Characterizing Topological Assumptions of Distributed Algorithms in Dynamic Networks

A.Casteigts, S.Chaumette, A.Ferreira

June 1, 2009

Distributed Algorithms

Level of abstraction

- Local interactions only.
- Abstraction of the communication model.

Distributed Algorithms

Level of abstraction

- Local interactions only.
- Abstraction of the communication model.

Graph Relabellings [Litovsky, Métivier, Sopena 1999]

- State of vertices and edges represented by labels.
- Distributed operations are transition patterns (*relabelling rules*) on these labels *(preconditions, actions)*.

Distributed Algorithms

Level of abstraction

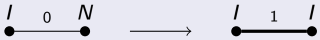
- Local interactions only.
- Abstraction of the communication model.

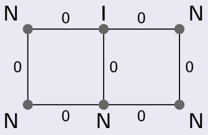
Graph Relabellings [Litovsky, Métivier, Sopena 1999]

- State of vertices and edges represented by labels.
- Distributed operations are transition patterns (*relabelling rules*) on these labels *(preconditions, actions)*.

Example (spanning tree with pre-selected root)

- initial states: I (the root), N (all others), 0 (edges)
- relabelling rule:





Distributed Algorithms

Level of abstraction

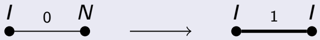
- Local interactions only.
- Abstraction of the communication model.

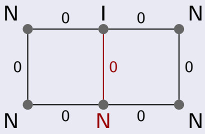
Graph Relabellings [Litovsky, Métivier, Sopena 1999]

- State of vertices and edges represented by labels.
- Distributed operations are transition patterns (*relabelling rules*) on these labels *(preconditions, actions)*.

Example (spanning tree with pre-selected root)

- initial states: I (the root), N (all others), 0 (edges)
- relabelling rule:





Distributed Algorithms

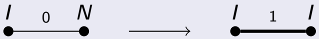
Level of abstraction

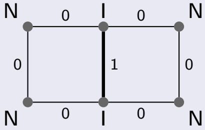
- Local interactions only.
- Abstraction of the communication model.

Graph Relabellings [Litovsky, Métivier, Sopena 1999]

- State of vertices and edges represented by labels.
- Distributed operations are transition patterns (*relabelling rules*) on these labels *(preconditions, actions)*.

Example (spanning tree with pre-selected root)

- initial states: I (the root), N (all others), 0 (edges)
- relabelling rule: 



Distributed Algorithms

Level of abstraction

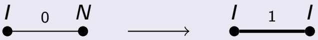
- Local interactions only.
- Abstraction of the communication model.

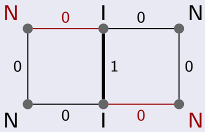
Graph Relabellings [Litovsky, Métivier, Sopena 1999]

- State of vertices and edges represented by labels.
- Distributed operations are transition patterns (*relabelling rules*) on these labels *(preconditions, actions)*.

Example (spanning tree with pre-selected root)

- initial states: I (the root), N (all others), 0 (edges)
- relabelling rule:





Distributed Algorithms

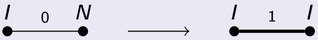
Level of abstraction

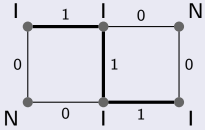
- Local interactions only.
- Abstraction of the communication model.

Graph Relabellings [Litovsky, Métivier, Sopena 1999]

- State of vertices and edges represented by labels.
- Distributed operations are transition patterns (*relabelling rules*) on these labels *(preconditions, actions)*.

Example (spanning tree with pre-selected root)

- initial states: I (the root), N (all others), 0 (edges)
- relabelling rule: 



Distributed Algorithms

Level of abstraction

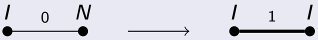
- Local interactions only.
- Abstraction of the communication model.

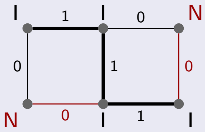
Graph Relabellings [Litovsky, Métivier, Sopena 1999]

- State of vertices and edges represented by labels.
- Distributed operations are transition patterns (*relabelling rules*) on these labels *(preconditions, actions)*.

Example (spanning tree with pre-selected root)

- initial states: I (the root), N (all others), 0 (edges)
- relabelling rule:





Distributed Algorithms

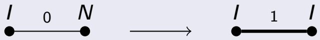
Level of abstraction

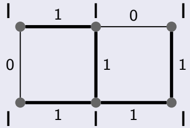
- Local interactions only.
- Abstraction of the communication model.

Graph Relabellings [Litovsky, Métivier, Sopena 1999]

- State of vertices and edges represented by labels.
- Distributed operations are transition patterns (*relabelling rules*) on these labels *(preconditions, actions)*.

Example (spanning tree with pre-selected root)

- initial states: I (the root), N (all others), 0 (edges)
- relabelling rule: 



Characterization

Two approaches

- Characterizing what can be done in a given context

Characterization

Two approaches

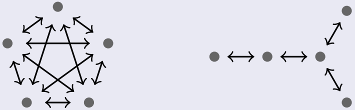
- Characterizing what can be done in a given context

- Characterizing in what context a given thing can be done

Characterization

Two approaches

- Characterizing what can be done in a given context
  - *e.g.* what can be done in a complete (or arborescent) interaction graphs

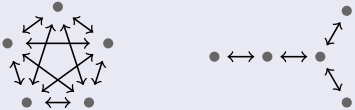


- Characterizing in what context a given thing can be done

Characterization

Two approaches

- Characterizing what can be done in a given context
  - *e.g.* what can be done in a complete (or arborescent) interaction graphs

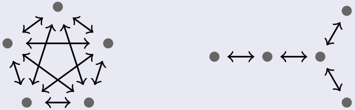


- Characterizing in what context a given thing can be done

Characterization

Two approaches

- Characterizing what can be done in a given context
  - *e.g.* what can be done in a complete (or arborescent) interaction graphs



- Characterizing in what context a given thing can be done
  - what properties a given algorithm requires on the graph dynamics? (*i.e.*, on the *topological* evolution)

