Distributed Computing on the (Fruit) Fly

Yuval Emek

Technion - Israel Institute of Technology

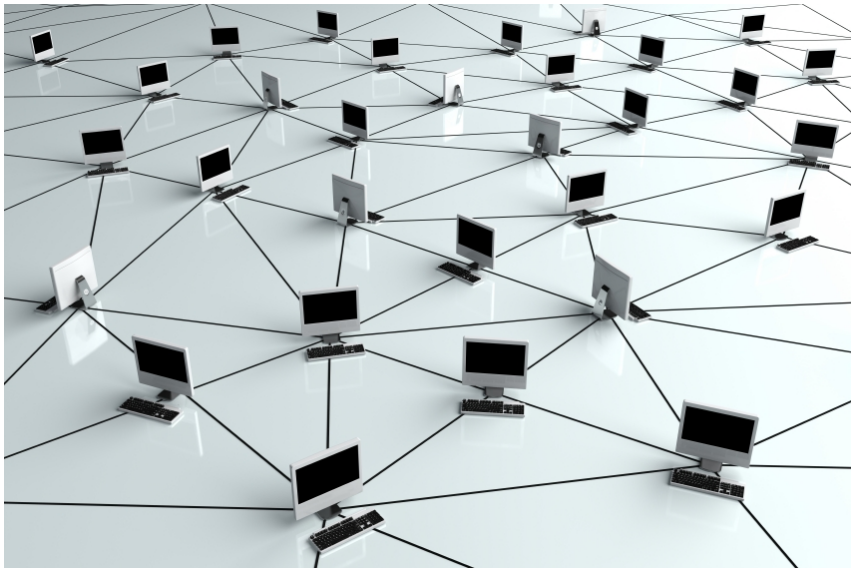
The 1st Workshop on Biological Distributed Algorithms
Jerusalem, October 2013

Synopsis

Distributed network algorithms

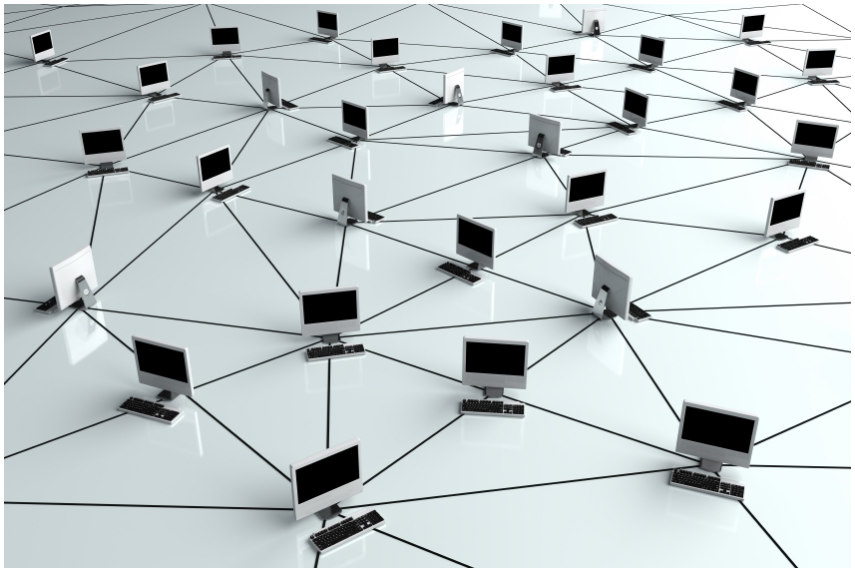
Synopsis

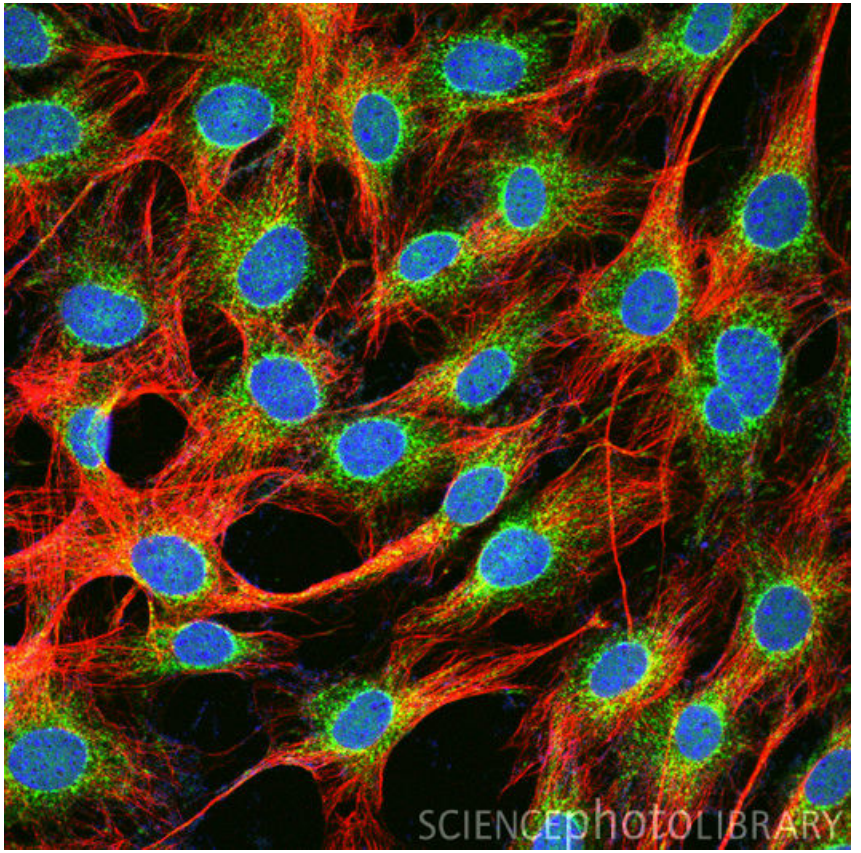
Distributed network algorithms



Synopsis

Distributed network algorithms

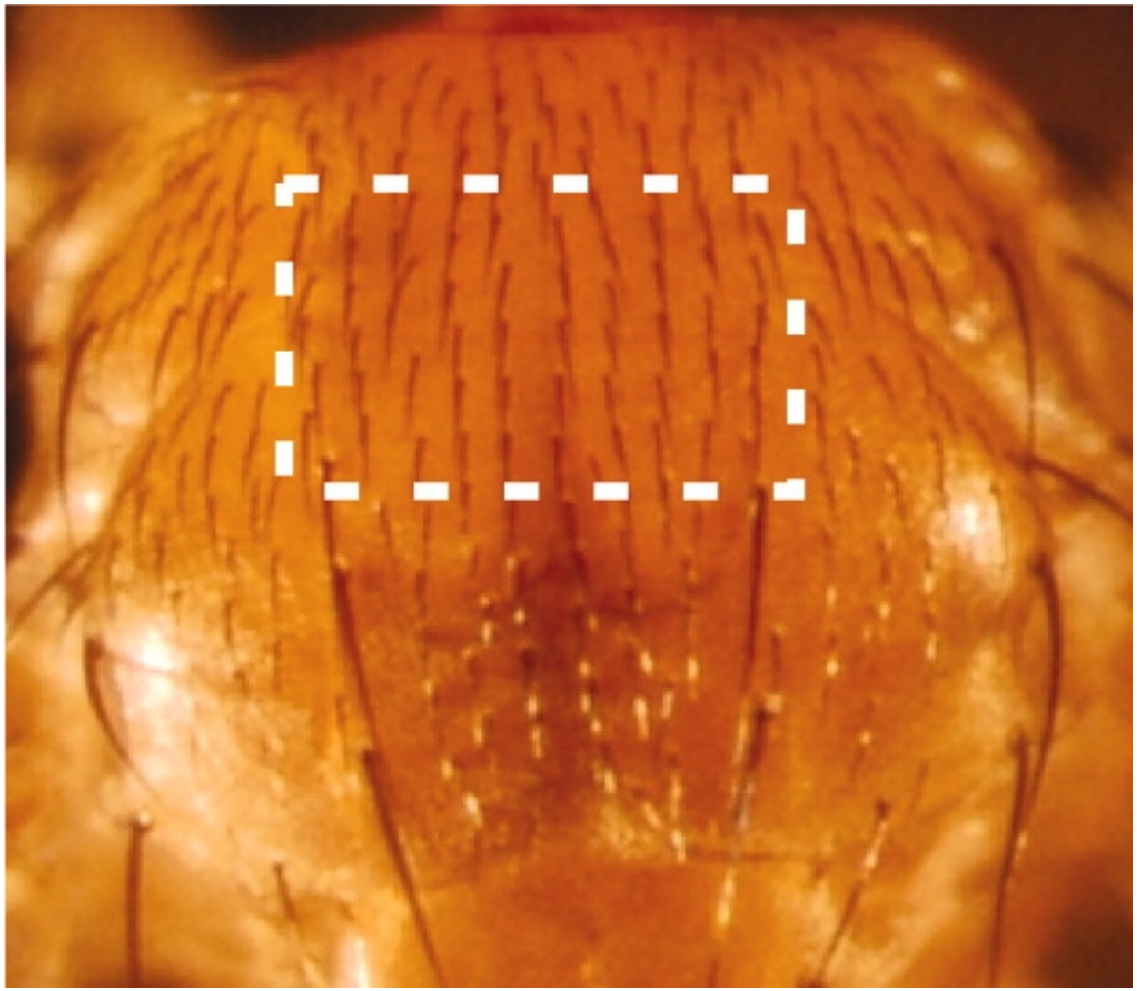




Mission: theory of distributed computing in biological cellular networks

Motivation

Selection of sensory organ precurser (SOP) cells = solving MIS
[Afek, Alon, Barad, Hornstein, Barkai, Bar-Joseph 11]

