token is on this node
- $\rightarrow$: relation from child to parent

General principle

[C. et al. 2013]

3 rules :





initial states:

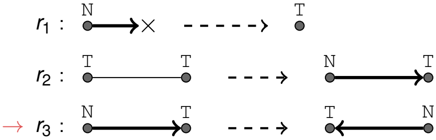
- T for every node,

meaning of the states:

- T: a token is on this node
- N: no token is on this node
- $\rightarrow$: relation from child to parent

General principle *[C. et al. 2013]*

3 rules :

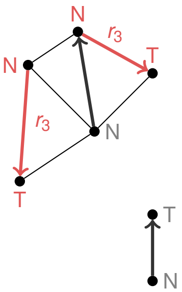


initial states:

- T for every node,

meaning of the states:

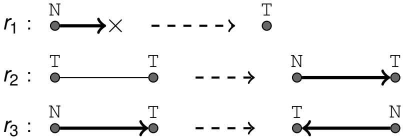
- T: a token is on this node
- N: no token is on this node
- $\rightarrow$: relation from child to parent

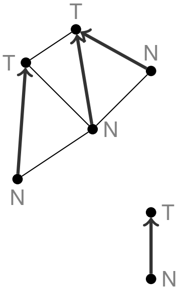


General principle

[C. et al. 2013]

3 rules :





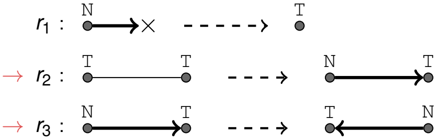
initial states:

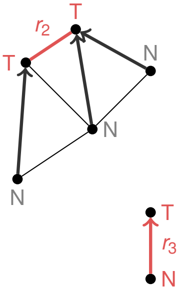
- T for every node,

meaning of the states:

- T: a token is on this node
- N: no token is on this node
- $\rightarrow$: relation from child to parent

3 rules :





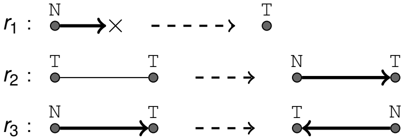
initial states:

- T for every node,

meaning of the states:

- T: a token is on this node
- N: no token is on this node
- → : relation from child to parent

3 rules :

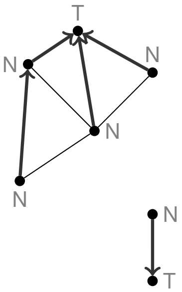


<u>initial states:</u>

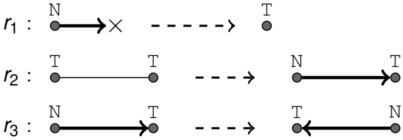
- T for every node,

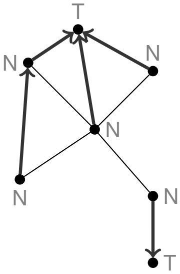
<u>meaning of the states:</u>

- T: a token is on this node
- N: no token is on this node
- $\rightarrow$: relation from child to parent



3 rules :





<u>initial states:</u>

- T for every node,

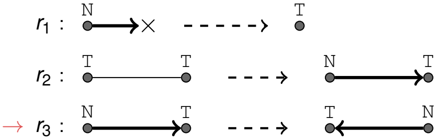
<u>meaning of the states:</u>

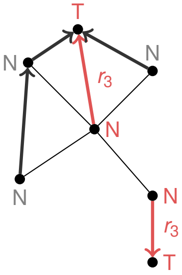
- T: a token is on this node
- N: no token is on this node
- $\rightarrow$: relation from child to parent

General principle

[C. et al. 2013]

3 rules :





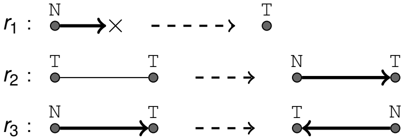
<ins>initial states:</ins>

- $\mathtt{T}$ for every node,

<ins>meaning of the states:</ins>

- $\mathtt{T}$: a token is on this node
- $\mathtt{N}$: no token is on this node
- $\rightarrow$: relation from child to parent

3 rules :

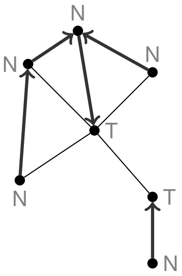


initial states:

- T for every node,

meaning of the states:

- T: a token is on this node
- N: no token is on this node
- $\rightarrow$: relation from child to parent

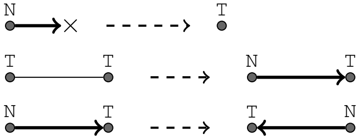


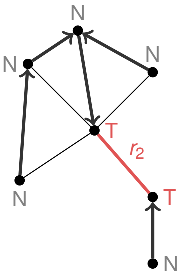
3 rules :

r_1 :

$\rightarrow$ r_2 :

r_3 :





initial states:

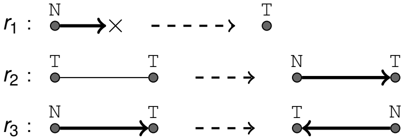
- T for every node,

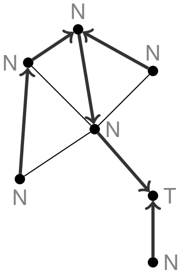
meaning of the states:

- T: a token is on this node
- N: no token is on this node
- $\rightarrow$: relation from child to parent

General principle *[C. et al. 2013]*

3 rules :





<u>initial states:</u>

- T for every node,

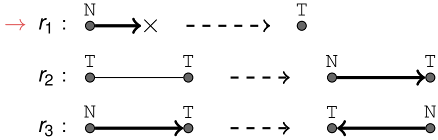
<u>meaning of the states:</u>

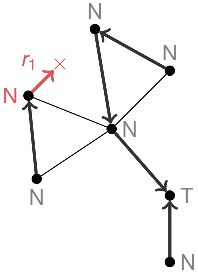
- T: a token is on this node
- N: no token is on this node
- $\rightarrow$: relation from child to parent

