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 [Litovský, 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: \[ \begin{array}{ccc} I & 0 & N \\ \rightarrow & & \rightarrow \\ I & 1 & I \end{array} \]
**Level of abstraction**

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

**Graph Relabellings** [Litovský, 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: $I \xrightarrow{0,N} I \xrightarrow{1,1}$
**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: 
  
  \[
  \begin{array}{ccc}
  I & 0 & N \\
  \rightarrow & & \rightarrow \\
  I & 1 & I \\
  \end{array}
  \]
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: \[ \begin{array}{c}
  I & 0 & N \\
  \rightarrow \\
  I & 1 & I
\end{array} \]
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: 
  \[
  \begin{array}{c}
  I & 0 & N \\
  \end{array}
  \rightarrow
  \begin{array}{c}
  I & 1 & I \\
  \end{array}
  \]
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: 

\[
\begin{array}{c}
\text{I} & \text{0} & \text{N} \\
\text{1} & \text{1} & \text{I}
\end{array}
\]
Distributed Algorithms

Level of abstraction

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

Graph Relabellings [Litovský, 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: \[ \begin{array}{ccc} I & 0 & N \\ \rightarrow & & \end{array} \]
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
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
Characterizing Topological Assumptions of Dist. Algo. in Dynamic Networks

### 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)*
Propagating an information

- initial states: I for the initial emitter, N for all the other nodes
- relabelling rule: $I \rightarrow N \rightarrow I$
Intuitive example

Propagating an information

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

- relabelling rule:

- example scenario:
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:
Propagating an information

- initial states: \( I \) for the initial emitter, \( N \) for all the other nodes

- relabelling rule:

\[
\begin{array}{c}
I \\
\rightarrow \\
I
\end{array}
\quad
\begin{array}{c}
N \\
\rightarrow \\
N
\end{array}
\]

- example scenario:
Propagating an information

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

- **relabeling rule:**
  \[ \begin{array}{c}
  I \quad N \\
  \quad \rightarrow \\
  \quad I \\
  \end{array} \]

- **example scenario:**

```
\begin{array}{c}
  a \quad b \quad c \\
  \quad [t_1, t_2] \quad [t_3, t_4] \\
\end{array}
```
Formalism to represent dynamic topology

Evolving graphs [Ferreira 2004]

period $t_0 \rightarrow t_1$

\[ G_0 = (G, S_G, S_T) \]

is the corresponding Evolving Graph.

\[ J_{a, e} = \{ (a, b, 1), (b, c, 1), (c, d, 1), (d, e, 2) \} \]

is a journey from $a$ to $e$.

\[ J_{a, e} = \{ (a, c, 0), (c, e, 2) \} \]

is a strict journey from $a$ to $e$.

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

\[ \text{e.g. Journey (path over time).} \]
Formalism to represent dynamic topology

<table>
<thead>
<tr>
<th>Evolving graphs [Ferreira 2004]</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>period</strong> $t_0 \rightarrow t_1$</td>
</tr>
<tr>
<td>$G_0$</td>
</tr>
</tbody>
</table>

$G_0 = (V_0, E_0)$, $G_1 = (V_1, E_1)$

$V_0 = \{a, b, c, d, e\}$

$E_0 = \{(a, c), (b, c), (b, d)\}$

$V_1 = \{a, b, c, d\}$

$E_1 = \{(a, c), (b, c), (b, d)\}$

$S_T = \{t_0, t_1, t_2, t_3, t_4\}$

$S_G = \{G_0, G_1\}$

$G = (S_G, S_T)$ is the corresponding Evolving Graph.