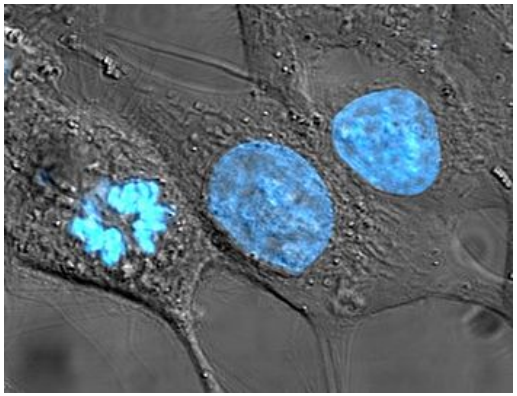


1. Cells as computing devices
2. Abstract distributed computing models
3. Networked finite state machines
4. Results
   - MIS algorithm
5. Conclusions

Local computation

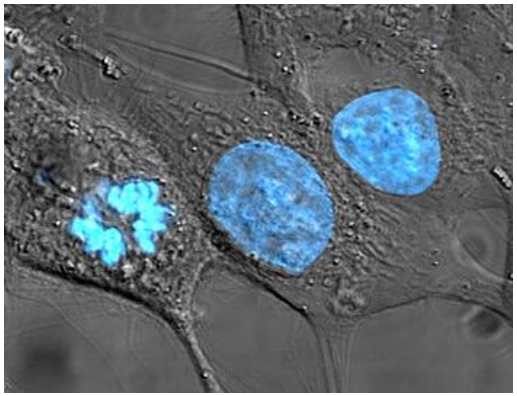
Local computation

Nucleus: analogous to central processing unit



Local computation

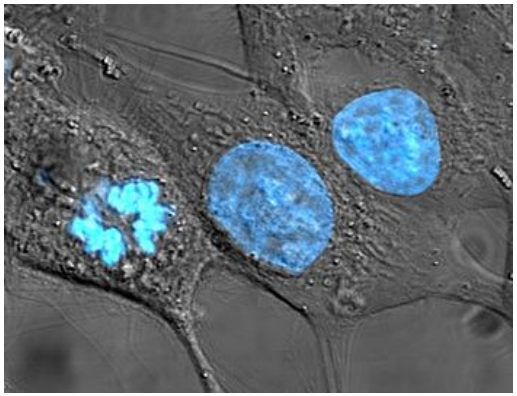
Nucleus: analogous to central processing unit



- Code = DNA (strings of nucleotides)

Local computation

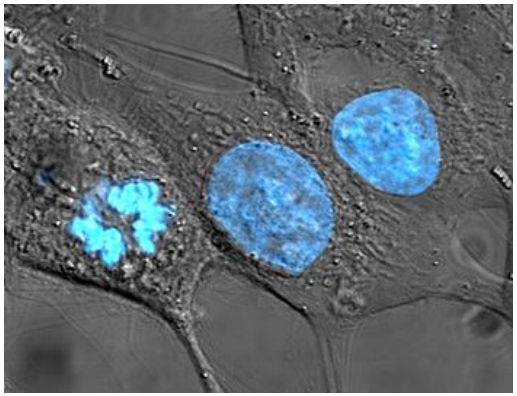
Nucleus: analogous to central processing unit



- Code = DNA (strings of nucleotides)
- Instructions = genes (DNA substrings)

Local computation

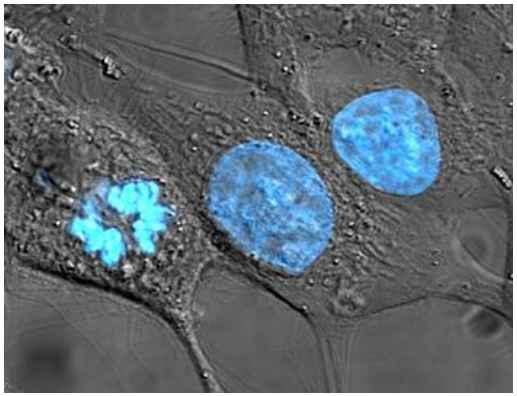
Nucleus: analogous to central processing unit



- Code = DNA (strings of nucleotides)
- Instructions = genes (DNA substrings)
- Execution = gene expression
  - transcribed to RNA molecules

Local computation

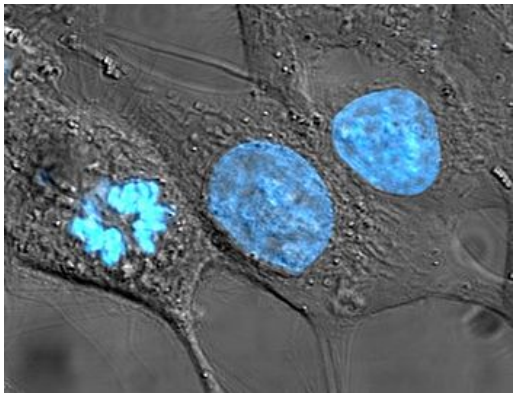
Nucleus: analogous to central processing unit



- Code = DNA (strings of nucleotides)
- Instructions = genes (DNA substrings)
- Execution = gene expression
  - transcribed to RNA molecules
- Main question:
  which genes are currently expressed?

Local computation

Nucleus: analogous to central processing unit



- Code = DNA (strings of nucleotides)
- Instructions = genes (DNA substrings)
- Execution = gene expression
  - transcribed to RNA molecules
- Main question:
  which genes are currently expressed?
  - Analogous to CPU's current state

Communication

Communication

Juxtacrine (direct contact): respects network's topology

Communication

Juxtacrine (direct contact): respects network's topology

Delivery of message m from cell x to cell y

Communication

Juxtacrine (direct contact): respects network's topology

Delivery of message m from cell x to cell y

1. x produces molecule m

Communication

Juxtacrine (direct contact): respects network's topology

Delivery of message m from cell x to cell y

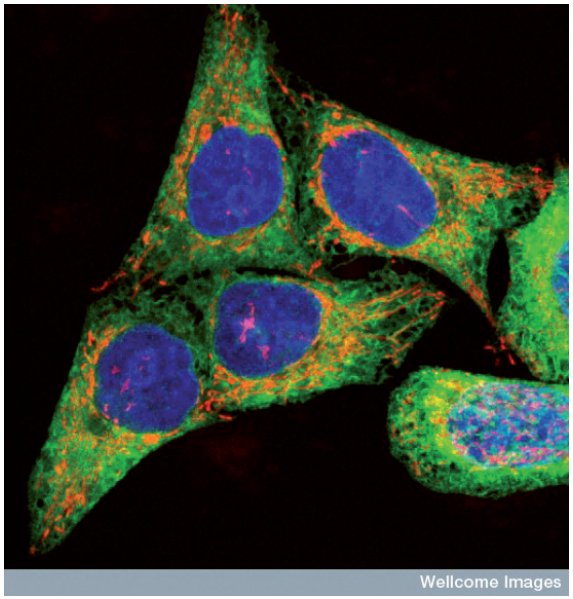
1. x produces molecule m
2. m crosses from x to y

Communication

Juxtacrine (direct contact): respects network's topology

Delivery of message m from cell x to cell y

1. x produces molecule m
2. m crosses from x to y
   - gap junction connecting two cytoplasms