General principle

[C. et al. 2013]

3 rules :





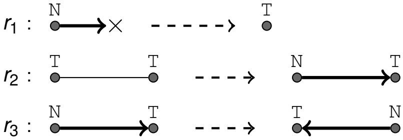
initial states:

- T for every node,

meaning of the states:

- T: a token is on this node
- N: no token is on this node
- $\rightarrow$: relation from child to parent

3 rules :

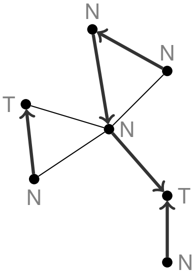


initial states:

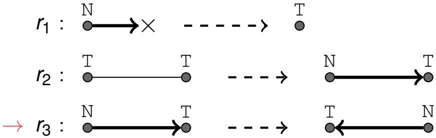
- T for every node,

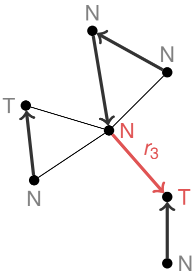
meaning of the states:

- T: a token is on this node
- N: no token is on this node
- $\rightarrow$: relation from child to parent



3 rules :





<u>initial states:</u>

- T for every node,

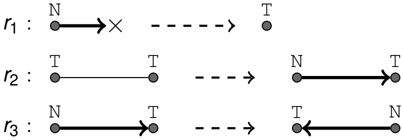
<u>meaning of the states:</u>

- T: a token is on this node
- N: no token is on this node
- $\rightarrow$: relation from child to parent

General principle

[C. et al. 2013]

3 rules :

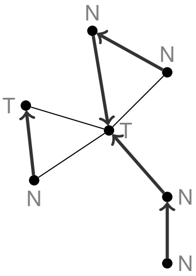


<u>initial states:</u>

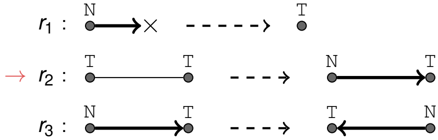
- T for every node,

<u>meaning of the states:</u>

- T: a token is on this node
- N: no token is on this node
- $\rightarrow$: relation from child to parent



3 rules :

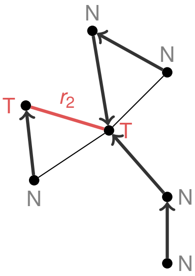


initial states:

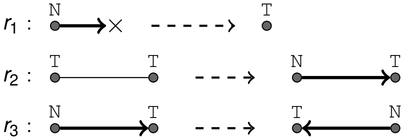
- T for every node,

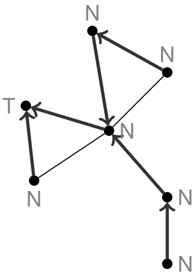
meaning of the states:

- T: a token is on this node
- N: no token is on this node
- $\rightarrow$: relation from child to parent



3 rules :





initial states:

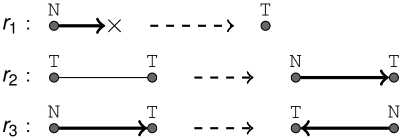
- T for every node,

meaning of the states:

- T: a token is on this node
- N: no token is on this node
- $\rightarrow$: relation from child to parent

General principle *[C. et al. 2013]*

3 rules :

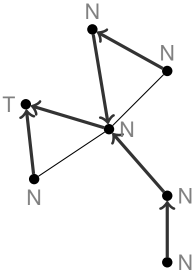


initial states:

- T for every node,

meaning of the states:

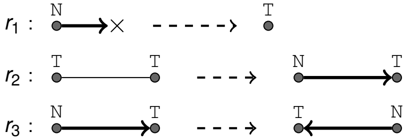
- T: a token is on this node
- N: no token is on this node
- $\rightarrow$: relation from child to parent



Properties that hold permanently:

General principle [C. et al. 2013]

3 rules :

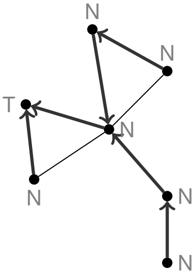


initial states:

- T for every node,

meaning of the states:

- T: a token is on this node
- N: no token is on this node
- $\rightarrow$: relation from child to parent

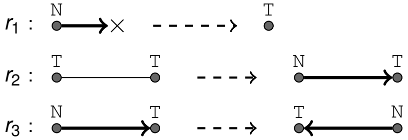


Properties that hold permanently:

- Each node belongs to exactly one tree
- There is exactly one token per tree
- There are no cycles

General principle [C. et al. 2013]

3 rules :

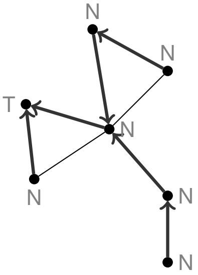


initial states:

- T for every node,

meaning of the states:

- T: a token is on this node
- N: no token is on this node
- $\rightarrow$: relation from child to parent



Properties that hold permanently:

- Each node belongs to exactly one tree
- There is exactly one token per tree
- There are no cycles

How about performance?