Diagram:

Graphical representation:

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

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

Possibility to express topological properties, and to define related concepts.
Formalism to represent dynamic topology

<table>
<thead>
<tr>
<th>Period</th>
<th>Graph</th>
<th>Period</th>
<th>Graph</th>
<th>Period</th>
<th>Graph</th>
</tr>
</thead>
<tbody>
<tr>
<td>$t_0 \rightarrow t_1$</td>
<td>$G_0$</td>
<td>$t_1 \rightarrow t_2$</td>
<td>$G_1$</td>
<td>$t_2 \rightarrow t_3$</td>
<td>$G_2$</td>
</tr>
</tbody>
</table>

Evolving graphs [Ferreira 2004]

period $t_0 \rightarrow t_1$

$G_0$

period $t_1 \rightarrow t_2$

$G_1$

period $t_2 \rightarrow t_3$

$G_2$
Formalism to represent dynamic topology

### Evolving graphs [Ferreira 2004]

<table>
<thead>
<tr>
<th>Period</th>
<th>Graph</th>
<th>Period</th>
<th>Graph</th>
<th>Period</th>
<th>Graph</th>
<th>Period</th>
<th>Graph</th>
</tr>
</thead>
<tbody>
<tr>
<td>$t_0 \rightarrow t_1$</td>
<td><img src="image" alt="G0" /></td>
<td>$t_1 \rightarrow t_2$</td>
<td><img src="image" alt="G1" /></td>
<td>$t_2 \rightarrow t_3$</td>
<td><img src="image" alt="G2" /></td>
<td>$t_3 \rightarrow t_4$</td>
<td><img src="image" alt="G3" /></td>
</tr>
<tr>
<td>$a \rightarrow c \rightarrow e \rightarrow d \rightarrow b$</td>
<td>$a \rightarrow c \rightarrow d \rightarrow b \rightarrow e$</td>
<td>$a \rightarrow c \rightarrow e \rightarrow d \rightarrow b$</td>
<td>$a \rightarrow c \rightarrow d \rightarrow b \rightarrow e$</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

$S_T = t_0, t_1, t_2, t_3, t_4$

$S_G = G_0, G_1, G_2, G_3$

$G_i \in S_G$

$G = (G, S_G, S_T)$

is the corresponding Evolving Graph.

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

$J_{a,e} = \{(a,c,0), (c,e,2)\}$ is a strict journey from $a$ to $e$. 
Formalism to represent dynamic topology

### Evolving graphs [Ferreira 2004]

<table>
<thead>
<tr>
<th>Period</th>
<th>Graph</th>
</tr>
</thead>
<tbody>
<tr>
<td>$t_0 \rightarrow t_1$</td>
<td>$G_0 = {a \rightarrow c, c \rightarrow b, b \rightarrow d}$</td>
</tr>
<tr>
<td>$t_1 \rightarrow t_2$</td>
<td>$G_1 = {a \rightarrow c, c \rightarrow e}$</td>
</tr>
<tr>
<td>$t_2 \rightarrow t_3$</td>
<td>$G_2 = {a \rightarrow c, c \rightarrow d}$</td>
</tr>
<tr>
<td>$t_3 \rightarrow t_4$</td>
<td>$G_3 = {c \rightarrow e}$</td>
</tr>
</tbody>
</table>

Let $S_T = \{t_0, t_1, t_2, t_3, t_4\}$

Let $S_G = \{G_0, G_1, G_2, G_3\}$

Let $G = \bigcup_{G_i \in S_G}$

$G$ is the corresponding Evolving Graph.
**Formalism to represent dynamic topology**

**Evolving graphs** [Ferreira 2004]

<table>
<thead>
<tr>
<th>Period $t_0 \rightarrow t_1$</th>
<th>Period $t_1 \rightarrow t_2$</th>
<th>Period $t_2 \rightarrow t_3$</th>
<th>Period $t_3 \rightarrow t_4$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$G_0$</td>
<td>$G_1$</td>
<td>$G_2$</td>
<td>$G_3$</td>
</tr>
<tr>
<td>$a \bullet$</td>
<td>$a \bullet$</td>
<td>$a \bullet$</td>
<td>$a \bullet$</td>
</tr>
<tr>
<td>$b \bullet$</td>
<td>$b \bullet$</td>
<td>$b \bullet$</td>
<td>$b \bullet$</td>
</tr>
<tr>
<td>$c \bullet$</td>
<td>$c \bullet$</td>
<td>$c \bullet$</td>
<td>$c \bullet$</td>
</tr>
<tr>
<td>$d \bullet$</td>
<td>$d \bullet$</td>
<td>$d \bullet$</td>
<td>$d \bullet$</td>
</tr>
<tr>
<td>$e \bullet$</td>
<td>$e \bullet$</td>
<td>$e \bullet$</td>
<td>$e \bullet$</td>
</tr>
</tbody>
</table>

