Distributed Computing Models & Algorithms

Sébastien Tixeuil
sebastien.tixeuil@lip6.fr

Three Domains

Three Approaches

Complexity-driven

Complexity

Model A

Problem 1
Complexity-driven

Model-driven

Problem-driven

A Map of Models
A Map of Models

Model A ← Model B → Model C ← Model D → Model E → Model F

Static Networks

Parallel Computing

Parallel Computing
Parallel Computing

- Tractable Sequential Problems
- Homogeneity
- Synchrony
- Reliable
- Focus on Efficiency

Distributed Computing
Intrinsically distributed problems
Heterogeneity
Asynchrony
Unreliable
Focus on **Computability and Complexity**

A distributed system is one in which the failure of a computer you didn’t even know existed can render your own computer unusable.

*Leslie Lamport*
Asynchronous Distributed Execution

- **Sequence** of « processor or link » actions
- **(liveness)** Each processor executes an infinite number of actions (or terminates)
- **(liveness)** Each enabled link action eventually occurs

**Client-Server**

- **Initially:**
  - **Send** Request to Server
- **Upon receipt of Response from Server:**
  - **Terminate**
- **Upon receipt of Request from Client:**
  - **Send** Response to Client
Synchronous Distributed Execution

- **Alternating sequence** of processor and link phases
  - In the **processor phase**, all processors that have not terminated execute their actions
  - In the **link phase**, all links execute their actions
Space-Time Diagram

Flooding

Synchronous Flooding

Synchronous Flooding
Synchronous Flooding

Asynchronous Flooding

Asynchronous Flooding

Asynchronous Flooding
Asynchronous Flooding

Asynchronous Flooding

Asynchronous Flooding

Asynchronous Flooding
Executions

Synchronous vs. Asynchronous

Synchronous vs. Asynchronous

Leader Election
Leader Election

- Message complexity?
  - Lower bound?
- Weaker model?
  - No IDs?
  - No Orientation?
  - General communication graph?

Static Networks

Passively Mobile Networks

Mobility-induced Dynamic Networks
Mobility-induced Dynamic Networks

Static Algorithms for Mobile Networks

Mobility-induced Dynamic Networks

Link Lifetime
The WiMAX bandwidth and signal strength in the V2I scenario for LOS and NLOS environments are shown in Figs. 4c and 4d respectively. The results represent a run that lasted rather than the time.

On the use of WiMAX and Wi-Fi to provide in-vehicle connectivity and media distribution, Lerotholi S. Majela; Marthinus J. Booyzen, Industrial Technology (ICIT), 2013 IEEE International Conference on

Fig. 4b shows a graph of the two vehicles traveling in opposite directions at an average relative speed of 64 km/h in an urban area. The average contact time recorded was 33s and the average maximum communication range was found to be 302 m with an average bandwidth of 13.7 Mbps per test run taken over the period of established contact, average jitter of 1.88 ms and an average of 51.7 MB data transferred per contact period. The maximum peak bandwidth of 31.7 Mbps
Round-trip Time