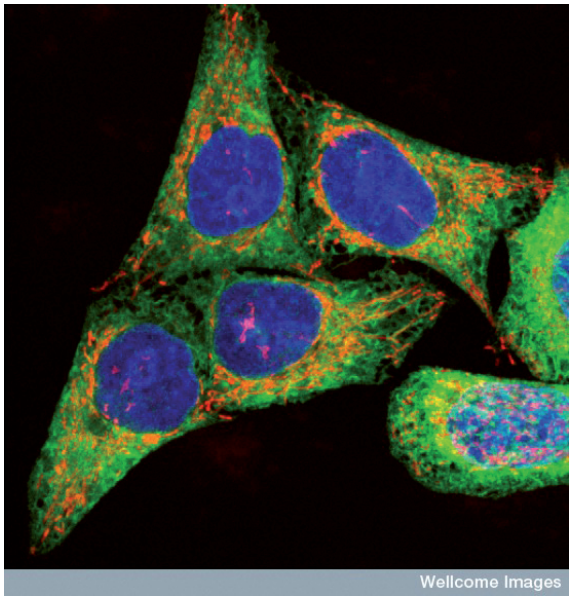


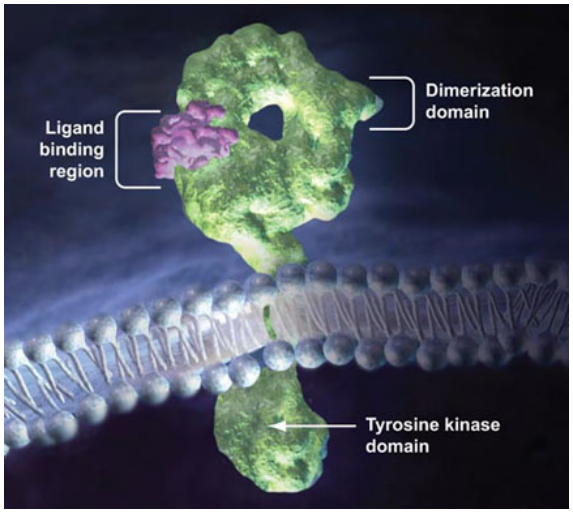
Communication

Juxtacrine (direct contact): respects network's topology

Delivery of message m from cell x to cell y

1. x produces molecule m
2. m crosses from x to y
   - gap junction connecting two cytoplasms
   - binds to crossmembrane receptor



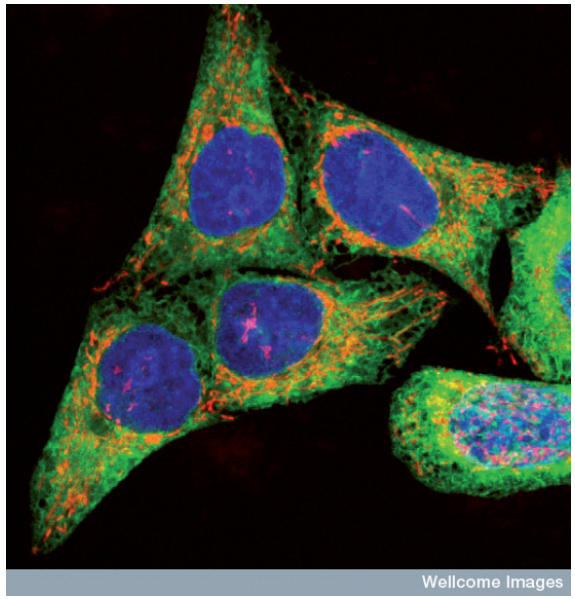


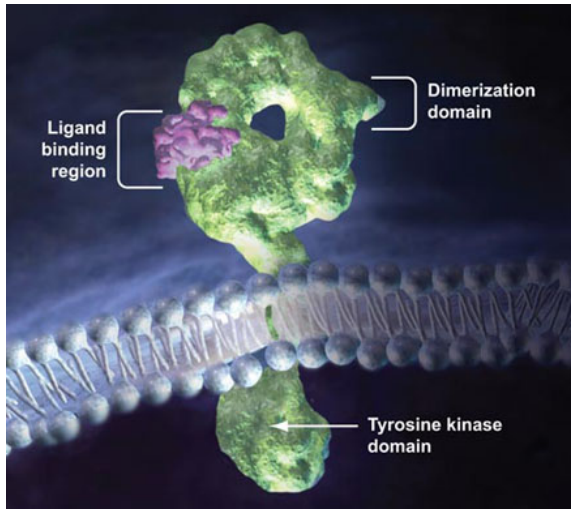
Communication

Juxtacrine (direct contact): respects network's topology

Delivery of message m from cell x to cell y

1. x produces molecule m
2. m crosses from x to y
   - gap junction connecting two cytoplasms
   - binds to crossmembrane receptor
3. Triggers a signaling cascade inside y



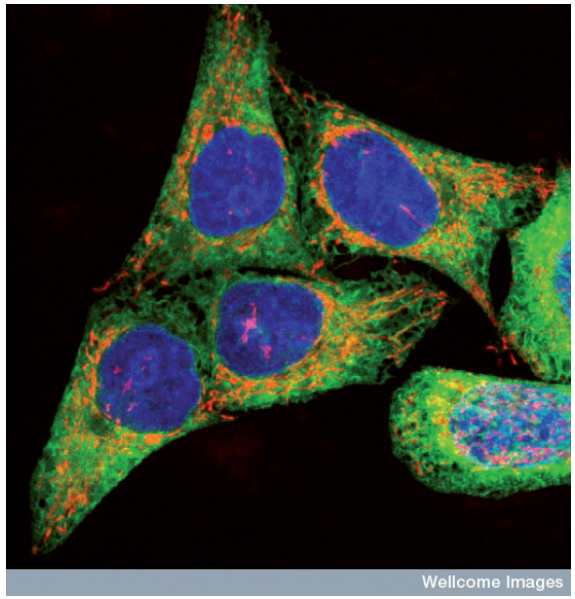


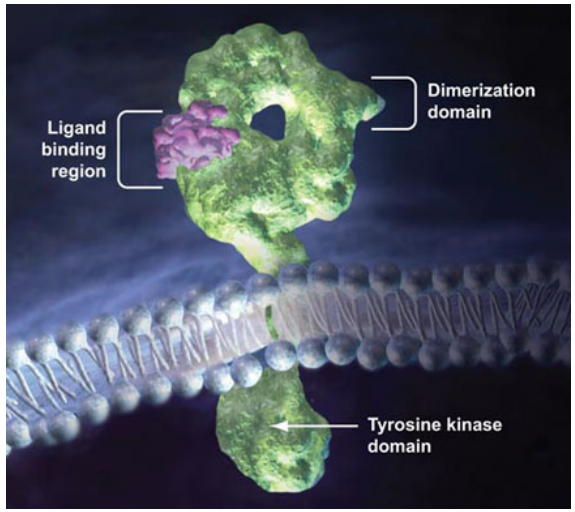
Communication

Juxtacrine (direct contact): respects network's topology

Delivery of message m from cell x to cell y

1. x produces molecule m
2. m crosses from x to y
   - gap junction connecting two cytoplasms
   - binds to crossmembrane receptor
3. Triggers a signaling cascade inside y
4. Modifies concentration levels in nucleus



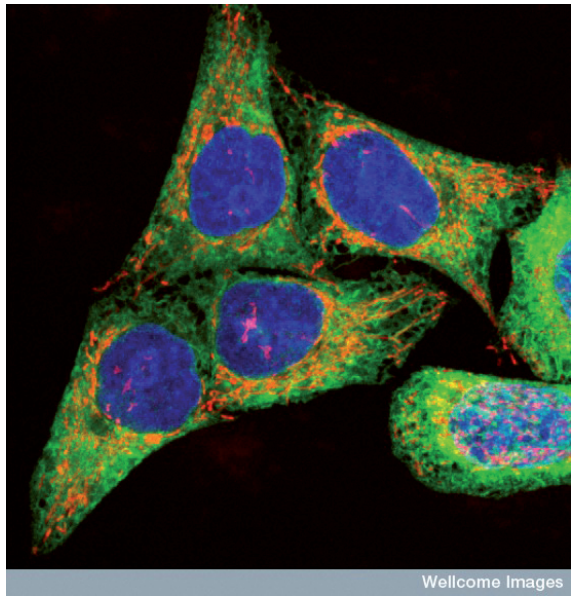


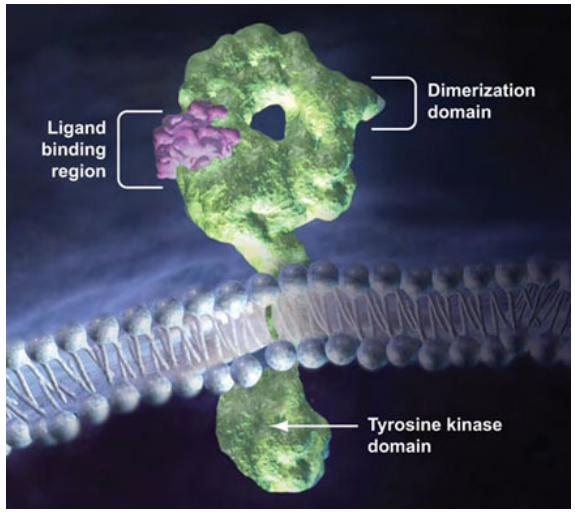
Communication

Juxtacrine (direct contact): respects network's topology

Delivery of message m from cell x to cell y

1. x produces molecule m
2. m crosses from x to y
   - gap junction connecting two cytoplasms
   - binds to crossmembrane receptor
3. Triggers a signaling cascade inside y
4. Modifies concentration levels in nucleus
5. Affects y's gene expression