$S_T = t_0, t_1, t_2, t_3, t_4$

$S_G = G_0, G_1, G_2, G_3$

$G = \bigcup_{G_i \in S_G} = \text{graphical representation}$

$G = (G, S_G, S_T)$ is the corresponding *Evolving Graph*.
Formalism to represent dynamic topology

Evolving graphs [Ferreira 2004]

Period $t_0 \rightarrow t_1$

$$\begin{array}{c}
\begin{figure}\[a c & c d & e \]
\end{figure}\\
\end{array}$$

Period $t_1 \rightarrow t_2$

$$\begin{array}{c}
\begin{figure}\[a b & c & d \]
\end{figure}\\
\end{array}$$

Period $t_2 \rightarrow t_3$

$$\begin{array}{c}
\begin{figure}\[a c & d e \]
\end{figure}\\
\end{array}$$

Period $t_3 \rightarrow t_4$

$$\begin{array}{c}
\begin{figure}\[a b & c & d \]
\end{figure}\\
\end{array}$$

$S_T = t_0, t_1, t_2, t_3, t_4$

$S_G = G_0, G_1, G_2, G_3$

$G = \bigcup\left\{ G_i \in S_G \right\}$

$G = (G, S_G, S_T)$

is the corresponding Evolving Graph.

↓ graphical representation ↓
Formalism to represent dynamic topology

Evolving graphs [Ferreira 2004]

\[ S_T = t_0, t_1, t_2, t_3, t_4 \]
\[ S_G = G_0, G_1, G_2, G_3 \]
\[ G = \bigcup_{G_i \in S_G} = \]
\[ G = (G, S_G, S_T) \]

is the corresponding Evolving Graph.

↓ graphical representation ↓
Formalism to represent dynamic topology

Evolving graphs [Ferreira 2004]

\[ S_T = t_0, t_1, t_2, t_3, t_4 \]
\[ S_G = G_0, G_1, G_2, G_3 \]
\[ G = \bigcup_{G_i \in S_G} = \]

\[ G = (G, S_G, S_T) \]

is the corresponding Evolving Graph.

\[ J_{a, e} = \{ (a, c, 0), (c, e, 2) \} \]
is a strict journey from \(a\) to \(e\).

\[ \rightarrow \] possibility to express topological properties, and to define related concepts.
Formalism to represent dynamic topology

Evolving graphs [Ferreira 2004]

\[ S_T = t_0, t_1, t_2, t_3, t_4 \]
\[ S_G = G_0, G_1, G_2, G_3 \]
\[ G = \bigcup_{G_i \in S_G} G_i \]

\[ G = (G, S_G, S_T) \]
is the corresponding Evolving Graph.

↓ graphical representation ↓

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

e.g. Journey (path over time).
Evolving graphs [Ferreira 2004]

$S_T = t_0, t_1, t_2, t_3, t_4$
$S_G = G_0, G_1, G_2, G_3$
$G = \bigcup_{G_i \in S_G} = \text{graphical representation}$

\[ \{ (a, b, 1), (b, c, 1), (c, d, 1), (d, e, 2) \} \]

is a journey from $a$ to $e$.

\[ \rightarrow \text{possibility to express topological properties, and to define related concepts.} \]

\[ \text{e.g. Journey (path over time).} \]
Formalism to represent dynamic topology

Evolving graphs [Ferreira 2004]

\[ S_T = t_0, t_1, t_2, t_3, t_4 \]
\[ S_G = G_0, G_1, G_2, G_3 \]
\[ G = \bigcup_{G_i \in S_G} = \]

\[ \{ G, S_G, S_T \} \]
is the corresponding *Evolving Graph*.

\[ \downarrow \text{graphical representation} \downarrow \]

\[ J_{a, e} = \{(a, b, 1), (b, c, 1), (c, d, 1), (d, e, 2)\} \]
is a journey from a to e.

\[ J_{a, e} = \{(a, c, 0), (c, e, 2)\} \]
is a strict journey from a to e.