- Convergence is not expected

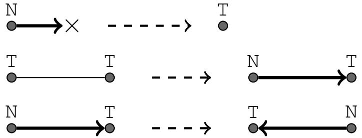
$\rightarrow$ metric of interest: # trees per components (in normal regime)

3 rules :

r_1 :

r_2 :

r_3 :

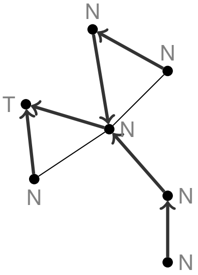


<u>initial states:</u>

- T for every node,

<u>meaning of the states:</u>

- T: a token is on this node
- N: no token is on this node
- $\rightarrow$: relation from child to parent



Properties that hold <u>permanently</u>:

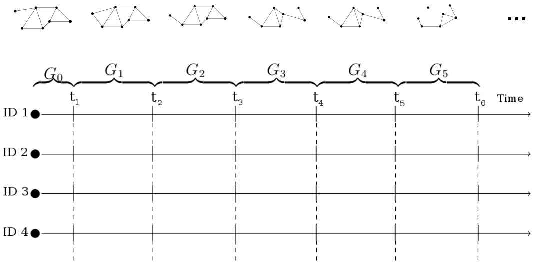
- Each node belongs to exactly one tree
- There is exactly one token per tree
- There are no cycles

How about performance?

- Convergence is not expected

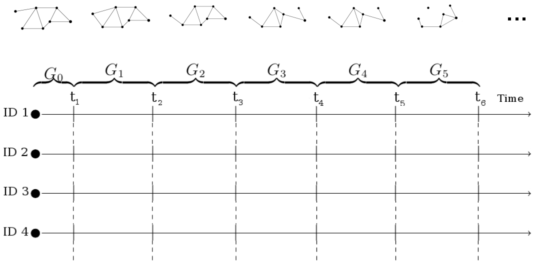
$\rightarrow$ metric of interest: # trees per components (in normal regime)

OK, now the message passing version...


...
G_0
G_1
G_2
G_3
G_4
G_5
t_1
t_2
t_3
t_4
t_5
t_6
Time
ID 1
ID 2
ID 3
ID 4

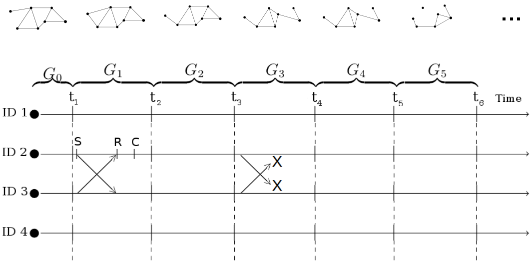
In each round :

1. Send (S) (local broadcast)
2. Receive (R) (reciprocal)
3. Compute (C)



In each round :

1. Send (S) (local broadcast)
2. Receive (R) (reciprocal)
3. Compute (C)



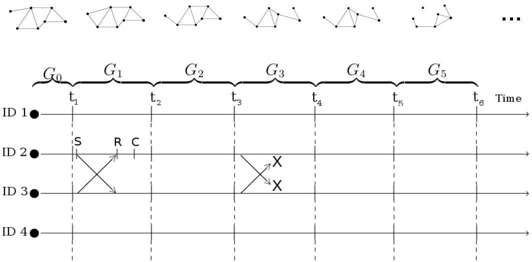
In each round :

1. Send (S) (local broadcast)
2. Receive (R) (reciprocal)
3. Compute (C)

States of the nodes

- After round i:

→ State of node v, written $s_i(v)$



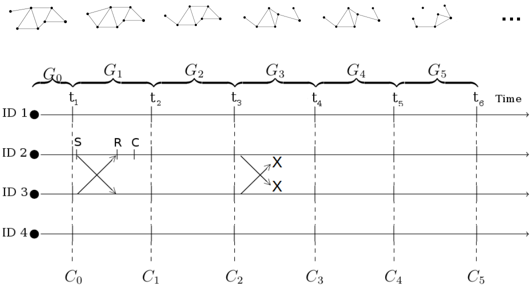
Fully synchronous message passing model *[Kuhn et al., STOC 2010]*

In each round :

1. Send (S) (local broadcast)
2. Receive (R) (reciprocal)
3. Compute (C)

States of the nodes

- After round i:
  - → State of node v, written $s_i(v)$
  - → Global state (configuration): $C_i = \{s_i(v) : v \in V\}$

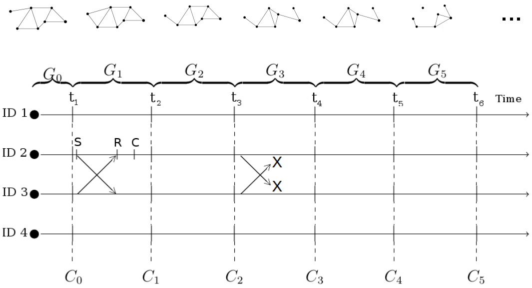


In each round :

1. Send (S) (local broadcast)
2. Receive (R) (reciprocal)
3. Compute (C)

States of the nodes

- After round i:
  - → State of node v, written $s_i(v)$
  - → Global state (configuration): $C_i = \{s_i(v) : v \in V\}$



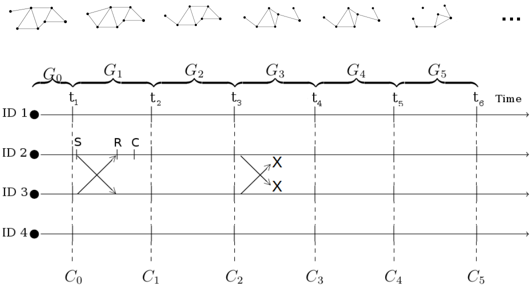
+assumption of unique identifiers

In each round :

1. Send (S) (local broadcast)
2. Receive (R) (reciprocal)
3. Compute (C)

States of the nodes

- After round i:
- → State of node v, written $s_i(v)$
- → Global state (configuration): $C_i = \{s_i(v) : v \in V\}$



+assumption of unique identifiers

Properties that used to hold permanently in the coarse-grain model...
... become properties that hold at the end of each round (i.e. in the C_is).