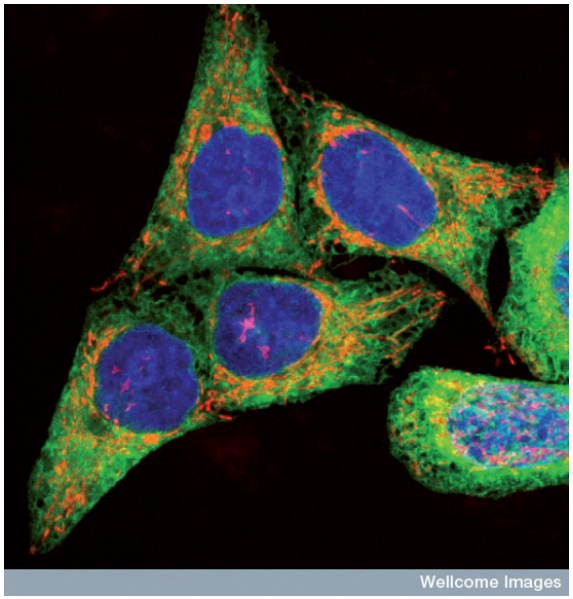
Communication

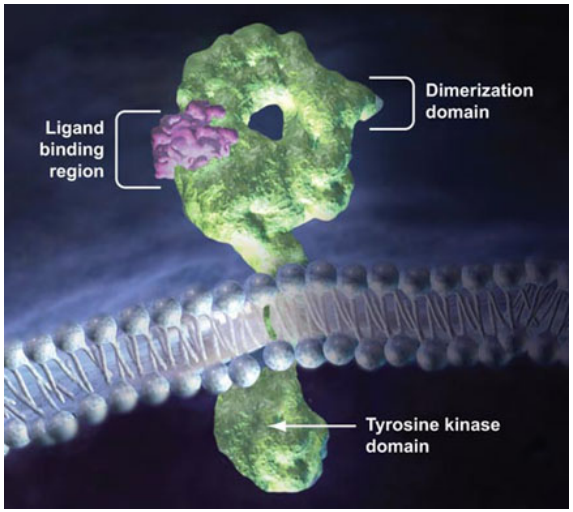
Juxtacrine (direct contact): respects network's topology

Delivery of message m from cell x to cell y

1. x produces molecule m
2. m crosses from x to y
   - gap junction connecting two cytoplasms
   - binds to crossmembrane receptor
3. Triggers a signaling cascade inside y
4. Modifies concentration levels in nucleus
5. Affects y's gene expression

- Gap junction/receptor = port





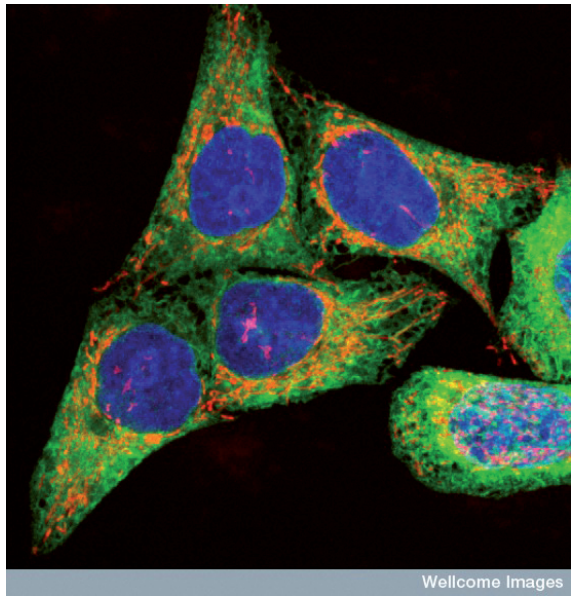
Communication

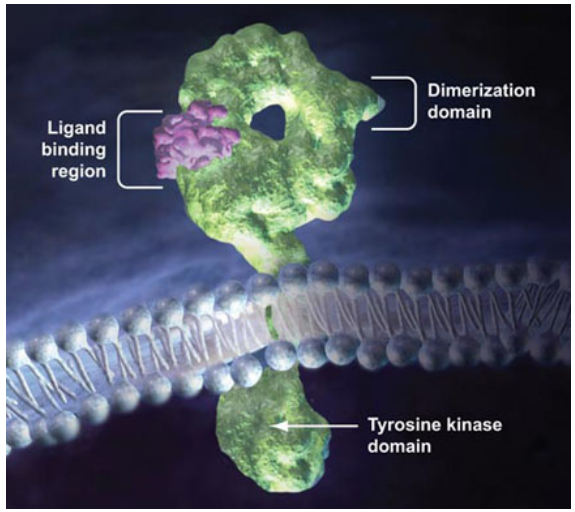
Juxtacrine (direct contact): respects network's topology

Delivery of message m from cell x to cell y

1. x produces molecule m
2. m crosses from x to y
   - gap junction connecting two cytoplasms
   - binds to crossmembrane receptor
3. Triggers a signaling cascade inside y
4. Modifies concentration levels in nucleus
5. Affects y's gene expression

- Gap junction/receptor = port
- No sense of direction
  - all neighbors look the same