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

*e.g.* Journey (path over time).
Relabellings over Evolving Graphs

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

\[ \mathcal{X} = R_A[t_{\text{last}}-1, t_{\text{last}}] \circ \cdots \circ \text{Event} t_{\text{last}}-1 \circ \cdots \circ \text{Event} t_1 \circ \cdots \circ R_A[t_0, t_1] (G_0) \]

We note \( \mathcal{X}_A/G \) the set of all possible execution sequences of an algorithm \( A \) over an evolving graph \( G \).

Topology-related necessary condition:

\[ \neg C_N(G) \implies \nexists \mathcal{X} \in \mathcal{X}_A/G | \text{success} \].

Topology-related sufficient condition:

\[ C_S(G) \implies \forall \mathcal{X} \in \mathcal{X}_A/G, \text{success} \].
Relabellings over Evolving Graphs

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

\[ X = R_A[t_{last-1}, t_{last}] \circ \text{Event} \circ \ldots \circ \text{Event} \circ R_A[t_0, t_1] \]

We note \( X_A/G \) the set of all possible execution sequences of an algorithm \( A \) over an evolving graph \( G \).

Topology-related necessary condition:

\[ \neg C_N(G) = \Rightarrow \not\exists X \in X_A/G | \text{success} \]

Topology-related sufficient condition:

\[ C_S(G) = \Rightarrow \forall X \text{ in } X_A/G, \text{success} \]

→ possibility to prove formally that a given property is necessary, or sufficient.
An execution is an alternated sequence of relabellings and topological events:

\[ X = \mathcal{R}_A[t_{\text{last} - 1}, t_{\text{last}}] \circ \text{Event}_{t_{\text{last} - 1}} \circ \ldots \circ \text{Event}_{t_i} \circ \mathcal{R}_A[t_{i - 1}, t_i] \circ \ldots \circ \text{Event}_{t_1} \circ \mathcal{R}_A[t_0, t_1](G_0) \]
An execution is an alternated sequence of relabellings and topological events:

\[ X = \mathcal{R}_A[t_{last-1}, t_{last}] \circ \text{Event}_{t_{last-1}} \circ \ldots \circ \text{Event}_{t_i} \circ \mathcal{R}_A[t_{i-1}, t_i] \circ \ldots \circ \text{Event}_{t_1} \circ \mathcal{R}_A[t_0, t_1](G_0) \]

We note \( \mathcal{X}_{A/G} \) the set of all possible execution sequences of an algorithm \( A \) over an evolving graph \( G \).
Relabellings over Evolving Graphs

An execution is an alternated sequence of relabellings and topological events:
\[ X = R_{\mathcal{A}[t_{last-1}, t_{last}]} \cdot Event_{t_{last-1}} \circ \ldots \circ Event_{t_i} \circ R_{\mathcal{A}[t_{i-1}, t_i]} \circ \ldots \circ Event_{t_1} \circ R_{\mathcal{A}[t_0, t_1]}(G_0) \]

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

**Topology-related necessary condition:** \( \neg C_N(G) \implies \nexists X \in \mathcal{X}_{\mathcal{A}/G} | \text{success} \).

**Topology-related sufficient condition:** \( C_S(G) \implies \forall X \text{ in } \mathcal{X}_{\mathcal{A}/G}, \text{success} \).
Relabellings over Evolving Graphs

An execution is an alternated sequence of relabellings and topological events:
\[ X = \mathcal{R}_{A[t_{last-1}, t_{last}]} \circ \text{Event}_{t_{last-1}} \circ \ldots \circ \text{Event}_{t_i} \circ \mathcal{R}_{A[t_{i-1}, t_i]} \circ \ldots \circ \text{Event}_{t_1} \circ \mathcal{R}_{A[t_0, t_1]}(G_0) \]

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

**Topology-related necessary condition:** \( \neg C_N(G) \implies \nexists X \in \mathcal{X}_{A/G} \mid \text{success} \).  