Elements of the algorithm

Local state of a node

- `parent`
- `children`
- `status (T|N)`
  has token or not
- `score`
  (discussed later)
- `neighbors`
  in the current round
- `contender`
  neighbor to be selected as parent

Elements of the algorithm

Local state of a node (initial value)

- `parent` ($\perp$)
- `children` ($\emptyset$)
- `status (T|N)` (`T`)
  has token or not
- `score` (`ID`)
  (discussed later)
- `neighbors` ($\emptyset$)
  in the current round
- `contender` ($\perp$)
  neighbor to be selected as parent

Elements of the algorithm

Local state of a node (initial value)

- `parent` ($\perp$)
- `children` ($\emptyset$)
- `status (T|N)` (`T`)
  has token or not
- `score` (`ID`)
  (discussed later)
- `neighbors` ($\emptyset$)
  in the current round
- `contender` ($\perp$)
  neighbor to be selected as parent

Types of messages

3 types of messages:

- `HELLO (h)`
  default message
- `SELECT (S)`
  selects of a neighbor as parent
  ($\rightarrow$ tree merge)
- `FLIP (F)`
  gives the token to a child
  ($\rightarrow$ token circulation)

All can be overheard (and must be so).

Elements of the algorithm

Local state of a node (initial value)

- `parent` ($\perp$)
- `children` ($\emptyset$)
- `status (T|N)` (`T`)
  has token or not
- `score` (`ID`)
  (discussed later)
- `neighbors` ($\emptyset$)
  in the current round
- `contender` ($\perp$)
  neighbor to be selected as parent

Types of messages

3 types of messages:

- `HELLO (h)`
  default message
- `SELECT (S)`
  selects of a neighbor as parent
  ($\rightarrow$ tree merge)
- `FLIP (F)`
  gives the token to a child
  ($\rightarrow$ token circulation)

All can be overheard (and must be so).

Structure of a message

Tuple `(ID, status, type, targetID, score)`

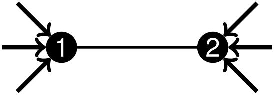
Some examples:

$\rightarrow$ (`myID`, `N`, `HELLO`, $\perp$, `score`)	(default message)
$\rightarrow$ (`myID`, `T`, `SELECT`, `contendedID`, `score`)	(tree merge)
$\rightarrow$ (`myID`, `T`, `FLIP`, `childID`, `score`)	(token circulation)

Local operations (1): Tree merge

Merging of two trees (`SELECT` message). Survive of the fittest (largest).

2 cases :

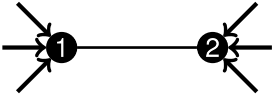


C_{i-1}

Local operations (1): Tree merge

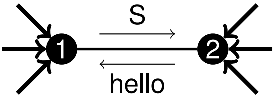
Merging of two trees (`SELECT` message). Survive of the fittest (largest).

2 cases :



C_{i-1}

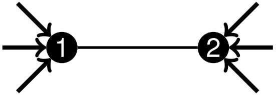
$e \in G_i$



Local operations (1): Tree merge

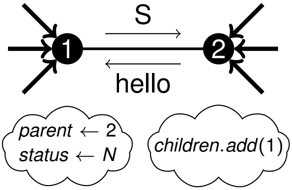
Merging of two trees (`SELECT` message). Survive of the fittest (largest).

2 cases :



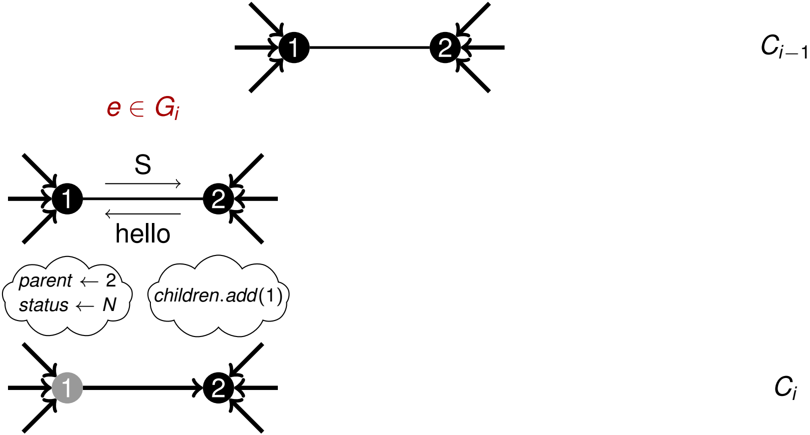
C_{i-1}

$e \in G_i$



Merging of two trees (`SELECT` message). Survive of the fittest (largest).

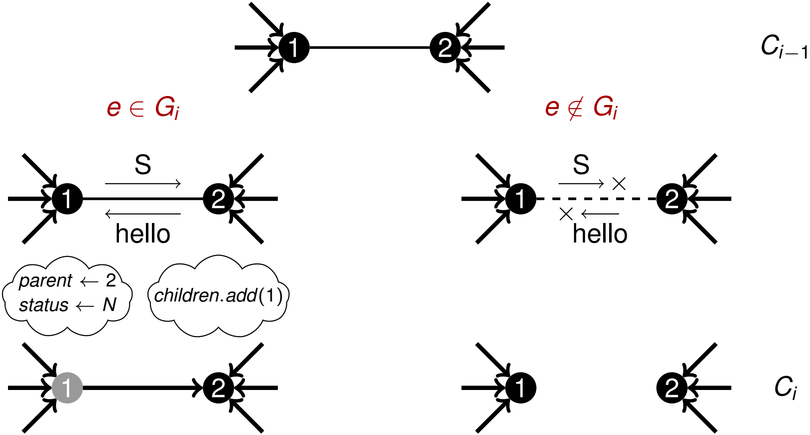
2 cases :



Local operations (1): Tree merge

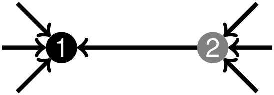
Merging of two trees (`SELECT` message). Survive of the fittest (largest).