Intuitive example

Propagating an information

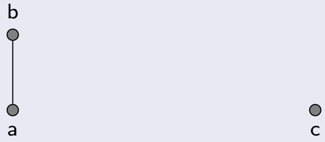
- initial states: I for the initial emitter, N for all the other nodes
- relabelling rule :

Intuitive example

Propagating an information

- initial states: I for the initial emitter, N for all the other nodes
- relabelling rule : 

- example scenario:

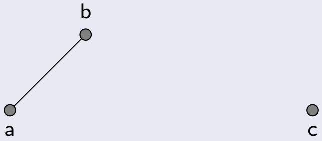


Intuitive example

Propagating an information

- initial states: I for the initial emitter, N for all the other nodes
- relabelling rule : 

- example scenario:

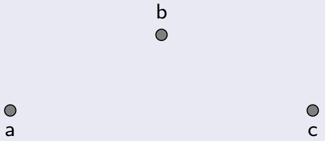


Intuitive example

Propagating an information

- initial states: I for the initial emitter, N for all the other nodes
- relabelling rule :

- example scenario:



Intuitive example

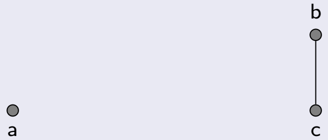
Propagating an information

- initial states: I for the initial emitter, N for all the other nodes
- relabelling rule : $\overset{I}{\bullet}\!\!-\!\!\!-\!\!\!-\!\!\overset{N}{\bullet} \longrightarrow \overset{I}{\bullet}\!\!-\!\!\!-\!\!\!-\!\!\overset{I}{\bullet}$
- example scenario:

Intuitive example

Propagating an information

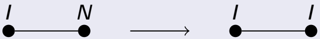
- initial states: I for the initial emitter, N for all the other nodes
- relabelling rule : 
- example scenario:



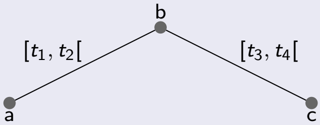
Intuitive example

Propagating an information

- initial states: I for the initial emitter, N for all the other nodes
- relabelling rule :



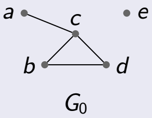
- example scenario:



Formalism to represent dynamic topology

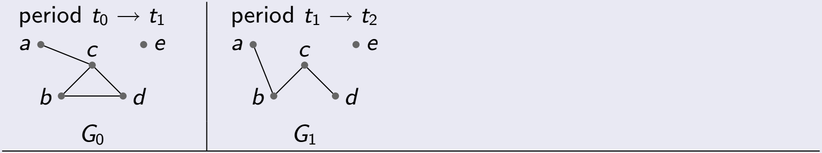
Evolving graphs [Ferreira 2004]

period $t_0 \longrightarrow t_1$



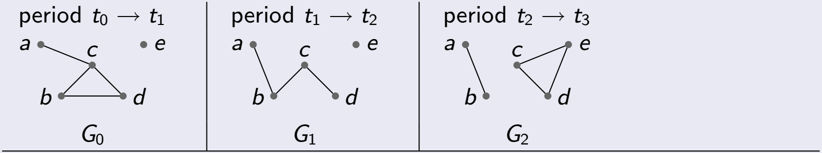
Formalism to represent dynamic topology

Evolving graphs [Ferreira 2004]



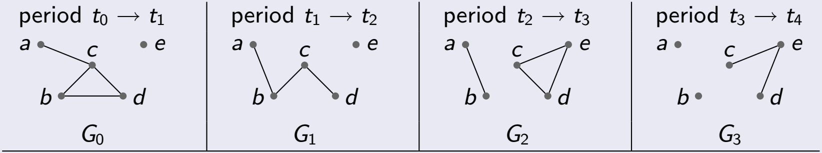
Formalism to represent dynamic topology

Evolving graphs [Ferreira 2004]



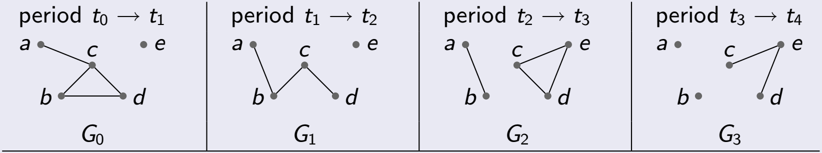
Formalism to represent dynamic topology

Evolving graphs [Ferreira 2004]



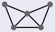
Formalism to represent dynamic topology

Evolving graphs [Ferreira 2004]



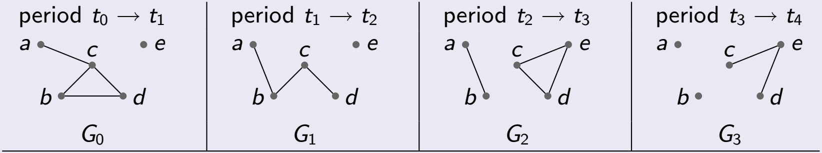
$\mathcal{S}_{\mathbb{T}} = t_0, t_1, t_2, t_3, t_4$

$\mathcal{S}_G = G_0, G_1, G_2, G_3$

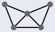
$G = \bigcup_{G_i \in \mathcal{S}_G} =$ 

Formalism to represent dynamic topology

Evolving graphs [Ferreira 2004]



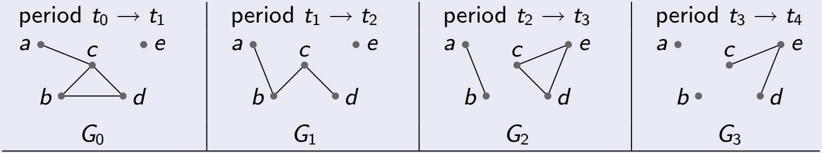
$$\left.\begin{array}{l} \mathcal{S}_{\mathbb{T}} = t_0, t_1, t_2, t_3, t_4 \\ \mathcal{S}_G = G_0, G_1, G_2, G_3 \\ G = \bigcup_{G_i \in \mathcal{S}_G} = \end{array}\right\} \mathcal{G} = (G, \mathcal{S}_G, \mathcal{S}_{\mathbb{T}})$$



is the corresponding *Evolving Graph*.