<table>
<thead>
<tr>
<th>Location</th>
<th>USA</th>
<th>London</th>
<th>Tokyo</th>
<th>Paris</th>
<th>Oslo</th>
<th>Amsterdam</th>
<th>Auckland</th>
<th>Berlin</th>
<th>Copenhagen</th>
<th>Dallas</th>
<th>Moscow</th>
<th>New York</th>
<th>Paris</th>
<th>Stockholm</th>
<th>Tokyo</th>
</tr>
</thead>
<tbody>
<tr>
<td>Amsterdam</td>
<td>441.7ms</td>
<td>204.819ms</td>
<td>396.819ms</td>
<td>368.819ms</td>
<td>196.819ms</td>
<td>205.819ms</td>
<td>249.819ms</td>
<td>202.819ms</td>
<td>249.819ms</td>
<td>202.819ms</td>
<td>249.819ms</td>
<td>202.819ms</td>
<td>249.819ms</td>
<td>202.819ms</td>
<td>249.819ms</td>
</tr>
<tr>
<td>Auckland</td>
<td>559.232ms</td>
<td>278.013ms</td>
<td>349.013ms</td>
<td>318.013ms</td>
<td>170.013ms</td>
<td>223.013ms</td>
<td>254.013ms</td>
<td>216.013ms</td>
<td>254.013ms</td>
<td>216.013ms</td>
<td>254.013ms</td>
<td>216.013ms</td>
<td>254.013ms</td>
<td>216.013ms</td>
<td>254.013ms</td>
</tr>
<tr>
<td>Berlin</td>
<td>215.71ms</td>
<td>201.013ms</td>
<td>285.013ms</td>
<td>255.013ms</td>
<td>135.013ms</td>
<td>188.013ms</td>
<td>159.013ms</td>
<td>121.013ms</td>
<td>159.013ms</td>
<td>121.013ms</td>
<td>159.013ms</td>
<td>121.013ms</td>
<td>159.013ms</td>
<td>121.013ms</td>
<td>159.013ms</td>
</tr>
<tr>
<td>Dallas</td>
<td>143.543ms</td>
<td>114.841ms</td>
<td>125.841ms</td>
<td>125.841ms</td>
<td>125.841ms</td>
<td>125.841ms</td>
<td>125.841ms</td>
<td>125.841ms</td>
<td>125.841ms</td>
<td>125.841ms</td>
<td>125.841ms</td>
<td>125.841ms</td>
<td>125.841ms</td>
<td>125.841ms</td>
<td>125.841ms</td>
</tr>
<tr>
<td>London</td>
<td>35.431ms</td>
<td>4.352ms</td>
<td>243.319ms</td>
<td>80.312ms</td>
<td>75.494ms</td>
<td>64.891ms</td>
<td>48.978ms</td>
<td>56.891ms</td>
<td>71.887ms</td>
<td>64.891ms</td>
<td>48.978ms</td>
<td>56.891ms</td>
<td>48.978ms</td>
<td>56.891ms</td>
<td>48.978ms</td>
</tr>
<tr>
<td>Los Angeles</td>
<td>137.283ms</td>
<td>140.844ms</td>
<td>112.285ms</td>
<td>96.898ms</td>
<td>58.887ms</td>
<td>71.887ms</td>
<td>71.887ms</td>
<td>71.887ms</td>
<td>71.887ms</td>
<td>71.887ms</td>
<td>71.887ms</td>
<td>71.887ms</td>
<td>71.887ms</td>
<td>71.887ms</td>
<td>71.887ms</td>
</tr>
<tr>
<td>Moscow</td>
<td>74.800ms</td>
<td>514.817ms</td>
<td>298.807ms</td>
<td>377.803ms</td>
<td>164.807ms</td>
<td>144.807ms</td>
<td>144.807ms</td>
<td>144.807ms</td>
<td>144.807ms</td>
<td>144.807ms</td>
<td>144.807ms</td>
<td>144.807ms</td>
<td>144.807ms</td>
<td>144.807ms</td>
<td>144.807ms</td>
</tr>
<tr>
<td>New York</td>
<td>170.84ms</td>
<td>375.843ms</td>
<td>171.84ms</td>
<td>171.84ms</td>
<td>171.84ms</td>
<td>171.84ms</td>
<td>171.84ms</td>
<td>171.84ms</td>
<td>171.84ms</td>
<td>171.84ms</td>
<td>171.84ms</td>
<td>171.84ms</td>
<td>171.84ms</td>
<td>171.84ms</td>
<td>171.84ms</td>
</tr>
<tr>
<td>Stockholm</td>
<td>50.706ms</td>
<td>213.84ms</td>
<td>123.358ms</td>
<td>110.507ms</td>
<td>181.84ms</td>
<td>181.84ms</td>
<td>181.84ms</td>
<td>181.84ms</td>
<td>181.84ms</td>
<td>181.84ms</td>
<td>181.84ms</td>
<td>181.84ms</td>
<td>181.84ms</td>
<td>181.84ms</td>
<td>181.84ms</td>
</tr>
<tr>
<td>Tokyo</td>
<td>267.793ms</td>
<td>268.811ms</td>
<td>172.841ms</td>
<td>186.878ms</td>
<td>186.878ms</td>
<td>186.878ms</td>
<td>186.878ms</td>
<td>186.878ms</td>
<td>186.878ms</td>
<td>186.878ms</td>
<td>186.878ms</td>
<td>186.878ms</td>
<td>186.878ms</td>
<td>186.878ms</td>
<td>186.878ms</td>
</tr>
</tbody>
</table>
Mobility vs. Global State

Stateless Algorithms
A routing algorithm is **stateless** if it is designed such that devices store *no information* about messages *between transmissions*. It is **stateful** otherwise.
Stateless Flooding