1 Cells as computing devices

2 Abstract distributed computing models

3 Networked finite state machines

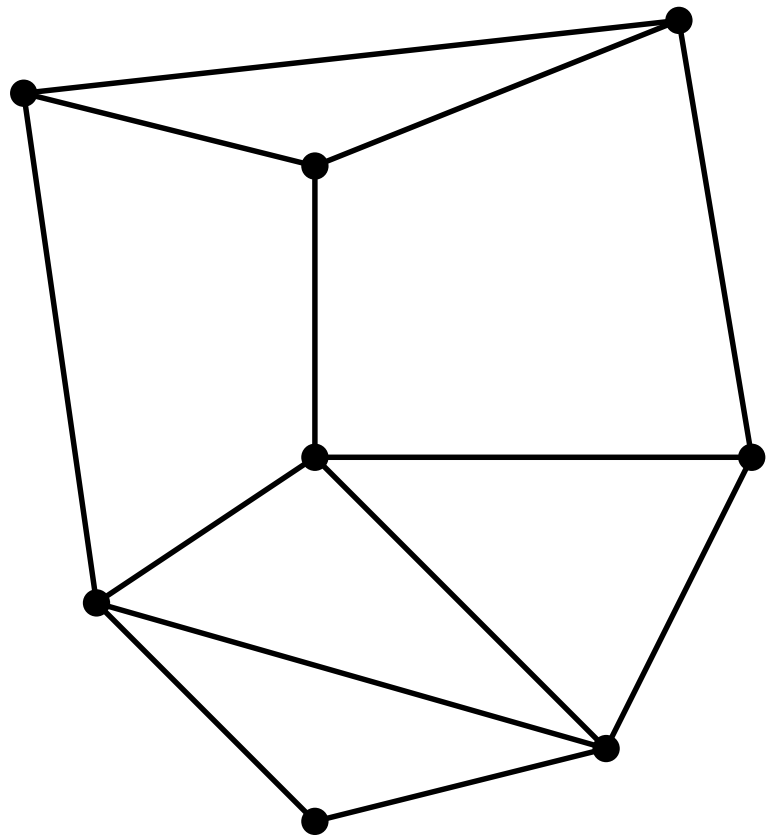
4 Results
- MIS algorithm

5 Conclusions

Message passing

Message passing

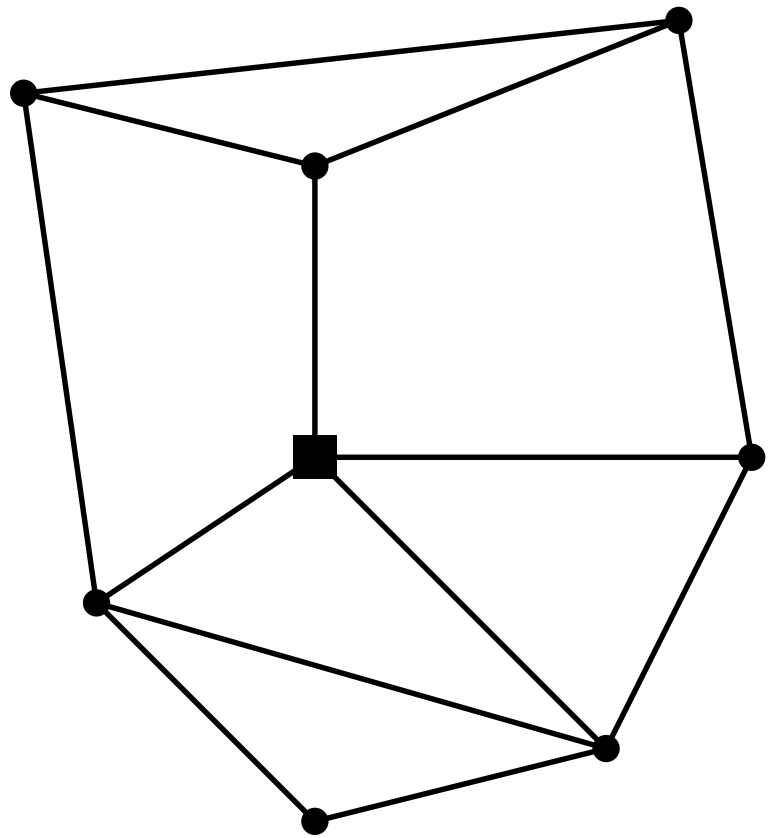
Nodes act locally (don't know global topology)



Message passing

Nodes act locally (don't know global topology)

In each step, node v:

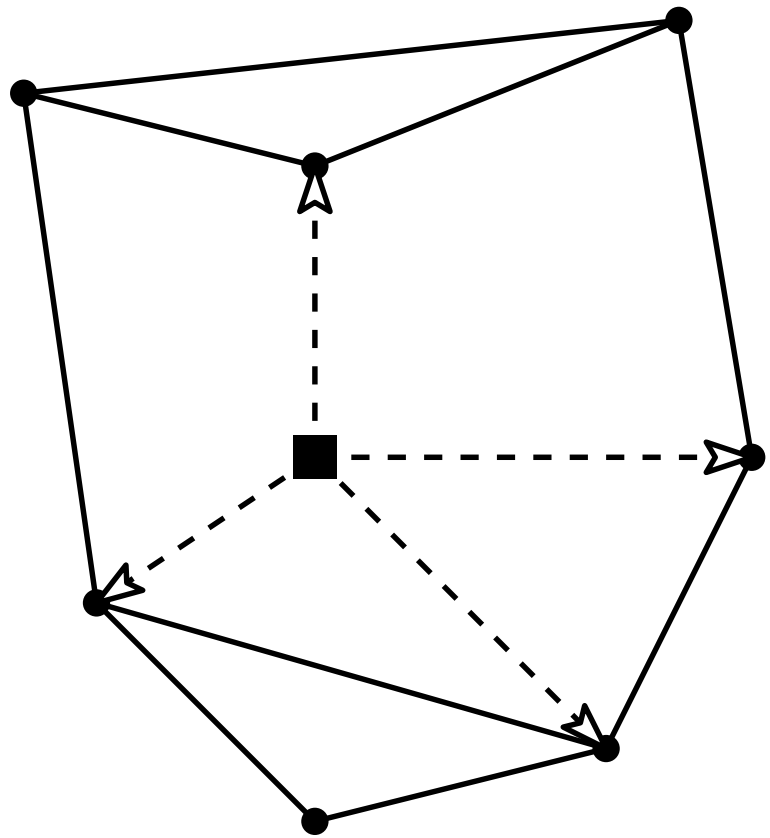


Message passing

Nodes act locally (don't know global topology)

In each step, node v:

- sends messages to $N(v)$

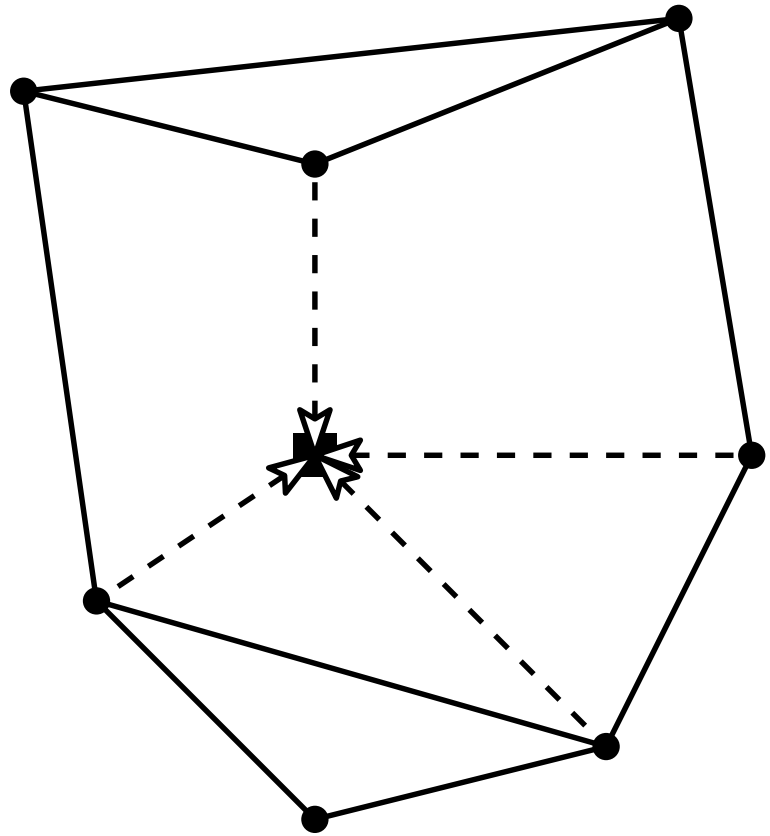


Message passing

Nodes act locally (don't know global topology)

In each step, node v:

- sends messages to $N(v)$
- receives messages from $N(v)$

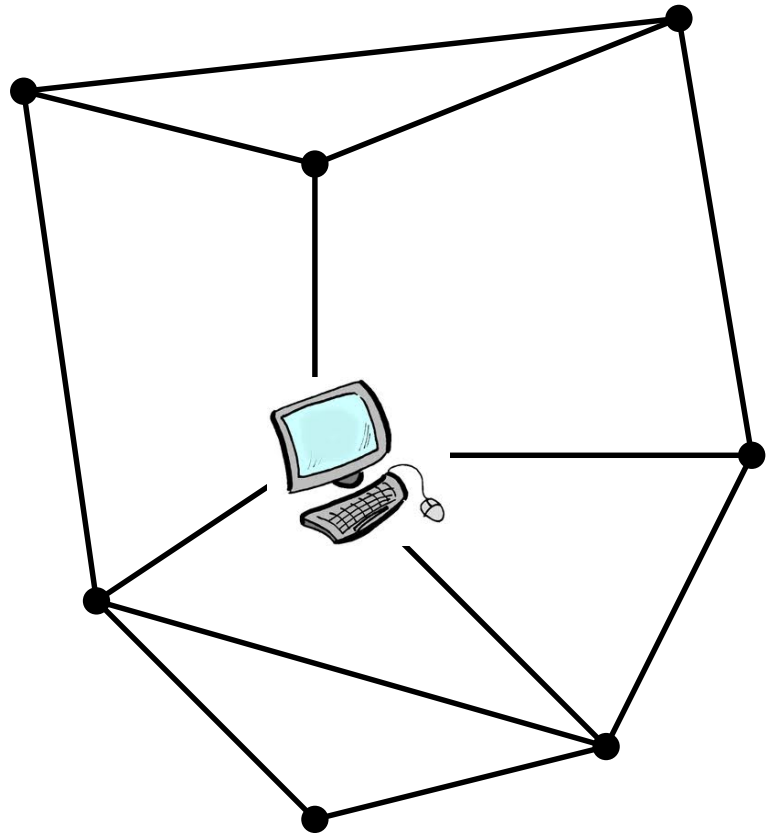


Message passing

Nodes act locally (don't know global topology)