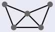
Formalism to represent dynamic topology

Evolving graphs [Ferreira 2004]



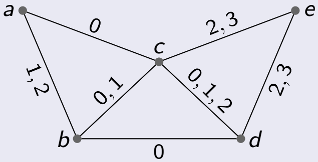
$\mathcal{S}_{\mathbb{T}} = t_0, t_1, t_2, t_3, t_4$

$\mathcal{S}_G = G_0, G_1, G_2, G_3$

$G = \bigcup_{G_i \in \mathcal{S}_G} =$ 

$\Big\}\ \mathcal{G} = (G, \mathcal{S}_G, \mathcal{S}_{\mathbb{T}})$ is the corresponding *Evolving Graph.*

↓ graphical representation ↓

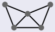


Formalism to represent dynamic topology

Evolving graphs [Ferreira 2004]

$\mathcal{S}_{\mathbb{T}} = t_0, t_1, t_2, t_3, t_4$

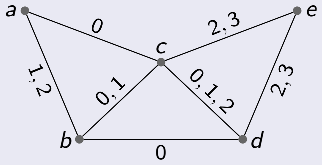
$\mathcal{S}_G = G_0, G_1, G_2, G_3$

$G = \bigcup_{G_i \in \mathcal{S}_G} =$ 

$\Bigg\} \; \mathcal{G} = (G, \mathcal{S}_G, \mathcal{S}_{\mathbb{T}})$

is the corresponding *Evolving Graph.*

↓ graphical representation ↓



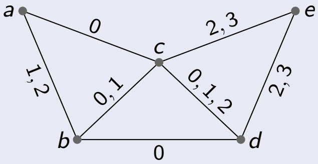
Formalism to represent dynamic topology

Evolving graphs [Ferreira 2004]

$$\left.\begin{array}{l}\mathcal{S}_{\mathbb{T}} = t_0, t_1, t_2, t_3, t_4 \\ \mathcal{S}_G = G_0, G_1, G_2, G_3 \\ G = \bigcup_{G_i \in \mathcal{S}_G} = \end{array}\right\} \mathcal{G} = (G, \mathcal{S}_G, \mathcal{S}_{\mathbb{T}})$$

is the corresponding *Evolving Graph.*

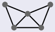
↓ graphical representation ↓



$\longrightarrow$ possibility to express topological properties, and to define related concepts.

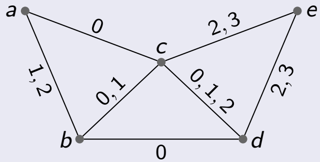
Formalism to represent dynamic topology

Evolving graphs [Ferreira 2004]

$$\mathcal{S}_{\mathbb{T}} = t_0, t_1, t_2, t_3, t_4$$
$$\mathcal{S}_G = G_0, G_1, G_2, G_3$$
$$G = \bigcup_{G_i \in \mathcal{S}_G} =$$ 

$\left.\right\} \mathcal{G} = (G, \mathcal{S}_G, \mathcal{S}_{\mathbb{T}})$ is the corresponding *Evolving Graph.*

↓ graphical representation ↓



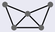
$\longrightarrow$ possibility to express topological properties, and to define related concepts.

e.g. Journey (path over time).

Formalism to represent dynamic topology

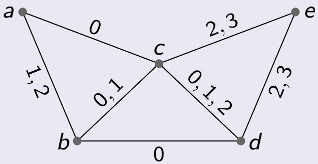
Evolving graphs [Ferreira 2004]

$$\left.\begin{array}{l}\mathcal{S}_{\mathbb{T}} = t_0, t_1, t_2, t_3, t_4 \\ \mathcal{S}_G = G_0, G_1, G_2, G_3 \\ G = \bigcup_{G_i \in \mathcal{S}_G} = \end{array}\right\} \mathcal{G} = (G, \mathcal{S}_G, \mathcal{S}_{\mathbb{T}})$$



is the corresponding *Evolving Graph*.

↓ graphical representation ↓



$\mathcal{J}_{a,e} = \{(a, b, 1), (b, c, 1), (c, d, 1), (d, e, 2)\}$
is a journey from a to e.

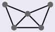
$\longrightarrow$ possibility to express topological properties, and to define related concepts.

e.g. Journey (path over time).

Formalism to represent dynamic topology

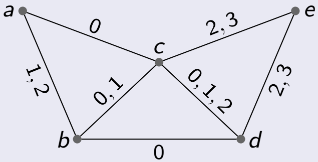
Evolving graphs [Ferreira 2004]

$$\left.\begin{array}{l} \mathcal{S}_{\mathbb{T}} = t_0, t_1, t_2, t_3, t_4 \\ \mathcal{S}_G = G_0, G_1, G_2, G_3 \\ G = \bigcup_{G_i \in \mathcal{S}_G} = \end{array}\right\} \mathcal{G} = (G, \mathcal{S}_G, \mathcal{S}_{\mathbb{T}})$$



is the corresponding *Evolving Graph.*

↓ graphical representation ↓



$\mathcal{J}_{a,e} = \{(a,b,1),(b,c,1),(c,d,1),(d,e,2)\}$ is a journey from a to e.

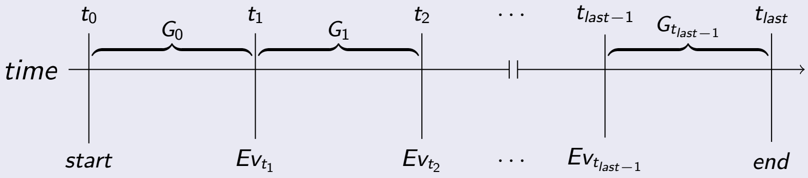
$\mathcal{J}_{a,e} = \{(a,c,0),(c,e,2)\}$ is a strict journey from a to e.

→ possibility to express topological properties, and to define related concepts.

e.g. Journey (path over time).