**Topology-related sufficient condition:** \( C_S(G) \implies \forall X \text{ in } \mathcal{X}_{A/G}, \text{success} \).

→ 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 \((C_N)\).

\[ \Rightarrow F_1: \text{graphs where } C_N \text{ is verified for at least one node (1-J-*)} \]

\[ \Rightarrow F_3: \text{same as } F_1 \text{ but with strict journeys (1-J-strict-*)} \]

\[ \Rightarrow F_2: \text{graphs where } C_S \text{ is verified for all nodes (*-J-*)} \]

\[ \Rightarrow F_4: \text{same as } F_2 \text{ but with strict journeys (*-J-strict-*)} \]
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 ($C_N$).
- **sufficient** that a strict journey exists from the emitter to every other node ($C_S$).
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 \( (C_N) \).
- **sufficient** that a strict journey exists from the emitter to every other node \( (C_S) \).

Resulting classes of evolving graphs (or dynamic networks)

\( \rightarrow F_1 \): graphs where \( C_N \) is verified for at least one node (1-J-∗).

\( \rightarrow F_3 \): same as \( F_1 \) but with strict journeys (1-J-strict-∗).

\( \rightarrow F_2 \): graphs where \( C_S \) is verified for all nodes (∗-J-∗).

\( \rightarrow F_4 \): same as \( F_2 \) but with strict journeys (∗-J-strict-∗).
Classes of evolving graphs

Propagation algorithm conditions

In order to inform all the nodes, it is:

- \textit{necessary} that a journey exists from the emitter to every other node (\(C_N\)).
- \textit{sufficient} that a strict journey exists from the emitter to every other node (\(C_S\)).

Resulting classes of evolving graphs (or dynamic networks)

\[ \rightarrow \mathcal{F}_1 : \text{graphs where } C_N \text{ is verified for at least one node (1-J-*)}. \]
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 \((C_N)\).
- **sufficient** that a strict journey exists from the emitter to every other node \((C_S)\).

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

- \(\mathcal{F}_1\): graphs where \(C_N\) is verified for at least one node (1-\(J\)-*).
- \(\mathcal{F}_3\): same as \(\mathcal{F}_1\) but with strict journeys (1-\(J_{\text{strict}}\)-*).
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 ($C_N$).
- **sufficient** that a strict journey exists from the emitter to every other node ($C_S$).

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

- $\mathcal{F}_1$: graphs where $C_N$ is verified for at least one node ($1-J-*$).
- $\mathcal{F}_3$: same as $\mathcal{F}_1$ but with strict journeys ($1-J_{\text{strict}}-*$).
- $\mathcal{F}_2$: graphs where $C_S$ is verified for all nodes ($*J-J-*$).
### 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 ($C_N$).
- **sufficient** that a strict journey exists from the emitter to every other node ($C_S$).

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

- $\rightarrow \mathcal{F}_1$: graphs where $C_N$ is verified for at least one node (1-$\mathcal{I}$-*$).  
- $\rightarrow \mathcal{F}_3$: same as $\mathcal{F}_1$ but with strict journeys (1-$\mathcal{I}_{\text{strict}}$-*$).  
- $\rightarrow \mathcal{F}_2$: graphs where $C_S$ is verified for all nodes (*-$\mathcal{I}$-*$).  
- $\rightarrow \mathcal{F}_4$: same as $\mathcal{F}_2$ but with strict journeys (*-$\mathcal{I}_{\text{strict}}$-*$).
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: \(C_i \rightarrow C_{i+1} F\)
- \((N\) 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:
  \[
  \begin{array}{ccc}
  C, i & N & C, i + 1 \\
  \end{array}
  \]
  
  (\(N\) means non-counted, \(F\) means counted)