In each step, node v:

- sends messages to $N(v)$
- receives messages from $N(v)$
- performs local computation

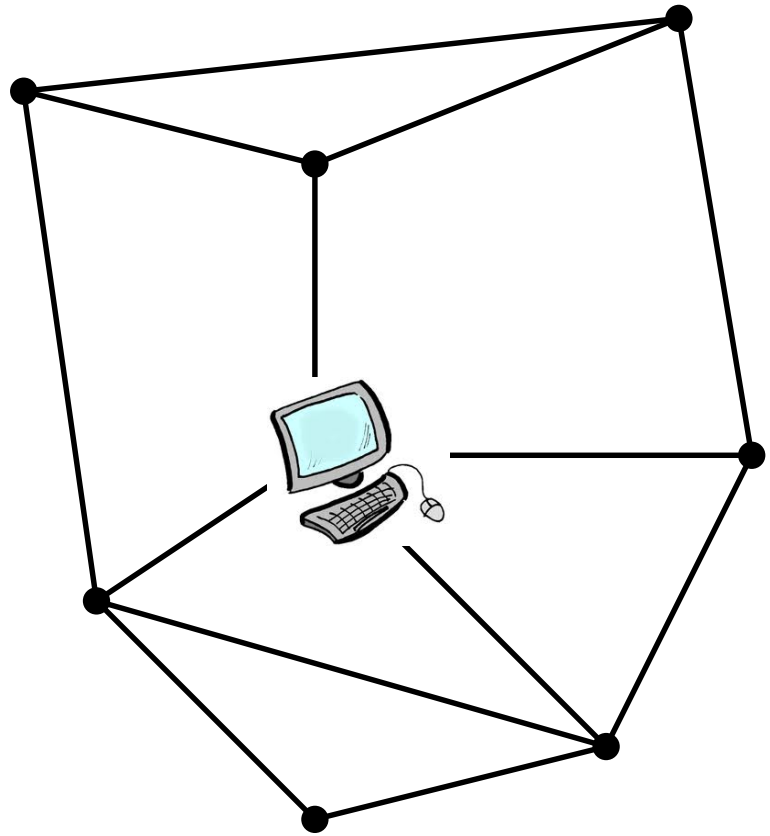


Message passing

Nodes act locally (don't know global topology)

In each step, node v:

- sends messages to $N(v)$
- receives messages from $N(v)$
- performs local computation



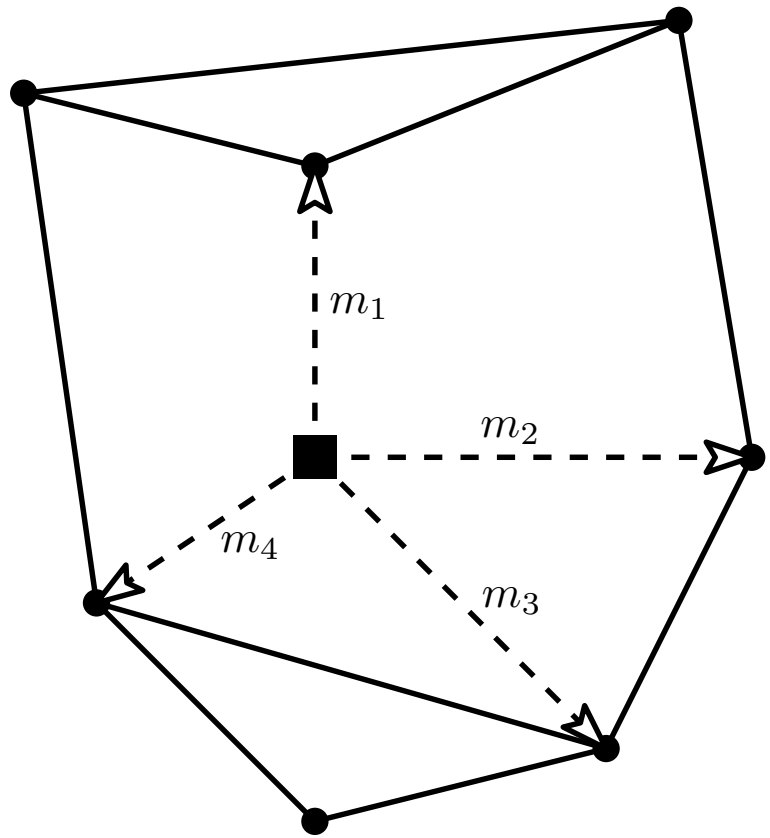
Communication too strong

Message passing

Nodes act locally (don't know global topology)

In each step, node v:

- sends messages to $N(v)$
- receives messages from $N(v)$
- performs local computation



Communication too strong

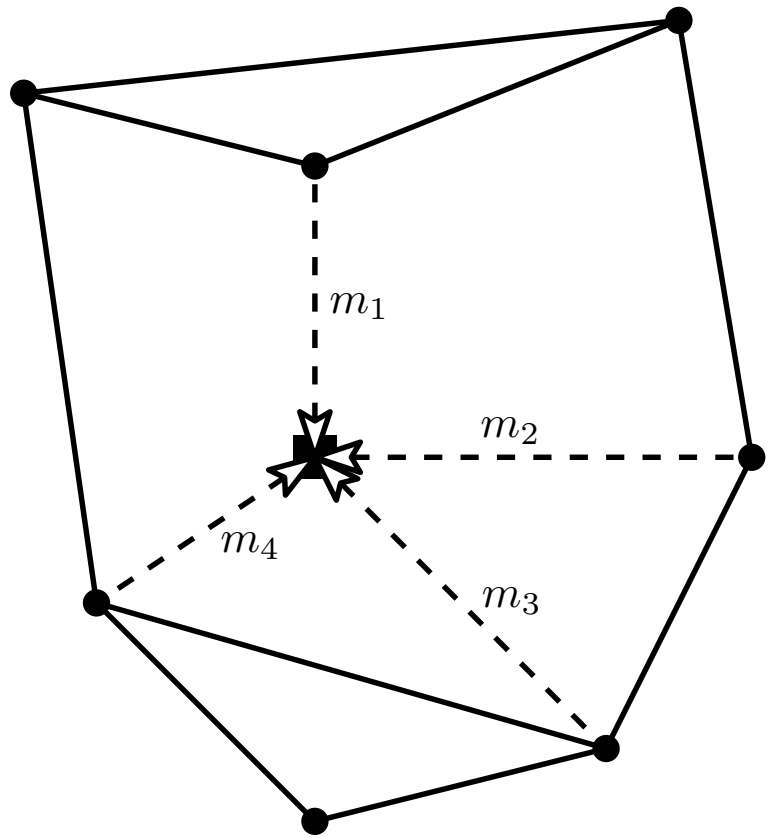
- independent messages to/from each neighbor

Message passing

Nodes act locally (don't know global topology)

In each step, node v:

- sends messages to $N(v)$
- receives messages from $N(v)$
- performs local computation

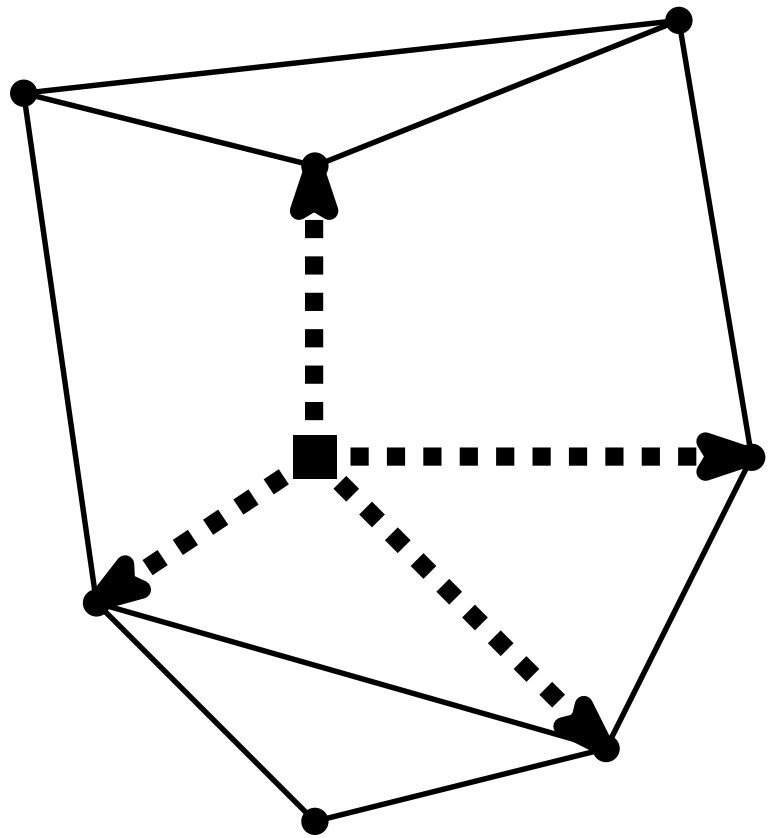


Communication too strong

- independent messages to/from each neighbor

Message passing

Nodes act locally (don't know global topology)