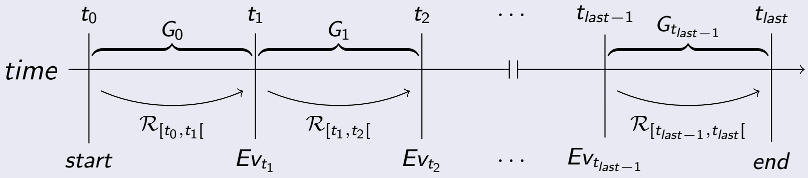
Combination

Relabellings over Evolving Graphs



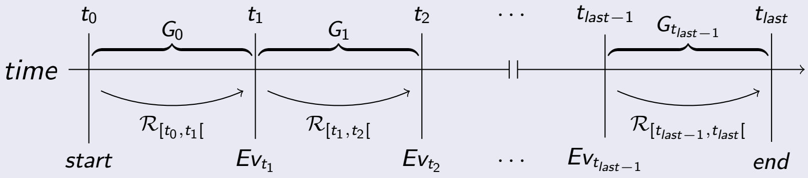
Combination

Relabellings over Evolving Graphs



Combination

Relabellings over Evolving Graphs

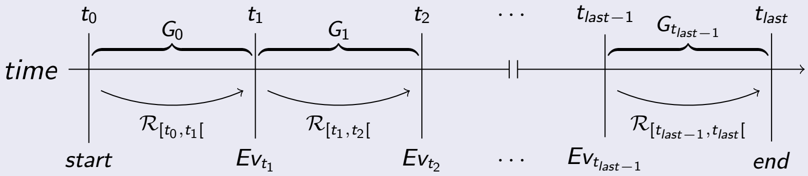


An execution is an alternated sequence of relabellings and topological events:

$$X = \mathcal{R}_{\mathcal{A}_{[t_{last-1},t_{last}[}} \circ Event_{t_{last-1}} \circ .. \circ Event_{t_i} \circ \mathcal{R}_{\mathcal{A}_{[t_{i-1},t_i[}} \circ .. \circ Event_{t_1} \circ \mathcal{R}_{\mathcal{A}_{[t_0,t_1[}}(\mathbf{G}_0)$$

Combination

Relabellings over Evolving Graphs

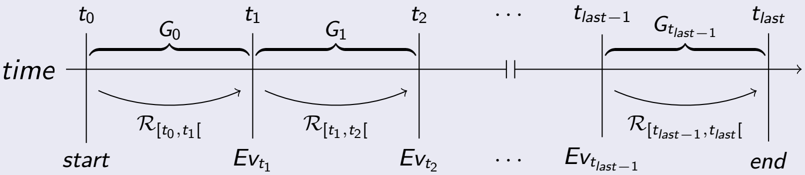


An execution is an alternated sequence of relabellings and topological events:
$X = \mathcal{R}_{\mathcal{A}_{[t_{last-1},t_{last}[}} \circ Event_{t_{last-1}} \circ .. \circ Event_{t_i} \circ \mathcal{R}_{\mathcal{A}_{[t_{i-1},t_i[}} \circ .. \circ Event_{t_1} \circ \mathcal{R}_{\mathcal{A}_{[t_0,t_1[}}(\mathbf{G}_0)$

We note $\mathcal{X}_{\mathcal{A}/\mathcal{G}}$ the set of all possible execution sequences of an algorithm $\mathcal{A}$ over an evolving graph $\mathcal{G}$

Combination

Relabellings over Evolving Graphs



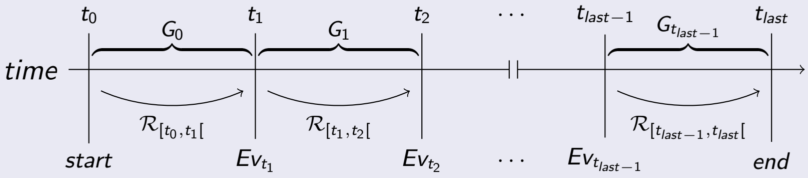
An execution is an alternated sequence of relabellings and topological events:
$X = \mathcal{R}_{\mathcal{A}_{[t_{last-1},t_{last}[}} \circ Event_{t_{last-1}} \circ .. \circ Event_{t_i} \circ \mathcal{R}_{\mathcal{A}_{[t_{i-1},t_i[}} \circ .. \circ Event_{t_1} \circ \mathcal{R}_{\mathcal{A}_{[t_0,t_1[}}(\mathbf{G}_0)$

We note $\mathcal{X}_{\mathcal{A}/\mathcal{G}}$ the set of all possible execution sequences of an algorithm $\mathcal{A}$ over an evolving graph $\mathcal{G}$

Topology-related necessary condition: $\neg\mathcal{C}_{\mathcal{N}}(\mathcal{G}) \implies \nexists X \in \mathcal{X}_{\mathcal{A}/\mathcal{G}} \mid success$.
Topology-related sufficient condition: $\mathcal{C}_{\mathcal{S}}(\mathcal{G}) \implies \forall X$ in $\mathcal{X}_{\mathcal{A}/\mathcal{G}}, success$.

Combination

Relabellings over Evolving Graphs



An execution is an alternated sequence of relabellings and topological events:

$X = \mathcal{R}_{\mathcal{A}_{[t_{last-1},t_{last}[}} \circ Event_{t_{last-1}} \circ .. \circ Event_{t_i} \circ \mathcal{R}_{\mathcal{A}_{[t_{i-1},t_i[}} \circ .. \circ Event_{t_1} \circ \mathcal{R}_{\mathcal{A}_{[t_0,t_1[}}(\mathbf{G}_0)$

We note $\mathcal{X}_{\mathcal{A}/\mathcal{G}}$ the set of all possible execution sequences of an algorithm $\mathcal{A}$ over an evolving graph $\mathcal{G}$

Topology-related necessary condition: $\neg\mathcal{C}_{\mathcal{N}}(\mathcal{G}) \implies \nexists X \in \mathcal{X}_{\mathcal{A}/\mathcal{G}} \mid$ *success*.
Topology-related sufficient condition: $\mathcal{C}_{\mathcal{S}}(\mathcal{G}) \implies \forall X$ in $\mathcal{X}_{\mathcal{A}/\mathcal{G}}$, *success*.

$\rightarrow$ possibility to prove formally that a given property is necessary, or sufficient.

Classes of evolving graphs

Propagation algorithm conditions

In order to inform all the nodes, it is:

- *necessary* that a journey exists from the emitter to every other node ($\mathcal{C}_{\mathcal{N}}$).