2 cases :



Local operations (2): Token circulation

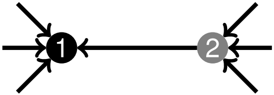
Circulation of the token, within the tree (FLIP messages). The child is taken at random. 2 cases:



C_{i-1}

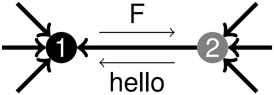
Local operations (2): Token circulation

Circulation of the token, within the tree (FLIP messages). The child is taken at random. 2 cases:



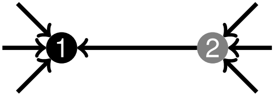
C_{i-1}

$e \in G_i$



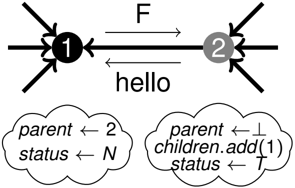
Local operations (2): Token circulation

Circulation of the token, within the tree (`FLIP` messages). The child is taken at random. 2 cases:

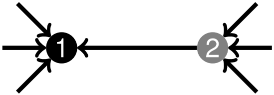


C_{i-1}

$e \in G_i$

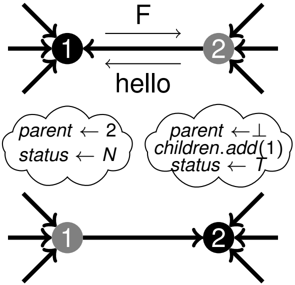


Circulation of the token, within the tree (`FLIP` messages). The child is taken at random. 2 cases:



C_{i-1}

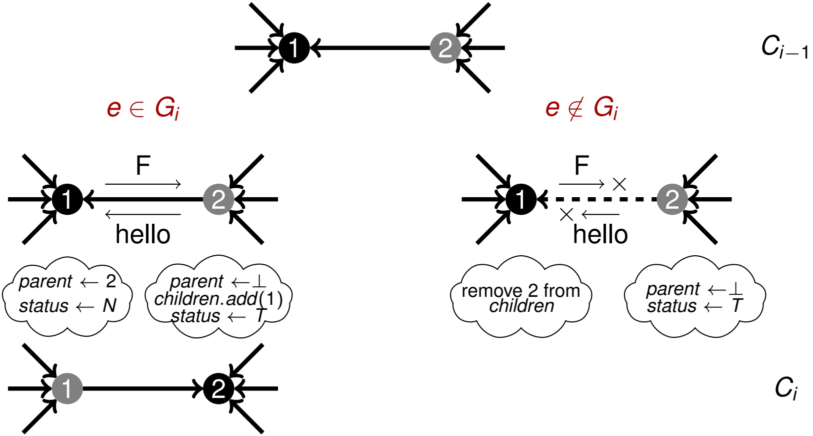
$e \in G_i$



C_i

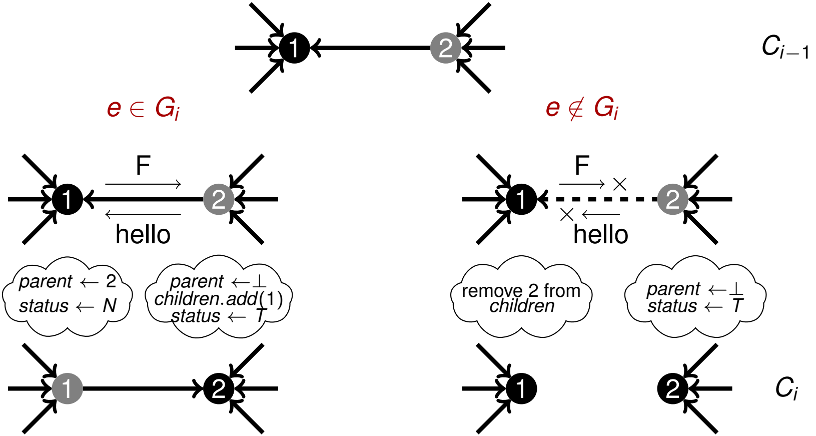
Local operations (2): Token circulation

Circulation of the token, <u>within the tree</u> (`FLIP` messages). The child is taken at random. 2 cases:



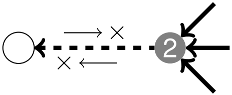
Local operations (2): Token circulation

Circulation of the token, within the tree (FLIP messages). The child is taken at random. 2 cases:



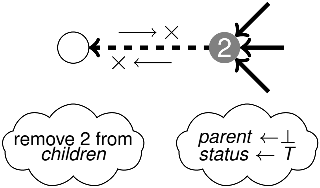
Local operations (3): Token regeneration

Local regeneration of token.



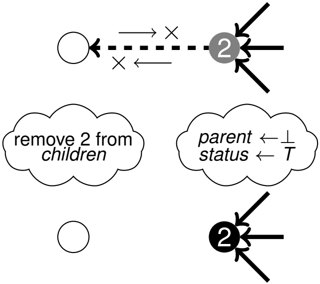
Local operations (3): Token regeneration

Local regeneration of token.



Local operations (3): Token regeneration

Local regeneration of token.