Stateless Flooding

Stateless Flooding

Stateless Flooding
Stateless Flooding

Flooding v2
Stateful Flooding
Stateful Flooding

- Each node is aware of its coordinates (and those of its neighbors)
- The message contains the coordinates of the destination
- **Goal**: deliver the message to the destination *without routing tables*

---

Geometric Routing

Progress vs. Distance

The notion of progress was used to define the MFR rule [84], which chooses the neighbor with the most forwarding progress within the transmission radius (see Figure 2). Similar to this rule, the greedy method in [19] selects the neighbor that minimizes the distance to the destination, which is equivalent to maximizing the advance. The angular criterion is used in Compass routing (CR)[48], where the neighbor is selected that minimizes the angle separation with respect to the destination.

Usually, the next hop in greedy forwarding is chosen among the neighbors with a positive progress (right of the dashed line in Figure 1) or with a positive advance (shaded area in Figure 1, also called greedy area). Selecting the next hop by the minimum distance or the maximum progress (MFR, greedy) gives an inherently loop-free forwarding rule independent of the unit disk graph assumption. Compass routing, which is based on the direction, is not loop-free [82].

Motivated by the observation that energy consumption can be reduced when using short links, required that the transmission range can be adjusted, the NFP (nearest with forwarding progress [38]) and NC (nearest closer [83]) criteria have been proposed. They select a neighbor which is closest to the forwarding node among all neighbors, but closer to the destination than the forwarding node itself, using distance or progress.

Which Criterion?

- **MFR**: most forwarding progress
- **CR**: minimize angular criterion
- **Greedy**: minimize distance to destination
- **NC**: nearest closer
- **NFP**: nearest with forwarding progress

2.2 Advanced Strategies

Greedy forwarding has one important drawback: it fails in local minimum situations where the forwarding node has no other neighbors closer to the destination. In some cases, a sophisticated strategy is necessary to recover from this situation; in other cases, a simple backward step is sufficient to be able to resume the greedy strategy successfully.

The GEDIR [82] method is such a greedy strategy with backward steps. Whenever a message has reached a local minimum, the packet is sent back to the previous hop, which applies the greedy rule again while excluding previously visited dead end nodes from the selection. This strategy is also loop-free.

Further improvements of the basic strategies Greedy, MFR and CR can be achieved if 2-hop information is available [82]. In this case, the forwarder selects a suitable neighbor out of the 2-hop neighborhood and forwards the packet to the direct neighbor that is connected to the selected node. Note, that 2-hop information has to be distributed, which requires a higher message overhead.

A greedy-based algorithm that goes beyond using 2-hop information is SPEED [34], which is designed to increase the relay speed. It uses an additional “backpressure” heuristic to avoid congested areas and void regions. The protocol relies on beaconing, extended by on-demand beacons for delay estimation and backpressure information. The forwarding works as follows: Nodes from the greedy area, whose relay speed is above a certain threshold, are selected probabilistically. The higher the relay speed the higher the probability to be selected. If no neighbor meets the relay speed requirement, the node drops the packet with a certain probability that depends on the failure ratio of packet forwarding to the neighbors. The necessary information to derive the failure ratio is gained from the neighbors by backpressure.
Planar Graph Routing

Planar Graphs!

Variables: previous hop, packet in greedy mode, distance to target, recovery mode.

Planar Graphs!
Greedy / Face / Greedy

Figure 7: Combined greedy/face routing: After reaching local minimum $u$ in greedy mode (dashed arrow), a face traversal is started (solid arrow) until a node $v$ is found that is closer to the target than $u$.

When using such combined algorithms, the greedy part can be performed using all links of the unit disk graph, while face routing needs a local planar subgraph. We will see in the next section how a planar subgraph can be constructed.

3.2 Planarization

In their paper on face routing, Bose et al. [10] proposed a local planar subgraph construction based on the so-called Gabriel graph (GG) [27]. The Gabriel graph of a given point set contains an edge $uv$ if Thales’ circle on $uv$, i.e. the circle having $uv$ as diameter, is empty. This circle is also called Gabriel circle over $uv$ within this context. The Gabriel graph is known to be planar and connected.