Classes of evolving graphs

Propagation algorithm conditions

In order to inform all the nodes, it is:

- *necessary* that a journey exists from the emitter to every other node ($\mathcal{C}_{\mathcal{N}}$).
- *sufficient* that a strict journey exists from the emitter to every other node ($\mathcal{C}_{\mathcal{S}}$).

Classes of evolving graphs

Propagation algorithm conditions

In order to inform all the nodes, it is:

- *necessary* that a journey exists from the emitter to every other node ($\mathcal{C}_{\mathcal{N}}$).
- *sufficient* that a strict journey exists from the emitter to every other node ($\mathcal{C}_{\mathcal{S}}$).

Resulting classes of evolving graphs (*or* dynamic networks)

Classes of evolving graphs

Propagation algorithm conditions

In order to inform all the nodes, it is:

- *necessary* that a journey exists from the emitter to every other node ($\mathcal{C}_{\mathcal{N}}$).
- *sufficient* that a strict journey exists from the emitter to every other node ($\mathcal{C}_{\mathcal{S}}$).

Resulting classes of evolving graphs (*or* dynamic networks)

$\longrightarrow$ $\mathcal{F}_1$: graphs where $\mathcal{C}_{\mathcal{N}}$ is verified for at least one node (1-$\mathcal{J}$-$*$).

Classes of evolving graphs

Propagation algorithm conditions

In order to inform all the nodes, it is:

- *necessary* that a journey exists from the emitter to every other node ($\mathcal{C}_{\mathcal{N}}$).
- *sufficient* that a strict journey exists from the emitter to every other node ($\mathcal{C}_{\mathcal{S}}$).

Resulting classes of evolving graphs (*or* dynamic networks)

$\longrightarrow$ $\mathcal{F}_1$: graphs where $\mathcal{C}_{\mathcal{N}}$ is verified for at least one node (1-$\mathcal{J}$-$*$).

$\longrightarrow$ $\mathcal{F}_3$: same as $\mathcal{F}_1$ but with strict journeys (1-$\mathcal{J}_{strict}$-$*$).

Classes of evolving graphs

Propagation algorithm conditions

In order to inform all the nodes, it is:

- *necessary* that a journey exists from the emitter to every other node ($\mathcal{C}_{\mathcal{N}}$).
- *sufficient* that a strict journey exists from the emitter to every other node ($\mathcal{C}_{\mathcal{S}}$).

Resulting classes of evolving graphs (*or* dynamic networks)

$\longrightarrow$ $\mathcal{F}_1$: graphs where $\mathcal{C}_{\mathcal{N}}$ is verified for at least one node (1-$\mathcal{J}$-$*$).

$\longrightarrow$ $\mathcal{F}_3$: same as $\mathcal{F}_1$ but with strict journeys (1-$\mathcal{J}_{strict}$-$*$).

$\longrightarrow$ $\mathcal{F}_2$: graphs where $\mathcal{C}_{\mathcal{S}}$ is verified for all nodes ($*$-$\mathcal{J}$-$*$).

Classes of evolving graphs

Propagation algorithm conditions

In order to inform all the nodes, it is:

- *necessary* that a journey exists from the emitter to every other node ($\mathcal{C}_\mathcal{N}$).
- *sufficient* that a strict journey exists from the emitter to every other node ($\mathcal{C}_\mathcal{S}$).

Resulting classes of evolving graphs (*or* dynamic networks)

$\longrightarrow$ $\mathcal{F}_1$: graphs where $\mathcal{C}_\mathcal{N}$ is verified for at least one node (1-$\mathcal{J}$-$*$).

$\longrightarrow$ $\mathcal{F}_3$: same as $\mathcal{F}_1$ but with strict journeys (1-$\mathcal{J}_{strict}$-$*$).

$\longrightarrow$ $\mathcal{F}_2$: graphs where $\mathcal{C}_\mathcal{S}$ is verified for all nodes ($*$-$\mathcal{J}$-$*$).

$\longrightarrow$ $\mathcal{F}_4$: same as $\mathcal{F}_2$ but with strict journeys ($*$-$\mathcal{J}_{strict}$-$*$).

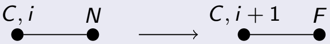
Classes of evolving graphs (2)

Enumeration algorithm with a pre-selected counter

- initial states: $(C, 1)$ for the counter, N for all other vertices.
- relabelling rule: $\overset{C,i}{\bullet}\!\!-\!\!-\!\!-\!\!\overset{N}{\bullet} \longrightarrow \overset{C,i+1}{\bullet}\!\!-\!\!-\!\!-\!\!\overset{F}{\bullet}$
- (N means `non-counted`, F means `counted`)

Classes of evolving graphs (2)

Enumeration algorithm with a pre-selected counter

- initial states: $(C,1)$ for the counter, N for all other vertices.
- relabelling rule: 

- (N means `non-counted`, F means `counted`)

$\mathcal{C}_{\mathcal{N}}$, $\mathcal{C}_{\mathcal{S}}$ and resulting classes

- $\mathcal{C}_{\mathcal{N}} = \mathcal{C}_{\mathcal{S}}$: the counter will share an edge with every other node (at possibly various times and durations).

Classes of evolving graphs (2)

Enumeration algorithm with a pre-selected counter

- initial states: $(C, 1)$ for the counter, N for all other vertices.
- relabelling rule: $\overset{C,i}{\bullet}\!\!-\!\!-\!\!-\!\!\overset{N}{\bullet} \quad\longrightarrow\quad \overset{C,i+1}{\bullet}\!\!-\!\!-\!\!-\!\!\overset{F}{\bullet}$
- (N means `non-counted`, F means `counted`)

$\mathcal{C}_{\mathcal{N}}$, $\mathcal{C}_{\mathcal{S}}$ and resulting classes

- $\mathcal{C}_{\mathcal{N}} = \mathcal{C}_{\mathcal{S}}$: the counter will share an edge with every other node (at possibly various times and durations).
  - $\longrightarrow$ $\mathcal{F}_5$: graphs where the condition holds for at least one vertex (also, failure whatever the counter if outside of this class).

