Port-Numbering Model
Distributed Systems

• Intuition:
  • distributed system
    \approx \text{communication network}
    \approx \text{network equipment + communication links}
  • distributed algorithm
    \approx \text{computer program}

• Precisely how are we going to model this?
Port Numbering
Port Numbering

- Network device = state machine with *communication ports*
- Ports are *numbered*: 1, 2, 3, ...

![Port Numbers: 1, 2, 3, 4]
Port-Numbered Network

- Network = several devices, *connections* between ports
  - we will formalise it as a triple \( N = (V, P, p) \)
Port-Numbered Network

- nodes $V = \{u, v, \ldots\}$
- ports $P = \{(u, 1), (u, 2), (u, 3), (u, 4), (v, 1), (v, 2), (v, 3), \ldots\}$
- connections $p(u, 4) = (v, 1), p(v, 1) = (u, 4), \ldots$
Port-Numbered Network

• nodes $V = \{u, v, \ldots\}$

• ports $P = \{(u, 1), (u, 2), (u, 3), (u, 4), (v, 1), (v, 2), (v, 3), \ldots\}$

• connections $p(u, 4) = (v, 1)$, $p(v, 1) = (u, 4)$, ...

not a complete example, some ports not connected!
Port-Numbered Network

- nodes $V = \{a, b, c, d\}$
- ports $P = \{(a, 1), (a, 2), (a, 3), (b, 1), (b, 2), (c, 1), (c, 2), (d, 1)\}$
- connections $p(a, 1) = (b, 1), \ p(b, 1) = (a, 1), \ ...$

all ports connected
Port-Numbered Network

- nodes \( V \) = a finite set
- ports \( P \) = a finite set of (node, number) pairs
- connections \( p \) = an involution \( P \rightarrow P \)

involution:
\[
p^{-1} = p \\
p(p(x)) = x
\]
Port-Numbered Network

- We may have *multiple connections* or *loops*
Port-Numbered Network

- *Simple* port-numbered network: no multiple connections, no loops
Port-Numbered Network

- *Underlying graph* of a simple port-numbered network
Distributed Algorithms
Distributed Algorithm

• State machine, \( x = \text{current state} \):
  • \( x \leftarrow \text{init}(z) \): initial state for local input \( z \)
  • \textbf{send}(x): construct \textit{outgoing messages}
    • \text{send}(x) = \text{vector, one element per port}
  • \( x \leftarrow \text{receive}(x, m): \text{process} \textit{incoming messages} \)
    • \( m = \text{vector, one element per port} \)
Execution

• “Execution of algorithm $A$ in network $N$”
• All nodes of $N$ are *identical copies* of the same state machine $A$
  • functions *init*, *send*, and *receive* may depend on node degree (number of ports)
  • in all other aspects the nodes are identical
Execution

- All nodes are initialised
- Time step (communication round):
  - all nodes construct outgoing messages
  - messages are propagated
  - all nodes process incoming messages
- Continue until all nodes have stopped
Communication Round

• Construct *outgoing messages*
Communication Round

- Construct outgoing messages
- Exchange messages along communication links
Communication Round

- Construct outgoing messages
- Exchange messages along communication links
Communication Round

- Construct outgoing messages
- Exchange messages along communication links
- Process *incoming messages*
Communication Round

• Construct outgoing messages
• Exchange messages along communication links
• Process incoming messages

• Communication rounds are *synchronous*
• Each step happens synchronously in parallel for all nodes
• Everything is *deterministic*
Distributed Algorithm

• Algorithm designed chooses:
  • how to initialise nodes
  • how to construct outgoing messages
  • how to process incoming messages

• Network structure determines:
  • how messages are propagated between ports
Distributed Algorithm

• “Algorithm $A$ solves graph problem $\Pi$ on graph family $\mathcal{F}$”:

  • for any graph $G \in \mathcal{F}$,

  • for \textit{any simple port-numbered network} $N$ that has $G$ as underlying graph,

  • execution of $A$ on $N$ stops and produces a valid solution of $\Pi$
Distributed Algorithm

• “Algorithm A finds a minimum vertex cover in any regular graph”:
  • for any simple port-numbered network $N$ that has a regular graph as underlying graph,
  • execution of $A$ on $N$ stops,
  • the stopping states of the nodes are “0” and “1”,
  • nodes in state “1” form a minimum vertex cover
Example