This construction rule can be applied locally to a node’s 1-hop neighborhood in order to extract a planar subgraph. The Gabriel graph construction and the so-called relative neighborhood graph (RNG) [86, 39] are the two most prominent local planarization schemes. Planarization using the GG criterion removes an edge $uv$ if Thales’ circle on $uv$ contains another node $w$. Following the RNG criterion, an edge $uv$ is eliminated, if the intersection of two circles with radius $|uv|$ centered at $u$ and $v$ contains another node $w$ (see Figure 8). Applying the GG or RNG criterion to a unit disk graph yields a planar and connected graph, if the unit disk graph is connected.

Self-stabilization

Example

$$U_0 = a$$

$$U_{n+1} = \frac{U_n}{2} \text{ if } U_n \text{ is even}$$

$$U_{n+1} = \frac{3U_n+1}{2} \text{ if } U_n \text{ is odd}$$

Example

$$U_0 = a$$

$$U_{n+1} = \frac{U_n}{2} \text{ if } U_n \text{ is even}$$

$$U_{n+1} = \frac{3U_n+1}{2} \text{ if } U_n \text{ is odd}$$

<table>
<thead>
<tr>
<th>$n$</th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
<th>10</th>
<th>11</th>
<th>12</th>
</tr>
</thead>
<tbody>
<tr>
<td>$U_n$</td>
<td>7</td>
<td>11</td>
<td>17</td>
<td>26</td>
<td>13</td>
<td>20</td>
<td>10</td>
<td>5</td>
<td>8</td>
<td>4</td>
<td>2</td>
<td>1</td>
<td>2</td>
</tr>
</tbody>
</table>
Example

\[ U_0 = a \]

\[ U_{n+1} = \frac{U_n}{2} \text{ if } U_n \text{ is even} \]

\[ U_{n+1} = \frac{3U_n + 1}{2} \text{ if } U_n \text{ is odd} \]
Self-stabilization

Distributed Systems

- **Configuration**: product of the local states of system components
- **Execution**: interleaving of the local executions of the system components

Distributed Systems

- **Classical**: Starting from a particular initial configuration, the system immediately exhibits correct behavior
- **Self-stabilizing**: Starting from any initial configuration, the system eventually reaches a configuration from which its behavior is correct

- **Self-stabilizing**: Starting from any initial configuration, the system eventually reaches a configuration from which its behavior is correct
- Defined by Dijkstra in 1974
- Advocated by Lamport in 1984 to address fault-tolerant issues
- Stale states due to mobility can be recovered!
Example of a sequential program:

```c
int x = 0;
...
if( x == 0 ) {
    // code assuming x equals 0
} else {
    // code assuming x does not equal 0
}
```
Atomicity

- A "stabilizing" sequential program

```java
int x = 0;
...
while( x == x ) {
    x = 0;
    // code assuming x equals 0
}
```

Problem

```java
0 iconst_0
1 istore_1
2 goto 7
5 iconst_0
6 istore_1
7 iload_1
8 iload_1
9 if_icmpeq 5
```
Communications

Example

- **Shared memory**: in one atomic step, read the state of all neighbors and write own state

- **Guarded command**

  ![Guarded command diagram]

  \[
  \text{Guard } \rightarrow \text{Action}
  \]

  Executed if Guard is true

**Example**

\[
\text{true } \rightarrow \text{Distance}_i := \min_{j \in \text{Neighbors}_i} \{\text{Distance}_j + 1\}
\]

**Example**

\[
\text{true } \rightarrow \text{Distance}_i := \min_{j \in \text{Neighbors}_i} \{\text{Distance}_j + 1\}
\]
true → \text{Distance}_i := \min_{j \in \text{Neighbors}_i} \{\text{Distance}_j + 1\}
true → \( \text{Distance}_i := \text{Min}_{j \in \text{Neighbors}_i} \{ \text{Distance}_j + 1 \} \)
true → \( \text{Distance}_i := \text{Min}_{j \in \text{Neighbors}_i} \{ \text{Distance}_j + 1 \} \)
Scheduling

- **Scheduler** (a.k.a. **Daemon**): the daemon chooses among activatable processes those that will execute their actions
  - can be seen as an **adversary** whose role is to prevent stabilization

Spatial Scheduling

\[
true \rightarrow \text{color}_i := \text{Min}\{\Delta \setminus \{\text{color}_j | j \in \text{Neighbors}_i\}\}
\]

\[
\Delta = \{\text{red}, \text{green}, \text{blue}, \text{yellow}\}
\]