Classes of evolving graphs (2)

Enumeration algorithm with a pre-selected counter

- initial states: $(C, 1)$ for the counter, N for all other vertices.
- relabelling rule: $\overset{C,i}{\bullet}$ —— $\overset{N}{\bullet}$ $\longrightarrow$ $\overset{C,i+1}{\bullet}$ —— $\overset{F}{\bullet}$
- (N means `non-counted`, F means `counted`)

$\mathcal{C}_{\mathcal{N}}$, $\mathcal{C}_{\mathcal{S}}$ and resulting classes

- $\mathcal{C}_{\mathcal{N}} = \mathcal{C}_{\mathcal{S}}$: the counter will share an edge with every other node (at possibly various times and durations).
  - $\longrightarrow$ $\mathcal{F}_5$: graphs where the condition holds for at least one vertex (also, failure whatever the counter if outside of this class).
  - $\longrightarrow$ $\mathcal{F}_6$: graphs where the condition is verified for all nodes (success guaranteed whatever the counter. Also, not being in this class means that at least one node would fail as counter)

Classes of evolving graphs (3)

Decentralized counting algorithm

- initial states: $(C, 1)$ for all vertices.
- relabelling rule: $\overset{C,i}{\bullet}\!\!-\!\!\!-\!\!\!-\!\!\overset{C,j}{\bullet} \longrightarrow \overset{C,i+j}{\bullet}\!\!-\!\!\!-\!\!\!-\!\!\overset{F}{\bullet}$

Classes of evolving graphs (3)

Decentralized counting algorithm

- initial states: $(C, 1)$ for all vertices.
- relabelling rule: $\overset{C,i}{\bullet}\!\!-\!\!-\!\!-\!\!\overset{C,j}{\bullet} \longrightarrow \overset{C,i+j}{\bullet}\!\!-\!\!-\!\!-\!\!\overset{F}{\bullet}$

$\mathcal{C}_\mathcal{N}$ and resulting class

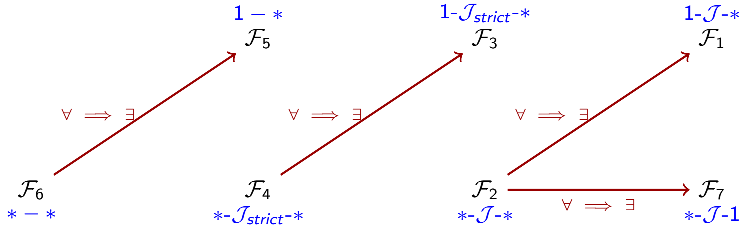
- $\mathcal{C}_\mathcal{N}$: at least one node can be reached by all the others by a journey.

  $\longrightarrow$ $\mathcal{F}_7$: idem.

Classification of dynamic networks

	$1 - *$	$1\text{-}\mathcal{J}_{strict}\text{-}*$	$1\text{-}\mathcal{J}\text{-}*$
	$\mathcal{F}_5$	$\mathcal{F}_3$	$\mathcal{F}_1$
$\mathcal{F}_6$	$\mathcal{F}_4$	$\mathcal{F}_2$	$\mathcal{F}_7$
$* - *$	$*\text{-}\mathcal{J}_{strict}\text{-}*$	$*\text{-}\mathcal{J}\text{-}*$	$*\text{-}\mathcal{J}\text{-}1$

Classification of dynamic networks



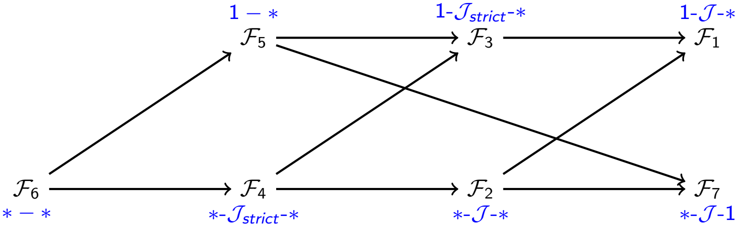
Classification of dynamic networks

$$
\begin{tikzcd}
 & \overset{1-*}{\mathcal{F}_5} \arrow[r, "- \implies \mathcal{J}_{strict}"] & \overset{1\text{-}\mathcal{J}_{strict}\text{-}*}{\mathcal{F}_3} & \overset{1\text{-}\mathcal{J}\text{-}*}{\mathcal{F}_1} \\
\underset{*-*}{\mathcal{F}_6} \arrow[ur] \arrow[r, "- \implies \mathcal{J}_{strict}"'] & \underset{*\text{-}\mathcal{J}_{strict}\text{-}*}{\mathcal{F}_4} \arrow[ur] & \underset{*\text{-}\mathcal{J}\text{-}*}{\mathcal{F}_2} \arrow[ur] \arrow[r] & \underset{*\text{-}\mathcal{J}\text{-}1}{\mathcal{F}_7}
\end{tikzcd}
$$

Classification of dynamic networks

$$
\begin{tikzcd}
 & 1-* & 1\text{-}\mathcal{J}_{strict}\text{-}* & 1\text{-}\mathcal{J}\text{-}* \\
 & \mathcal{F}_5 \arrow[r] & \mathcal{F}_3 \arrow[r, "\mathcal{J}_{strict} \implies \mathcal{J}"] & \mathcal{F}_1 \\
\mathcal{F}_6 \arrow[ur] \arrow[r] & \mathcal{F}_4 \arrow[ur] \arrow[r, "\mathcal{J}_{strict} \implies \mathcal{J}"'] & \mathcal{F}_2 \arrow[ur] \arrow[r] & \mathcal{F}_7 \\
- & *\text{-}\mathcal{J}_{strict}\text{-}* & *\text{-}\mathcal{J}\text{-}* & *\text{-}\mathcal{J}\text{-}1
\end{tikzcd}
$$

Classification of dynamic networks



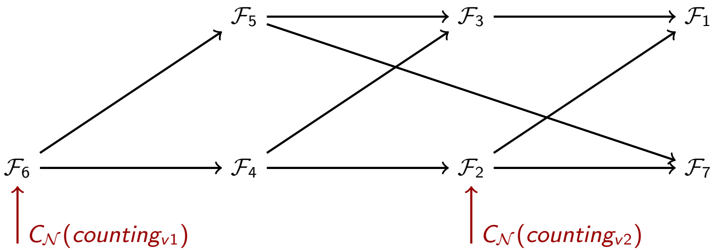
Algorithms Comparison

$$\begin{tikzcd}
 & \mathcal{F}_5 \arrow[r] \arrow[drr] & \mathcal{F}_3 \arrow[r] & \mathcal{F}_1 \\
\mathcal{F}_6 \arrow[ur] \arrow[r] & \mathcal{F}_4 \arrow[ur] \arrow[r] & \mathcal{F}_2 \arrow[ur] \arrow[r] & \mathcal{F}_7
\end{tikzcd}$$