\(C_N, C_S\) and resulting classes

- \(C_N = C_S\): 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:**
  \[
  C, i \quad N \quad C, i + 1 \quad F
  \]
- \((N\) means non-counted, \(F\) means counted) 

### \(C_N, C_S\) and resulting classes

- \(C_N = C_S\): the counter will share an edge with every other node (at possibly various times and durations).
  - \(\mathcal{F}_5\): 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: \(\begin{array}{c}
C, i \\
\hline
N \\
\hline
C, i + 1 \end{array}\) \(\rightarrow\) \(\begin{array}{c}
C, i + 1 \\
\hline
F \\
\hline
\end{array}\)
- \((N\) means non-counted, \(F\) means counted\)

\(C_N, C_S\) and resulting classes

- \(C_N = C_S\): the counter will share an edge with every other node (at possibly various times and durations).
  - \(F_5\): graphs where the condition holds for at least one vertex (also, failure whatever the counter if outside of this class).
  - \(F_6\): 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).
Decentralized counting algorithm

- initial states: \((C, 1)\) for all vertices.

- relabelling rule:

\[
\begin{array}{cccc}
C, i & C, j & \rightarrow & C, i + j & F
\end{array}
\]
### Decentralized counting algorithm

- initial states: $(C, 1)$ for all vertices.
- relabelling rule: $C_i \rightarrow C_{i+j}$

### $C_N$ and resulting class

- $C_N$: at least one node can be reached by all the others by a journey.
- $\rightarrow \mathcal{F}_I$: idem.
## Classification of dynamic networks

<table>
<thead>
<tr>
<th></th>
<th>Purpose</th>
<th>Applications</th>
<th>Conclusion</th>
</tr>
</thead>
<tbody>
<tr>
<td>(F_5)</td>
<td>1 - * (F_3)</td>
<td>1 - (J_{\text{strict}}) - * (F_1)</td>
<td></td>
</tr>
<tr>
<td>(F_4)</td>
<td>* - * (F_2)</td>
<td>* - (J_{\text{strict}}) - * (F_7)</td>
<td></td>
</tr>
</tbody>
</table>
Classification of dynamic networks

\[ \forall \Rightarrow \exists \]

\[ \forall \Rightarrow \exists \]

\[ \forall \Rightarrow \exists \]

\[ \forall \Rightarrow \exists \]

\[ \forall \Rightarrow \exists \]

\[ \forall \Rightarrow \exists \]

\[ \forall \Rightarrow \exists \]

A.Casteigts, S.Chaumette, A.Ferreira
Characterizing Topological Assumptions of Dist. Algo. in Dynamic Networks
Classification of dynamic networks

\[
\begin{align*}
F_5 & \xrightarrow{1 - \ast} F_3 \\
F_6 & \xrightarrow{\ast - \ast} F_4 \\
F_2 & \xrightarrow{\ast \neg \mathcal{J} - \ast} F_7
\end{align*}
\]
Classification of dynamic networks

\[
\begin{align*}
\mathcal{F}_6 & \xrightarrow{*-*} \mathcal{F}_4 & \xrightarrow{*-\mathcal{I}_{\text{strict}}-*} \mathcal{F}_2 & \xrightarrow{*-\mathcal{I}-*} \mathcal{F}_1 \\
\mathcal{F}_5 & \xrightarrow{1-*} \mathcal{F}_3 & \xrightarrow{1-\mathcal{I}_{\text{strict}}-*} \mathcal{I}_{\text{strict}} \Rightarrow \mathcal{I} & \xrightarrow{1-\mathcal{I}-*} \mathcal{F}_7
\end{align*}
\]
Classification of dynamic networks

- $F_6 \rightarrow F_4 \rightarrow F_2 \rightarrow F_7$
- $1 - J$
- $1-J_{strict}$
- $1-J$
- $1-J_{strict}$
- $1-J$
- $1-J_{strict}$
- $J_{strict}$