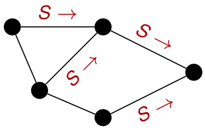


Loss of atomicity:

- A node can be involved in several operations at the same time
  $\Rightarrow$ arbitrary long chain of selections

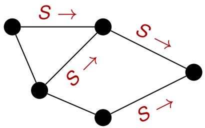
Loss of atomicity:

- A node can be involved in several operations at the same time
  $\Rightarrow$ arbitrary long chain of selections



Loss of atomicity:

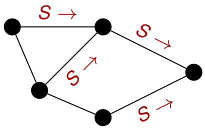
- A node can be involved in several operations at the same time
  $\Rightarrow$ arbitrary long chain of selections



- Initial merging process is faster, but

Loss of atomicity:

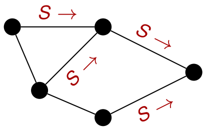
- A node can be involved in several operations at the same time
  $\Rightarrow$ arbitrary long chain of selections



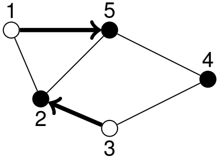
- Initial merging process is faster, but
- tricky configurations :

Loss of atomicity:

- A node can be involved in several operations at the same time
  ⇒ arbitrary long chain of selections



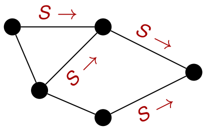
- Initial merging process is faster, but
- tricky configurations :



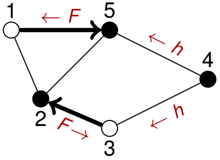
Main difficulty (→ the *score* technique)

Loss of atomicity:

- A node can be involved in several operations at the same time
  ⇒ arbitrary long chain of selections

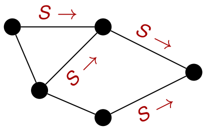


- Initial merging process is faster, but
- tricky configurations :

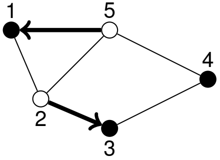


Loss of atomicity:

- A node can be involved in several operations at the same time
  ⇒ arbitrary long chain of selections



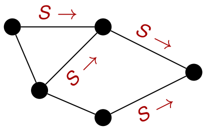
- Initial merging process is faster, but
- tricky configurations :



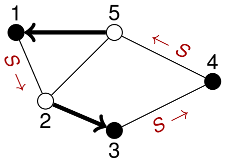
Main difficulty (→ the *score* technique)

Loss of atomicity:

- A node can be involved in several operations at the same time
  ⇒ arbitrary long chain of selections



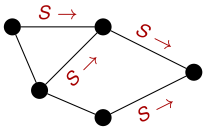
- Initial merging process is faster, but
- tricky configurations :



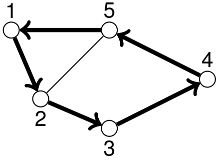
Main difficulty (→ the *score* technique)

Loss of atomicity:

- A node can be involved in several operations at the same time
  ⇒ arbitrary long chain of selections



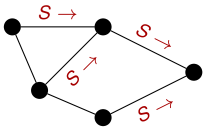
- Initial merging process is faster, but
- tricky configurations :



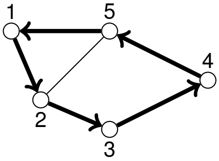
Main difficulty (→ the *score* technique)

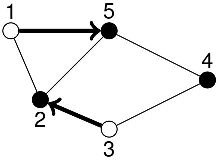
Loss of atomicity:

- A node can be involved in several operations at the same time
  ⇒ arbitrary long chain of selections



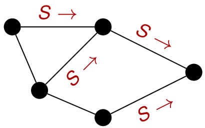
- Initial merging process is faster, but
- tricky configurations :



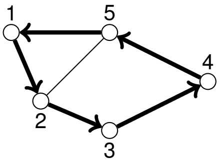


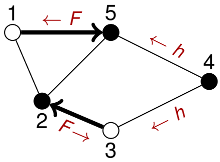
Loss of atomicity:

- A node can be involved in several operations at the same time
  ⇒ arbitrary long chain of selections



- Initial merging process is faster, but
- tricky configurations :

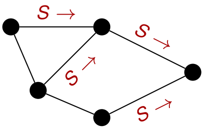




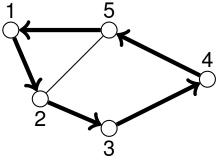
Main difficulty (→ the *score* technique)

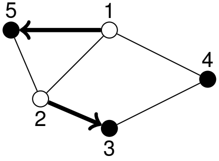
Loss of atomicity:

- A node can be involved in several operations at the same time
  $\Rightarrow$ arbitrary long chain of selections



- Initial merging process is faster, but
- tricky configurations :

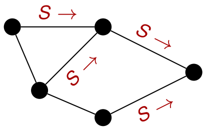




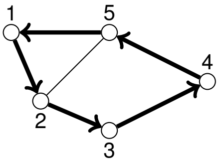
Main difficulty (→ the *score* technique)

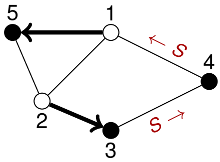
Loss of atomicity:

- A node can be involved in several operations at the same time
  ⇒ arbitrary long chain of selections



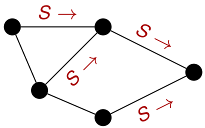
- Initial merging process is faster, but
- tricky configurations :



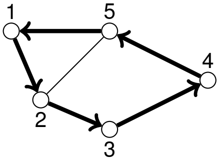


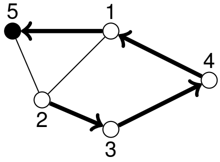
Loss of atomicity:

- A node can be involved in several operations at the same time
  ⇒ arbitrary long chain of selections



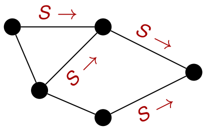
- Initial merging process is faster, but
- tricky configurations :



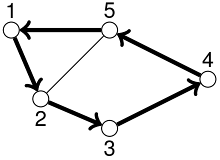


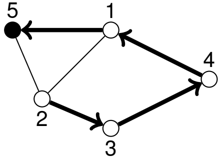
Loss of atomicity:

- A node can be involved in several operations at the same time
  $\Rightarrow$ arbitrary long chain of selections



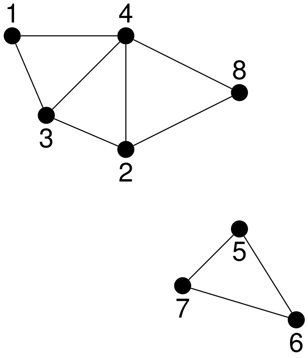
- Initial merging process is faster, but
- tricky configurations :





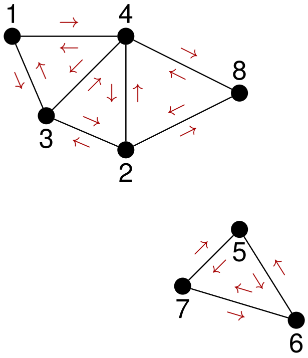
Lemma: scores remain a permutation of IDs !

Execution



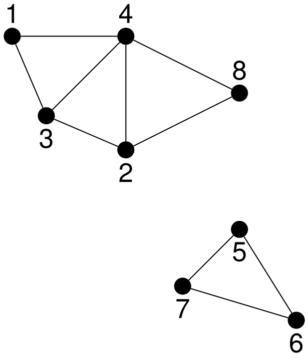
C_0

Execution



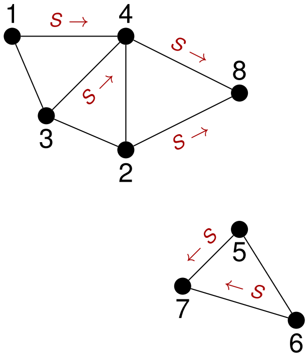
$round_1$

Execution



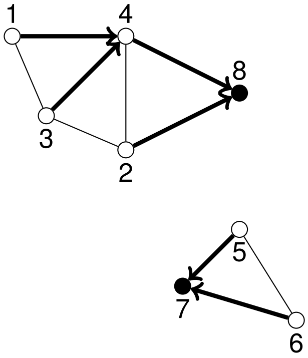
C_1

Execution



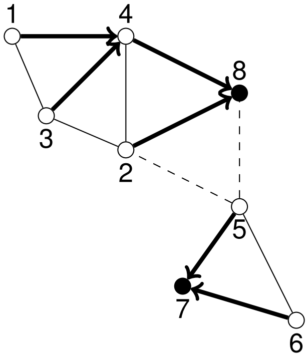
round$_2$

Execution



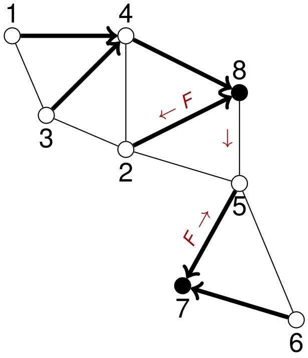
C_2

Execution



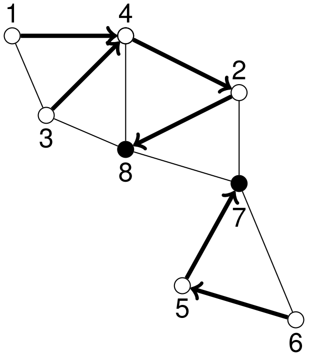
C_2

Execution



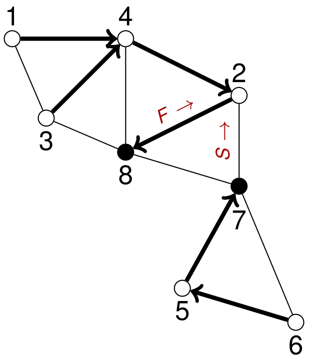
round$_3$

Execution

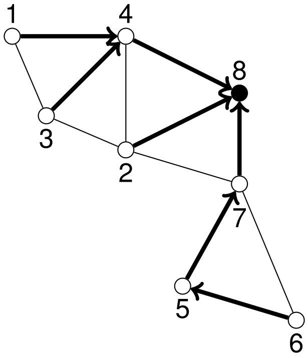


C_3

Execution

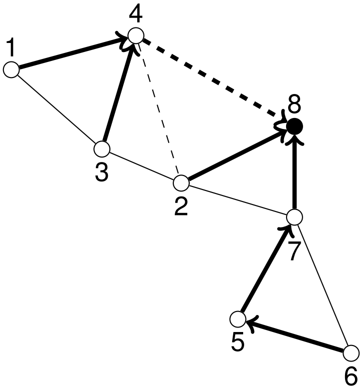


round$_4$



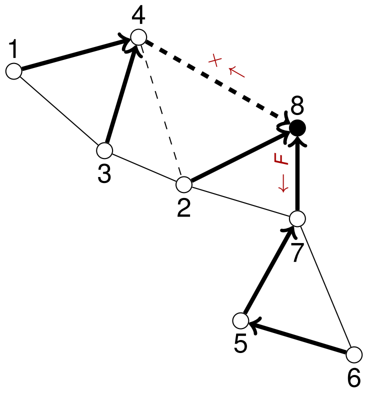
C_4

Execution



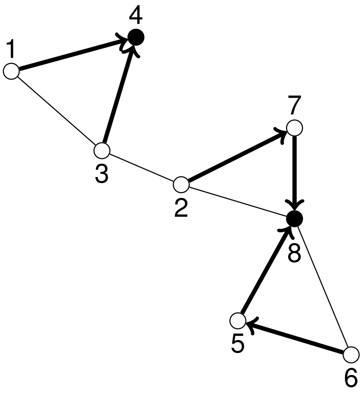
C_4

Execution



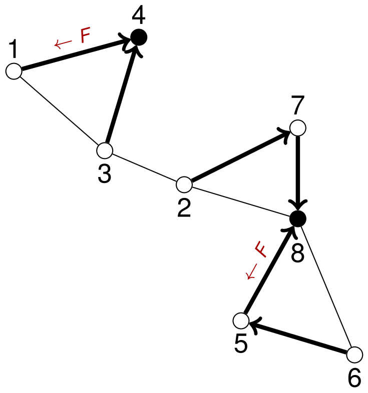
$round_5$

Execution



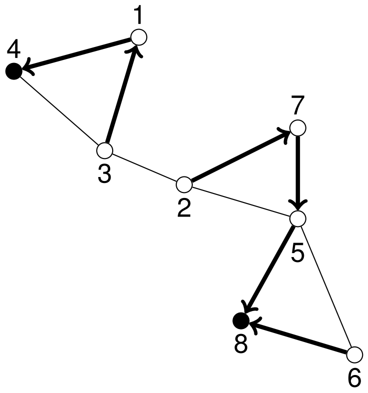
C_5

Execution



round$_6$

Execution



C_6

Intermediate Lemmas

Consistency and state equivalences (at the end of each round)

- $u.parent = \perp \iff u.state = \mathrm{T}$
- $u.parent = v \iff u \in v.children$

Intermediate Lemmas

Consistency and state equivalences (at the end of each round)

- $u.parent = \perp \iff u.state = \mathrm{T}$
- $u.parent = v \iff u \in v.children$

Pseudo trees (helping definitions)

- Pseudo tree : graph in which the outdegree is at most 1
- Pseudo forest : every node belongs to a pseudo tree
- Correct tree : no cycle & exactly one root
- Correct forest : every node belongs to a correct tree

Intermediate Lemmas

Consistency and state equivalences (at the end of each round)

- $u.parent = \perp \iff u.state = \mathrm{T}$
- $u.parent = v \iff u \in v.children$

Pseudo trees (helping definitions)

- Pseudo tree : graph in which the outdegree is at most 1
- Pseudo forest : every node belongs to a pseudo tree
- Correct tree : no cycle & exactly one root
- Correct forest : every node belongs to a correct tree

Lemmas on pseudo trees and pseudo forests

- Correct tree $\iff$ pseudo tree with *at least* one root.