Algorithms Comparison

$$\begin{tikzcd}
& \mathcal{F}_5 \arrow[r] \arrow[rrd] & \mathcal{F}_3 \arrow[r] & \mathcal{F}_1 \\
\mathcal{F}_6 \arrow[ru] \arrow[r] & \mathcal{F}_4 \arrow[r] \arrow[ru] & \mathcal{F}_2 \arrow[r] \arrow[ru] & \mathcal{F}_7 \\
C_{\mathcal{N}}(counting_{v1}) \arrow[u] & & C_{\mathcal{N}}(counting_{v2}) \arrow[u] &
\end{tikzcd}$$

Algorithms Comparison



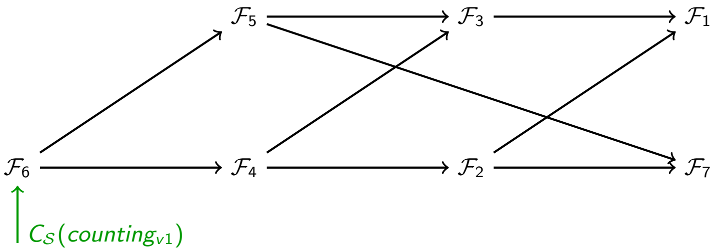
- It exists topologies where $counting_{v1}$ necessarily fails, while $counting_{v2}$ might have some chances of success.

Algorithms Comparison

$$
\begin{tikzcd}
 & \mathcal{F}_5 \arrow[r] \arrow[drr] & \mathcal{F}_3 \arrow[r] & \mathcal{F}_1 \\
\mathcal{F}_6 \arrow[ur] \arrow[r] & \mathcal{F}_4 \arrow[r] \arrow[ur] & \mathcal{F}_2 \arrow[r] \arrow[ur] & \mathcal{F}_7 \\
C_{\mathcal{S}}(counting_{v1}) \arrow[u] & & &
\end{tikzcd}
$$

- It exists topologies where $counting_{v1}$ necessarily fails, while $counting_{v2}$ might have some chances of success.

Algorithms Comparison



- It exists topologies where *counting*$_{v1}$ necessarily fails, while *counting*$_{v2}$ might have some chances of success.
- But if we know that the condition of *counting*$_{v1}$ is matched, then better using this one.

Algorithms Comparison

$$
\begin{tikzcd}
 & \mathcal{F}_5 \arrow[r] \arrow[drr] & \mathcal{F}_3 \arrow[r] & \mathcal{F}_1 \\
\mathcal{F}_6 \arrow[ur] \arrow[r] & \mathcal{F}_4 \arrow[ur] \arrow[r] & \mathcal{F}_2 \arrow[ur] \arrow[r] & \mathcal{F}_7
\end{tikzcd}
$$

- It exists topologies where $counting_{v1}$ necessarily fails, while $counting_{v2}$ might have some chances of success.
- But if we know that the condition of $counting_{v1}$ is matched, then better using this one.
- The choice depends on the expected properties of the target context.

Algorithms Comparison

$$
\begin{tikzcd}
 & \mathcal{F}_5 \arrow[r] \arrow[drr] & \mathcal{F}_3 \arrow[r] & \mathcal{F}_1 \\
\mathcal{F}_6 \arrow[ur] \arrow[r] & \mathcal{F}_4 \arrow[ur] \arrow[r] & \mathcal{F}_2 \arrow[ur] \arrow[r] & \mathcal{F}_7
\end{tikzcd}
$$

- It exists topologies where *counting*$_{v1}$ necessarily fails, while *counting*$_{v2}$ might have some chances of success.
- But if we know that the condition of *counting*$_{v1}$ is matched, then better using this one.
- The choice depends on the expected properties of the target context.
- It would be interesting to now what properties the target context is likely to match.

Automated Verification

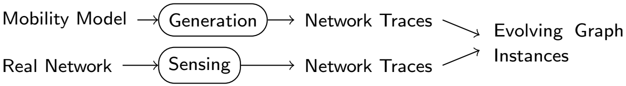
Mobility Model $\longrightarrow$ Generation $\longrightarrow$ Network Traces

Automated Verification

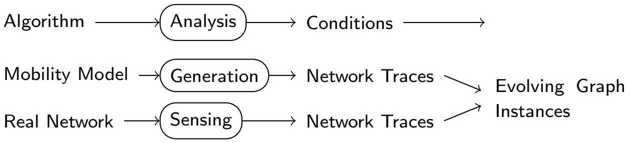
Mobility Model $\longrightarrow$ $\boxed{\text{Generation}}$ $\longrightarrow$ Network Traces

Real Network $\longrightarrow$ $\boxed{\text{Sensing}}$ $\longrightarrow$ Network Traces

Automated Verification



Automated Verification



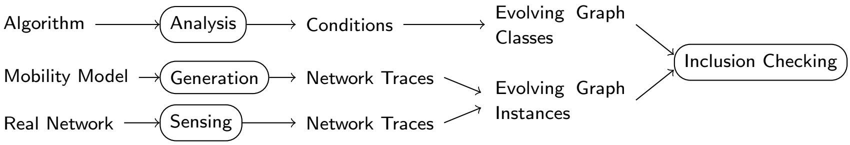
Automated Verification

Algorithm → Analysis → Conditions → Evolving Graph Classes

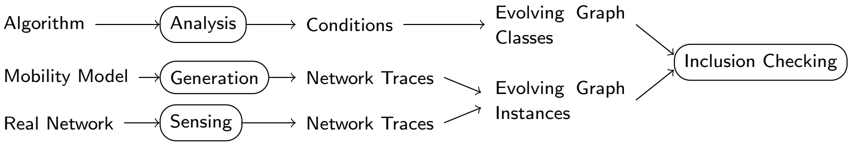
Mobility Model → Generation → Network Traces → Evolving Graph Instances

Real Network → Sensing → Network Traces → Evolving Graph Instances

Automated Verification

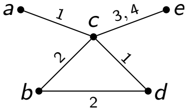


Automated Verification



$\mathcal{C}_{\mathcal{N}}$ (or $\mathcal{C}_{\mathcal{S}}$) is matched in a all/none/some cases? $\Longrightarrow$ decision.