Temporal Scheduling

- **Token** → pass token to left neighbor with probability \(\frac{1}{2}\)
  - token = ⬤ no token = ⬤
Temporal Scheduling

\[ \text{token} \rightarrow \text{pass token to left neighbor with probability } \frac{1}{2} \]

\[ \text{token} = \bullet \quad \text{no token} = \bigcirc \]

A Map of Daemons

Unfair \rightarrow Weak \rightarrow Strong \rightarrow Global
A Map of Daemons

Self-stabilization

Population Protocols
Population Protocols

• **Definition**

- A Population Protocol is a 6-tuple \((X, Y, Q, I, O, T)\)
  - \(X\): Set of inputs
  - \(Y\): Set of outputs
  - \(Q\): Set of states
  - \(I\): Input mapping function, \(X \rightarrow Q\)
  - \(O\): Output mapping function, \(Q \rightarrow O\)
  - \(T\): Transition function, \(Q \times Q \rightarrow Q \times Q\)
Example 2

- Inputs:
- Outputs:
  - #black > #white ?
Example 3

- Inputs: 0 1 2 3
- Outputs: 0 1 2 3 0 1 2 3
- Sum mod 4?
Population Protocols

Time-varying Graphs

- A time-varying graph (TVG) is a 5-tuple \((V,E,T,p,l)\)
  - \(V\): set of nodes
  - \(E\): (labelled) set of edges
  - \(T\): lifetime, \(T \subseteq \tau\)
  - \(p\): presence function, \(E \times T \rightarrow \{0,1\}\)
  - \(l\): latency function, \(E \times T \rightarrow \tau\)

Dynamic Graphs
Time-varying Graphs

- A time-varying graph (TVG) is a 5-tuple \((V,E,T,p',l')\)
  - \(V\): set of nodes
  - \(E\): (labelled) set of edges
  - \(T\): lifetime, \(T \subseteq \mathcal{T}\)
  - \(p'\): node presence function, \(V \times T \rightarrow \{0,1\}\)
  - \(l'\): node latency function, \(V \times T \rightarrow \mathcal{T}\)

Evolving Graphs

Example
Evolving Graphs

Evolving Graphs

Evolving Graphs

Evolving Graphs
There exists a node (C) from which a journey reaches every other node.
There exists a node (Center) such that there exists a journey from every other node to it and back

- There exists a node $r$ from which a journey reaches every other node $1 \rightsquigarrow *$
- There exists a node $r$ such that there exists a journey from every other node to it $* \rightsquigarrow 1$
- There exists a node $r$ such that there exists a journey from every other node to to and back $1 \rightsquigarrow *$

More Classes

- There exists a journey between any two nodes $\bullet \leftrightarrow \bullet$
- There exists a roundtrip journey between any two nodes $\bullet \overleftrightarrow{\bullet}$
- There exists a journey between any two nodes infinitely often $\bullet \leftrightarrow \bullet$
- Every edge appears infinitely often $\mathcal{R}$

More Classes

- Every edge appears infinitely often, and there is an upper bound between between two occurrences $\bullet \overleftrightarrow{\mathcal{B}}$
- Every edge appears infinitely often with some period $\mathcal{P}$

A Classification

- At any time, the graph is connected
- Every spanning subgraph lasts at least $T$ time units
- Every edge appears infinitely often, and the underlying graph is a clique $\mathcal{R}$
A Classification

Mobile Agents

Problems to Solve

- Exploration (perpetual or with stop)
- Mapping
- Rendez-vous
- Black hole search
- Capturing an intruder
Models

- **Network** (anonymous vs. ID based)
- **Agents** (anonymous vs. ID based)
- **Synchrony**
- **Initial** (structural) knowledge
- **Communications** (none, peebles, whiteboards)
- **Agent memory** (infinite, bounded, constant)

Mapping

Rendez-vous

- **Two** (or more) mobile agents must meet in a graph
- They start on distinct locations
- **Computability**?
- **Complexity**?
Rendez-vous in ID Graphs

Rendez-vous in A

DFS to find Smallest ID Node

Rendez-vous in Anonymous Graphs
Rendez-vous in Anonymous Graphs

Anonymous Graphs with Known ID (1,2) Agents

Anonymous Graphs with Known ID (1,2) Agents

Anonymous Graphs with Known N, ID Agents
Anonymous Graphs with Known N, ID Agents

Black Hole Search
Black Hole Search

- A *single* black hole in the graph
- The black hole *does not disconnect* the graph
- Identify each *adjacent edge*
- Minimize #agents, #moves
Asynchronous Black Hole Search