In each step, node v:

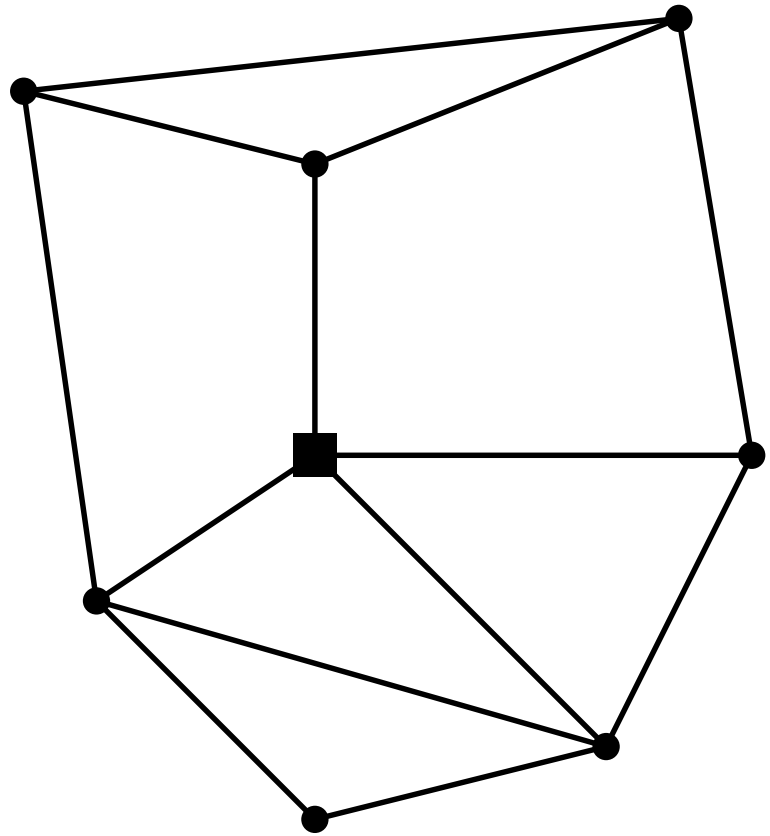
- sends messages to $N(v)$
- receives messages from $N(v)$
- performs local computation

Communication too strong

- independent messages to/from each neighbor
- # message types grows with n

The beeping model

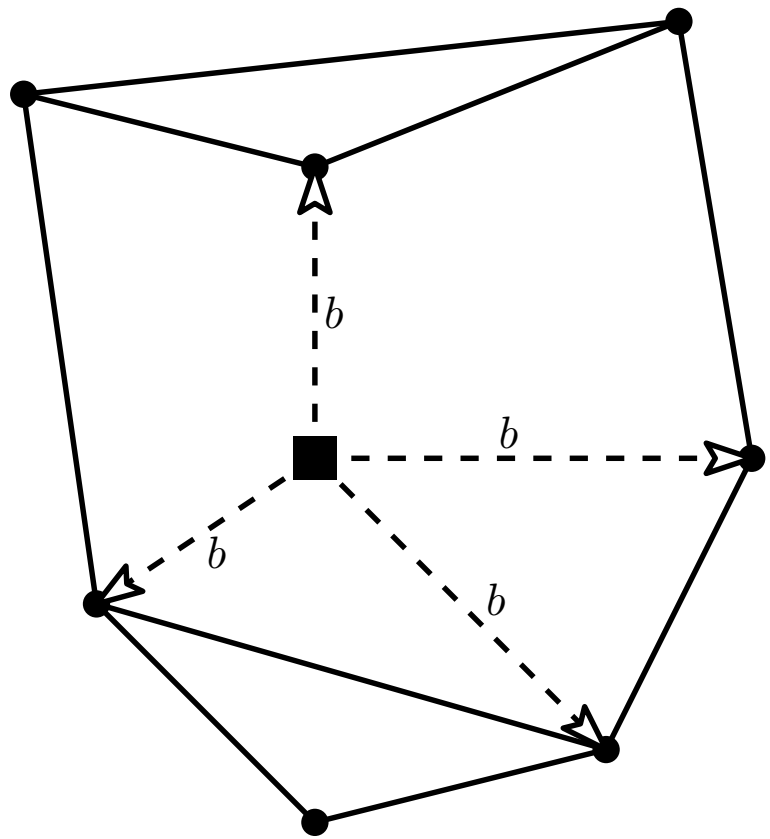
Introduced in [Cornejo, Kuhn 10]



The beeping model

Introduced in [Cornejo, Kuhn 10]

Messages = beeps (no information)

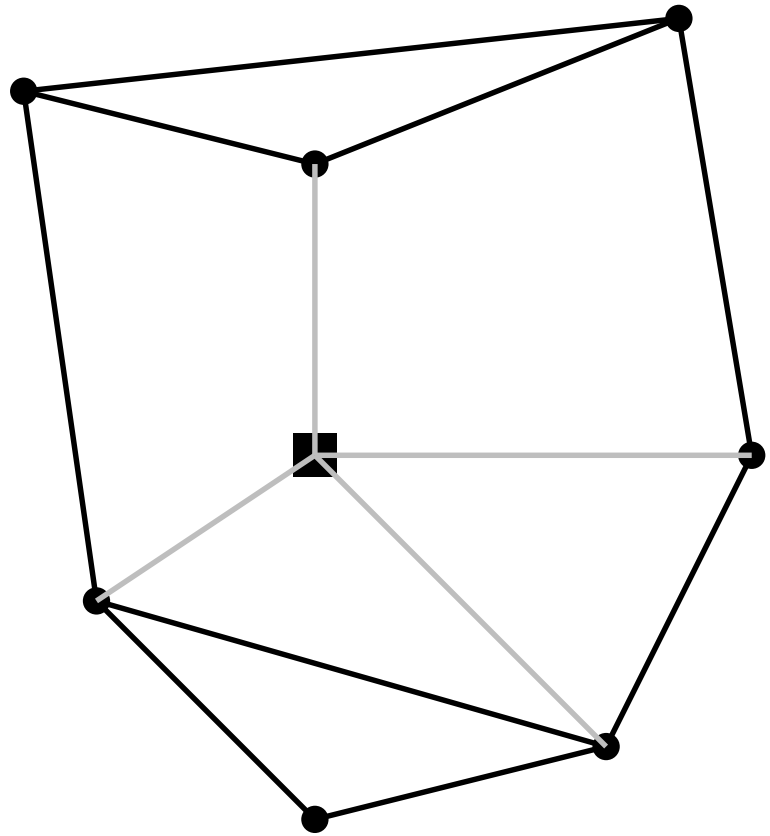


The beeping model

Introduced in [Cornejo, Kuhn 10]

Messages = beeps (no information)

Node distinguishes 0 and ≥ 1 beeps

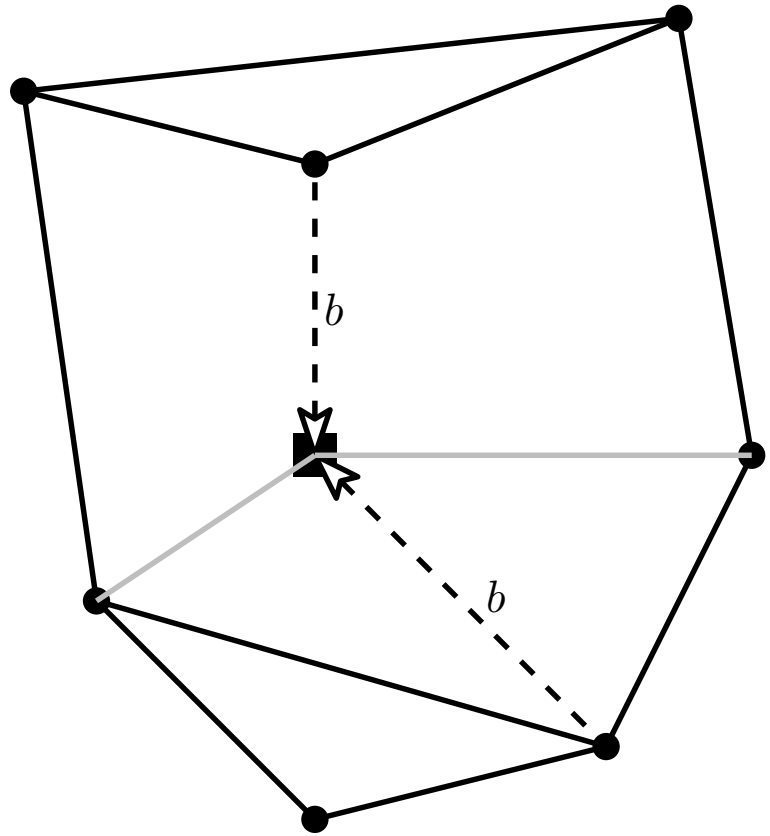


The beeping model

Introduced in [Cornejo, Kuhn 10]