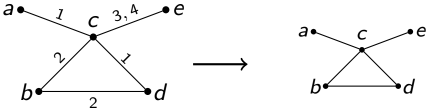
Automated Verification



$\longrightarrow$

$\mathcal{G} = (G, \mathcal{S}_G, \mathcal{S}_{\mathbb{T}})$

Automated Verification

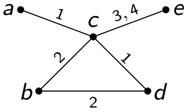


$\mathcal{G} = (G, \mathcal{S}_G, \mathcal{S}_{\mathbb{T}})$

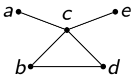
G

(Underlying graph)

Automated Verification

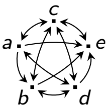


$\mathcal{G} = (G, \mathcal{S}_G, \mathcal{S}_{\mathbb{T}})$



G

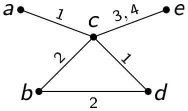
(Underlying graph)



H

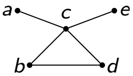
(Transitive closure of journeys)

Automated Verification

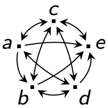


$\mathcal{G} = (G, \mathcal{S}_G, \mathcal{S}_\mathbb{T})$

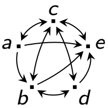
$\longrightarrow$



G
(Underlying graph)

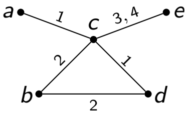


H
(Transitive closure of journeys)

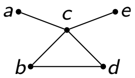


H_{strict}
(Transitive closure of strict journeys)

Automated Verification

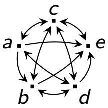


$\mathcal{G} = (G, \mathcal{S}_G, \mathcal{S}_{\mathbb{T}})$



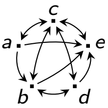
G

(Underlying graph)



H

(Transitive closure of journeys)



H_{strict}

(Transitive closure of strict journeys)

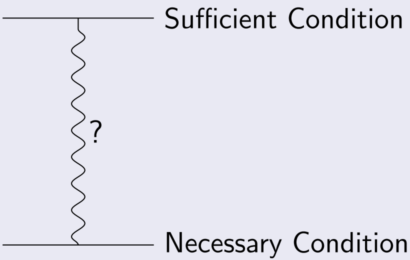
$\mathcal{G} \in \mathcal{F}_1$	$(1\text{-}\mathcal{J}\text{-}*)$	$\Longleftrightarrow$	H contains an out-dominating set of size 1.
$\mathcal{G} \in \mathcal{F}_2$	$(*\text{-}\mathcal{J}\text{-}*)$	$\Longleftrightarrow$	H is a complete graph.
$\mathcal{G} \in \mathcal{F}_3$	$(1\text{-}\mathcal{J}_{strict}\text{-}*)$	$\Longleftrightarrow$	H_{strict} contains an out-dominating set of size 1.
$\mathcal{G} \in \mathcal{F}_4$	$(*\text{-}\mathcal{J}_{strict}\text{-}*)$	$\Longleftrightarrow$	H_{strict} is a complete graph.
$\mathcal{G} \in \mathcal{F}_5$	$(1 - *)$	$\Longleftrightarrow$	G contains a dominating set of size 1.
$\mathcal{G} \in \mathcal{F}_6$	$(* - *)$	$\Longleftrightarrow$	G is a complete graph.
$\mathcal{G} \in \mathcal{F}_7$	$(*\text{-}\mathcal{J}\text{-}1)$	$\Longleftrightarrow$	H contains an in-dominating set of size 1.

Limitations

- Undirected graphs, bandwidth limitations, latency.
  (not restricted by the models)

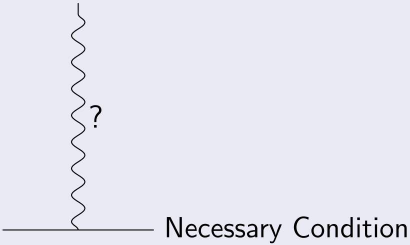
Limitations

- Undirected graphs, bandwidth limitations, latency. (not restricted by the models)
- Topology-related conditions.



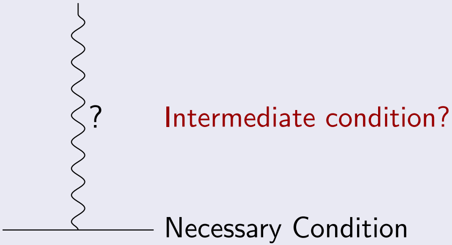
Limitations

- Undirected graphs, bandwidth limitations, latency. (not restricted by the models)
- Topology-related conditions.



Limitations

- Undirected graphs, bandwidth limitations, latency. (not restricted by the models)
- Topology-related conditions.



Limitations

- Undirected graphs, bandwidth limitations, latency. (not restricted by the models)
- Topology-related conditions.
- Scale to more complex algorithms?

Limitations

- Undirected graphs, bandwidth limitations, latency. (not restricted by the models)
- Topology-related conditions.
- Scale to more complex algorithms?

Prospects

Limitations

- Undirected graphs, bandwidth limitations, latency.
  (not restricted by the models)
- Topology-related conditions.
- Scale to more complex algorithms?

Prospects

- new algorithms to be characterized

Limitations

- Undirected graphs, bandwidth limitations, latency. (not restricted by the models)
- Topology-related conditions.
- Scale to more complex algorithms?

Prospects

- new algorithms to be characterized
- new resulting classes of evolving graphs

Limitations

- Undirected graphs, bandwidth limitations, latency.
  (not restricted by the models)
- Topology-related conditions.
- Scale to more complex algorithms?

Prospects

- new algorithms to be characterized
- new resulting classes of evolving graphs
- some insights about the networking impact of mobility

Thank you

Questions?