A. Casteigts, S. Chaumette, A. Ferreira
Characterizing Topological Assumptions of Dist. Algo. in Dynamic Networks
It exists topologies where \( \text{counting} v_1 \) necessarily fails, while \( \text{counting} v_2 \) might have some chances of success. But if we know that the condition of \( \text{counting} v_1 \) is matched, then better using this one. The choice depends on the expected properties of the target context. It would be interesting to know what properties the target context is likely to match.
Algorithms Comparison

$F_6\rightarrow F_4\rightarrow F_2\rightarrow F_7$

$F_5\rightarrow F_3\rightarrow F_1$

$C_N(\text{counting}_{v_1})$

$C_N(\text{counting}_{v_2})$
It exists topologies where \( counting_{v_1} \) necessarily fails, while \( counting_{v_2} \) might have some chances of success.
It exists topologies where $counting_{v_1}$ necessarily fails, while $counting_{v_2}$ might have some chances of success.
It exists topologies where \(\text{counting}_{v_1}\) necessarily fails, while \(\text{counting}_{v_2}\) might have some chances of success.

But if we know that the condition of \(\text{counting}_{v_1}\) is matched, then better using this one.
It exists topologies where \(\text{counting}_{v_1}\) necessarily fails, while \(\text{counting}_{v_2}\) might have some chances of success.

But if we know that the condition of \(\text{counting}_{v_1}\) is matched, then better using this one.

The choice depends on the expected properties of the target context.
It exists topologies where $\text{counting}_{v_1}$ necessarily fails, while $\text{counting}_{v_2}$ might have some chances of success.

But if we know that the condition of $\text{counting}_{v_1}$ 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 → Generation → Network Traces
Automated Verification

Mobility Model $\rightarrow$ Generation $\rightarrow$ Network Traces
Real Network $\rightarrow$ Sensing $\rightarrow$ Network Traces
Automated Verification

- Mobility Model \( \rightarrow \) Generation \( \rightarrow \) Network Traces \( \rightarrow \) Evolving Graph
- Real Network \( \rightarrow \) Sensing \( \rightarrow \) Network Traces

A.Casteigts, S.Chaumette, A.Ferreira
Characterizing Topological Assumptions of Dist. Algo. in Dynamic Networks 12 / 15
Automated Verification

Algorithm → Analysis → Conditions →

Mobility Model → Generation → Network Traces → Evolving Graph Instances

Real Network → Sensing → Network Traces →
Automated Verification

- Algorithm → Analysis → Conditions
- Mobility Model → Generation → Network Traces
- Real Network → Sensing → Network Traces

Evolving Graph Classes
Evolving Graph Instances

$C_N$ (or $C_S$) is matched in all/none/some cases? $\Rightarrow$ decision.
Automated Verification

Algorithm → Analysis → Conditions → Evolving Graph Classes
Mobility Model → Generation → Network Traces → Evolving Graph Instances
Real Network → Sensing → Network Traces → Inclusion Checking

C\(N\) (or C\(S\)) is matched in a all/none/some cases?
\[ \Rightarrow \text{decision.} \]
Automated Verification

$C_N$ (or $C_S$) is matched in a all/none/some cases? $\implies$ decision.
Automated Verification

\[ G = (G, S_G, S_T) \]
Automated Verification

\[ G = (G, S_G, S_T) \]

(Underlying graph)
Automated Verification

\[ G = (G, S_G, S_T) \]

(Underlying graph)

(Transitive closure of journeys)
Automated Verification

\[ G = (G, S_G, S_T) \]

\( G \) (Underlying graph) \( H \) (Transitive closure of journeys) \( H_{\text{strict}} \) (Transitive closure of strict journeys)

\[ A. Casteigts, S. Chaumette, A. Ferreira \]
Characterizing Topological Assumptions of Dist. Algo. in Dynamic Networks
Automated Verification

\[ G = (G, S_G, S_T) \]

(G: Underlying graph) (H: Transitive closure of journeys) (H\_strict: Transitive closure of strict journeys)

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

\[ \text{Sufficient Condition} \]
\[ ? \]
\[ \text{Necessary Condition} \]
## Limitations

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

![Diagram]

?  

---  

Necessary Condition
Limitations

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

Intermediate condition?

Necessary Condition
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?