• Design a distributed algorithm that finds a *minimum vertex cover* in $\mathcal{F} = \{\circ\circ\circ\circ, \circ\circ\circ\circ\circ\circ\}$
• Design a distributed algorithm that finds a *minimum vertex cover* in
  \( \mathcal{F} = \{\circ\cdots\circ\circ, \circ\cdots\circ\circ\circ\circ\circ\circ\} \)
Example

- Nodes of degree 1:
  - init$_1 = ?$, send$_1(?) = (A)$
  - receive$_1(?, A) = 0$, receive$_1(?, B) = 0$

- Nodes of degree 2:
  - init$_2 = ?$, send$_2(?) = (B, B)$
  - receive$_2(?, A, A) = 1$, receive$_2(?, A, B) = 1$
    receive$_2(?, B, A) = 1$, receive$_2(?, B, B) = 0$
Example

• Design a distributed algorithm that finds a minimum vertex cover in $\mathcal{F} = \{\circ\circ\circ\circ, \circ\circ\circ\circ\circ\circ\}$

• Solved!

• Running time: 1 communication round
General Principles
General Principles

- Synchronous execution
  - “worst case”
  - synchronisers exist
General Principles

• Synchronous execution

• Deterministic algorithms
  • cf. the name of this course
  • nodes do not have any source of randomness
General Principles

• Synchronous execution
• Deterministic algorithms
• Anonymous networks
  • identical nodes (except for their degree)
  • Chapters 5–6: what happens if each node has a unique name
General Principles

• Synchronous execution
• Deterministic algorithms
• Anonymous networks
• Time = number of communication rounds
  • focus on communication, not computation...
Examples
Maximal Matching

• We will design distributed algorithm BMM that finds a maximal matching in any 2-coloured graph
  • we assume that we are given a proper 2-colouring of the underlying graph as input
  • algorithm will output a maximal matching
Given
encoding of 2-colouring

Find
encoding of maximal matching
Maximal Matching

- Algorithm idea:
  - white nodes send *proposals* to their ports, one by one
  - black nodes *accept* the first proposal that they get
Maximal Matching

• Algorithm idea:
  • white nodes send *proposals* to their ports, one by one
  • black nodes *accept* the first proposal that they get
  • proposal–accept pair = edge in matching

• Running time: $O(\Delta)$
  • $\Delta = \text{maximum degree}$
Maximal Matching

• We can find a maximal matching if we are given a 2-colouring
  • some auxiliary information is necessary, as we will see in Chapter 3

• Application: vertex cover approximation
  • works correctly in any network, no need to have 2-colouring!
Vertex Cover

- We will design distributed algorithm VC3 that finds a 3-approximation of minimum vertex cover in any graph
  - each node stops and outputs “0” or “1”
  - nodes that output “1” form a 3-approximation of a minimum vertex cover for the underlying graph
Vertex Cover

- Given: a port-numbered network
  - drawing here just the underlying graph...
Vertex Cover

- Construct the *bipartite double cover*: two copies of each node, edges across
Vertex Cover

- Simulate algorithm BMM, outputs a *maximal matching* $M'$
Vertex Cover

- $C =$ nodes with at least one copy matched:
  3-approximation of minimum vertex cover!
Vertex Cover

- $C =$ nodes with at least one copy matched: 3-approximation of minimum vertex cover!

- Why vertex cover?
  - assume that there is an uncovered edge
  - conclude that $M'$ is not maximal
Vertex Cover

• $C =$ nodes with at least one copy matched: 3-approximation of minimum vertex cover!

• Why vertex cover?

• Why 3-approximation?
Vertex Cover

- Idea: matching in bipartite double cover $\rightarrow$ paths and/or cycles in original graph
Vertex Cover

• Any vertex cover contains at least 1/3 of nodes of any path or cycle

• 3-approximation if we take all of these
Summary

- We can solve non-trivial problems with distributed algorithms
  - e.g., 3-approximation of minimum vertex cover

- What next?
  - week 3: problems that cannot be solved at all
  - week 4: more positive results
  - weeks 5–6: what changes if the nodes have names?