Mobile Robots
Mobile Robots

- Autonomous (no central control)
- Homogeneous (run same algorithm)
- Identical (indistinguishable)
- Silent (no explicit communication)
Robot Life Cycle

Look

Use sensors to observe the world, get a **snapshot**

Sleep

Compute

Move

Robot Life Cycle

Look

Compute

Move

Sleep

Robot Life Cycle

Look

Use motors to **move toward** the destination point

Sleep

Compute

Move

Robot Life Cycle

Look

Compute

Move

Sleep

Robot Life Cycle

Look

Compute

Move

Sleep

remain **idle** for a while
Visibility

- Limited
- Full

Multiplicity Detection

*How many robots do you see?*

- No detection
- Weak multiplicity detection
- Strong multiplicity detection

Multiplicity

- No
- Weak
- Strong

Multiplicity

- No
- Local Weak
- Local Strong
- Global Weak
- Global Strong
Memory

Algorithm

Persistent Memory

Volatile Memory

Oblivious Robot Memory

Algorithm

Persistent Memory

Volatile Memory

Oblivious Robot Life Cycle

remain idle for a while, forget about the past

Sleep

Look

Compute

Move

Memory

Oblivious

Finite

Infinite
Two Axes
Direction and Orientation

One Axis
Direction and Orientation

Two Axes
Direction

Chirality
No Agreement

Overview

Scattering

*No two robots should occupy the same position*

- No deterministic solution
- No termination without multiplicity detection
Scattering

1 toss

$O(\log(n)\log\log(n))$ rounds


Scattering

$n$ robots: $2n^2$ destinations

$O(1)$ rounds


How Many Tosses?

Minimum?
Influence of multiplicity detection?
Relationship with scattering speed?

Optimal Speed

With strong multiplicity detection:
Algorithm with optimal #tosses terminates in $O(1)$ rounds

Without strong multiplicity detection:

$n$ robots $\rightarrow$ finite #destinations $\rightarrow$ max #destinations is independent of $n$

$O(1)$ rounds scattering of $n$ robots is impossible
How fast can we go?


Gathering
Gathering

Impossible for two robots

A bivalent configuration

Gathering vs. Convergence

- **Gathering**: robot must reach the same point in finite time
- **Convergence**: robots must get closer at time goes by
Center of Gravity

\[
\bar{c}[t] = \frac{1}{n} \sum_{i=1}^{n} \bar{r}_i[t]
\]

Center of Gravity

\[
\bar{c}[t] = \frac{1}{n} \sum_{i=1}^{n} \bar{r}_i[t]
\]

Center of Gravity

\[
\bar{c}[t] = \frac{1}{n} \sum_{i=1}^{n} \bar{r}_i[t]
\]

Center of Gravity of Positions

\[
\bar{c}[t] = \frac{1}{P} \sum_{i=1}^{P} \bar{p}_i[t]
\]
FSYNC Gathering

\[ \hat{c}(t) = \frac{1}{p} \sum_{i=1}^{p} \tilde{p}_i[t] \]

\[ \delta_0 < \delta \]

SSYNC Gathering?

SSYNC Gathering?
### Convergence & Gathering

<table>
<thead>
<tr>
<th></th>
<th>Convergence</th>
<th>2-Gathering</th>
<th>n-Gathering</th>
<th>n-Gathering +MD</th>
<th>n-Gathering +MD+WF</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>FSYNC</strong></td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td><strong>SSYNC</strong></td>
<td>Yes</td>
<td>No</td>
<td>No</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td><strong>ASYNC</strong></td>
<td>Yes</td>
<td>No</td>
<td>No</td>
<td>Yes</td>
<td>?</td>
</tr>
</tbody>
</table>


### Convergence & Gathering

<table>
<thead>
<tr>
<th></th>
<th>Convergence</th>
<th>2-Gathering</th>
<th>n-Gathering</th>
<th>n-Gathering + MD</th>
<th>n-Gathering + MD + WF</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>FSYNC</strong></td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td><strong>SSYNC</strong></td>
<td>Yes</td>
<td>No</td>
<td>No</td>
<td>Yes</td>
<td>?</td>
</tr>
<tr>
<td><strong>SYNC</strong></td>
<td>Yes</td>
<td>No</td>
<td>No</td>
<td>Yes</td>
<td>?</td>
</tr>
</tbody>
</table>