- In all configurations, the parent relation defines a pseudo forest

→ It is sufficient to prove that a root exists in the pseudo tree of every node after each round (node validity)

Reformulation of the proof

Node validity (recursive definition)

A node is valid in C_i if "at least one" token can be found in its pseudo-tree.

That is, if either $u.status = T$ or $u.parent$ is itself a valid node.

Reformulation of the proof

Node validity (recursive definition)

A node is valid in C_i if "at least one" token can be found in its pseudo-tree.

That is, if either $u.status = T$ or $u.parent$ is itself a valid node.

Conclusion of the proof (by induction on the number of rounds)

- In C_0, all nodes are valid.
- If all nodes are valid in C_i, then so are they in C_{i+1} (main Theorem)

$\rightarrow$ In each C_i every node belongs to a correct tree.

And in particular:

- Each node belongs to exactly one tree.
- There is exactly one token per tree.
- There can be no cycles.

See long version for details (CoRR, abs/1410.4373)

We tested the algorithm on a real data-set: Infocom06.

(This trace includes Bluetooth sightings by groups of users carrying small devices – iMotes – for four days at the IEEE Infocom 2006 Conference.)

Summary of the setting:

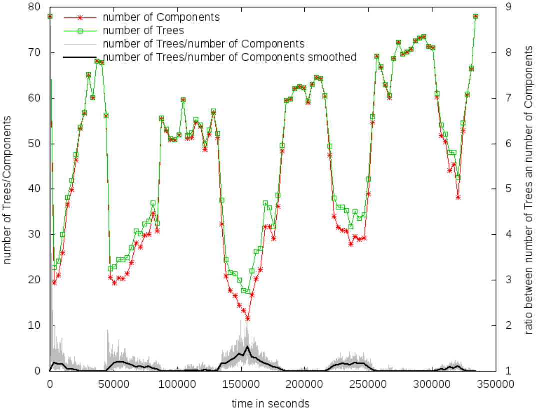
- 78 people equipped with bluetooth devices.
- More than 190 000 contacts between the devices.
- The detection of the new connections is done every seconds.
- The detection of the disconnections is done only every minute.

→ We used the JBotSim library (distributed algorithms in dynamic networks)

Number of trees per component:

- Mean value: 1.08
- Maximal value: 8.58

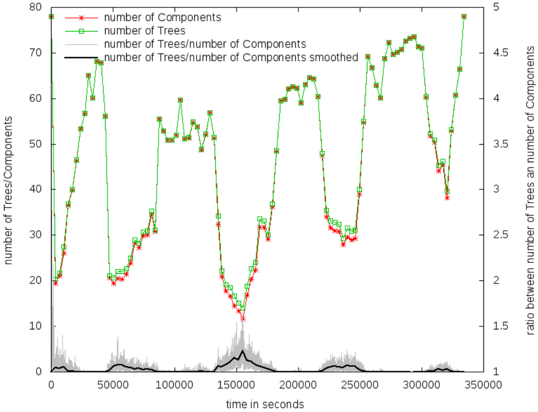
The time spent with one tree per component: 32.68%



Number of trees per component:

- Mean value: 1.03
- Maximal value: 2.77

The time spent with one tree per component: 46.89%



Conclusion and Future works

Conclusion

- Spanning forest principle in unrestricted dynamics
- Correctness is proved and behavior validated experimentally
- From graph relabelings to (synchronous) message-passing

Future works

- Complexity analysis remains open.

→ what model of dynamics to use?
(e.g. edge-markovian evolving graphs)

- Less synchronism?

Thank you !