### Convergence & Gathering

<table>
<thead>
<tr>
<th></th>
<th>Convergence</th>
<th>2-Gathering</th>
<th>n-Gathering</th>
<th>n-Gathering + MD</th>
<th>n-Gathering + MD + WF</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>FSYNC</strong></td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td><strong>SSYNC</strong></td>
<td>Yes</td>
<td>No</td>
<td>No</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td><strong>SYNC</strong></td>
<td>Yes</td>
<td>No</td>
<td>No</td>
<td>Yes</td>
<td>?</td>
</tr>
</tbody>
</table>


### Pattern Formation

<table>
<thead>
<tr>
<th></th>
<th>Convergence</th>
<th>2-Gathering</th>
<th>n-Gathering</th>
<th>n-Gathering + MD</th>
<th>n-Gathering + MD + WF</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>FSYNC</strong></td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td><strong>SSYNC</strong></td>
<td>Yes</td>
<td>No</td>
<td>No</td>
<td>Yes</td>
<td>?</td>
</tr>
<tr>
<td><strong>SYNC</strong></td>
<td>Yes</td>
<td>No</td>
<td>No</td>
<td>Yes</td>
<td>?</td>
</tr>
</tbody>
</table>

Quentin Bramas, Sébastien Tixeuil. Wait-Free Gathering Without Chirality. SIROCCO 2015: 313-327
Pattern Formation

The goal is to form the pattern, and then stay stationary

In general, NO

What about restricting the set of initial configurations?
What about adding conditions on the pattern?
What about adding capabilities to robots?

All robots are here
No, so from now, we assume the initial configuration does not have points of multiplicity

Pattern Formation

Yes, if robots agree on a common North and a common Right
Yes, if robots agree on a common North and \( n \) is odd

---

Nao Fujinaga, Yukiko Yamauchi, Hirotaka Ono, Shuji Kijima, Masafumi Yamashita:
Pattern Formation

Initial configuration $P$ → Pattern to form $F$

...assuming a common chirality, and $F$ does not have multiplicity points

Yes, if $\rho(P) | \rho(F)$ where $\rho(P)$ is the symmeetricity of $P$,
the maximum integer such that the rotation by $2\pi/\rho(P)$
is invariant for $P$.

No, otherwise

Nao Fujinaga, Yukiko Yamauchi, Hirotaka Ono, Shuji Kijima, Masafumi Yamashita:

Pattern Formation

Initial configuration $P$ → Pattern to form $F$

...assuming a common chirality, and $F$ does not have multiplicity points

Yes, with a randomized algorithm

... assuming robots do not "pause" while moving
... and using infinitely many random bits per activation

Yes, otherwise

Yukiko Yamauchi, Masafumi Yamashita: Randomized Pattern Formation Algorithm for Asynchronous Oblivious Mobile Robots. DISC 2014: 137-151

Pattern Formation

Initial configuration $P$ → Pattern to form $F$

...assuming a common chirality, and $F$ does not have multiplicity points

Yes, with a randomized algorithm

... assuming robots do not "pause" while moving
... and using infinitely many random bits per activation

No, otherwise

ASYNC Pattern Formation

<table>
<thead>
<tr>
<th>Pattern</th>
<th>Agreement</th>
<th>Chirality</th>
<th>Randomization</th>
</tr>
</thead>
<tbody>
<tr>
<td>Point</td>
<td>Yes</td>
<td>No</td>
<td>?</td>
</tr>
<tr>
<td>Divide Symmetry</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>NoMultiplicity</td>
<td>Yes</td>
<td>No</td>
<td>Yes</td>
</tr>
<tr>
<td>Not a Point</td>
<td>Yes</td>
<td>No</td>
<td>Yes</td>
</tr>
<tr>
<td>Arbitrary</td>
<td>Yes</td>
<td>No</td>
<td>?</td>
</tr>
</tbody>
</table>

Mobile Robots

- **Fundamental**, well established model
- **Space-centric**, complexity results
- **Time-centric**, computability results

Static Networks

Conclusion
Mobility as an Adversary

- Can corrupt the distributed state of a network
- Can reduce communication capacity
- Can increase uncertainty
- Can increase protocol complexity

Mobility as a Friend

- Mobility can be the solution to the problem
- Mobility can improve efficiency
- Mobility can promote simplicity

Distributed Computing

Thank You