Messages = beeps (no information)

Node distinguishes 0 and ≥ 1 beeps

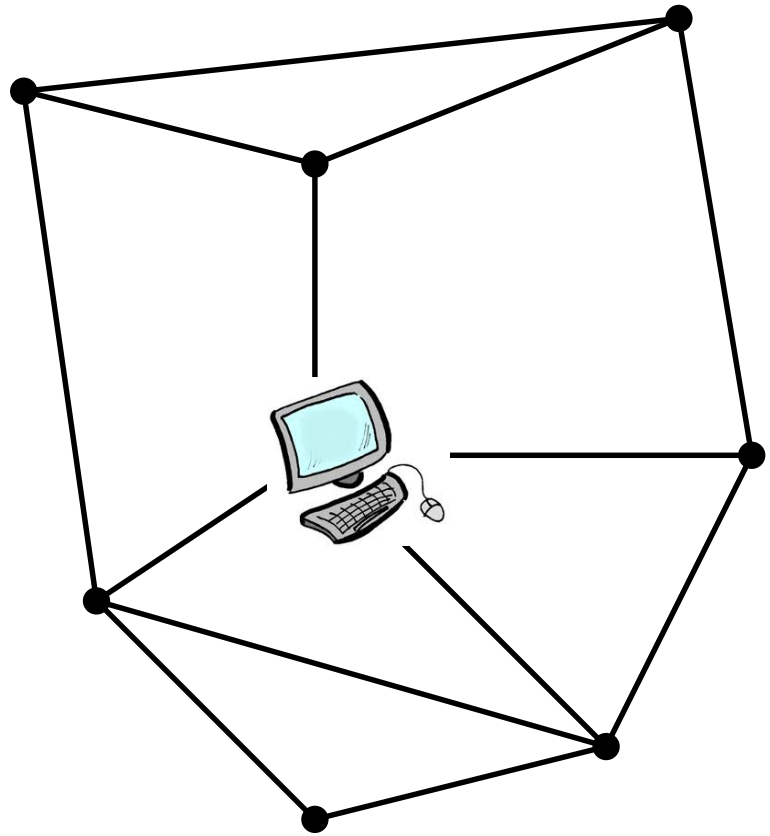


The beeping model

Introduced in [Cornejo, Kuhn 10]

Messages = beeps (no information)

Node distinguishes 0 and ≥ 1 beeps



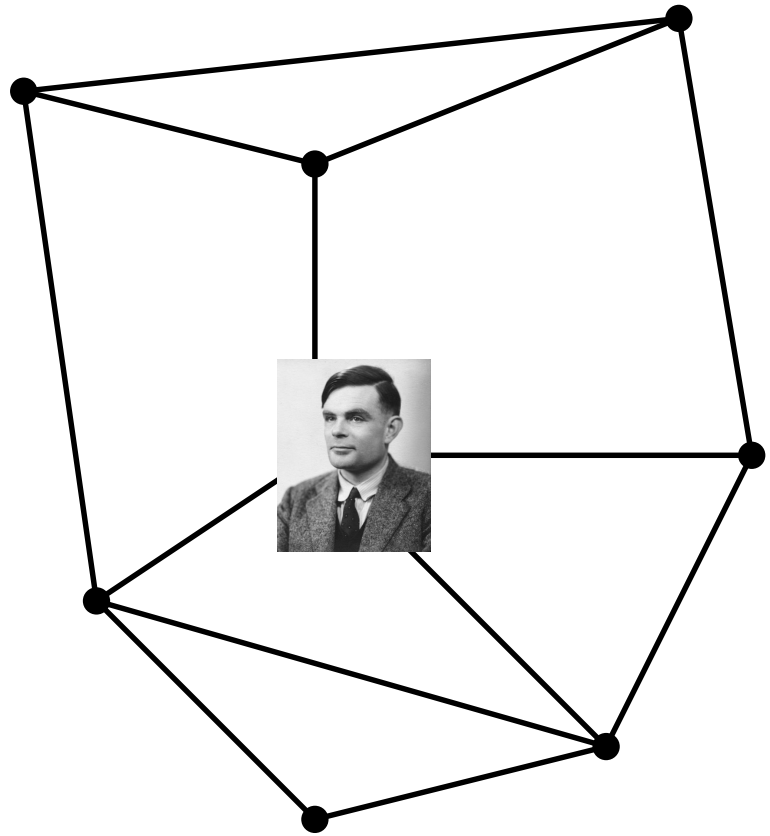
Local computation too strong

The beeping model

Introduced in [Cornejo, Kuhn 10]

Messages = beeps (no information)

Node distinguishes 0 and ≥ 1 beeps



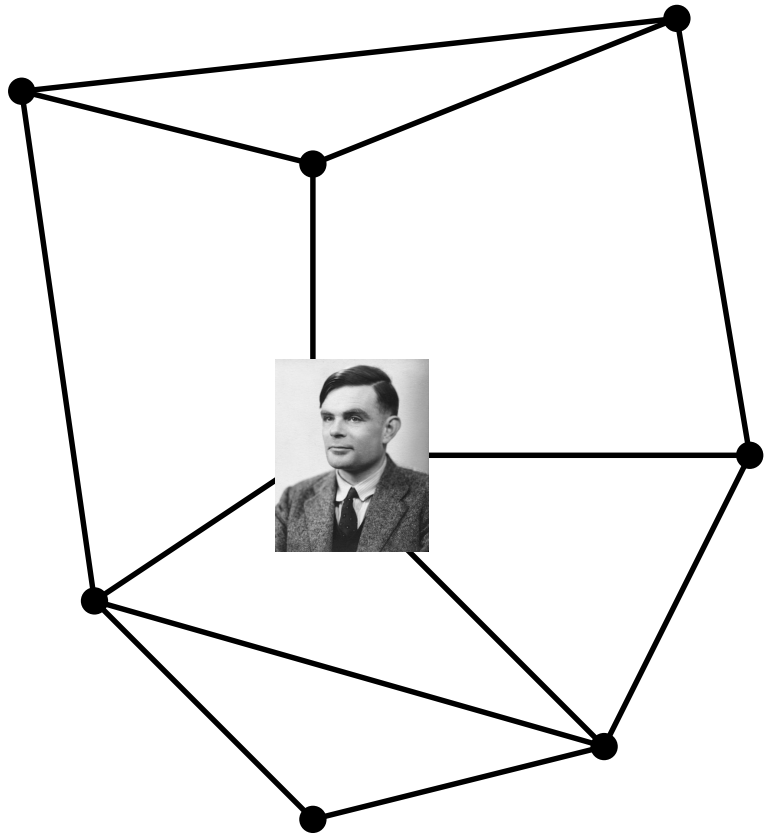
Local computation too strong

The beeping model

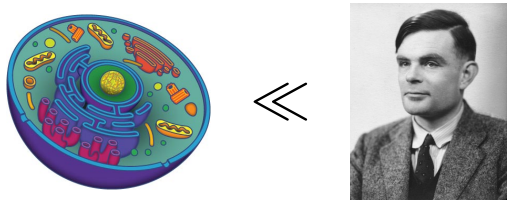
Introduced in [Cornejo, Kuhn 10]

Messages = beeps (no information)

Node distinguishes 0 and ≥ 1 beeps



Local computation too strong



Finite state machines (a.k.a. automata)

Finite state machines (a.k.a. automata)

- A fixed collection of states

Finite state machines (a.k.a. automata)

- A fixed collection of states
- A fixed collection of input signals (a.k.a. alphabet)

Finite state machines (a.k.a. automata)

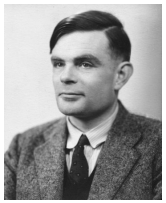
- A fixed collection of states
- A fixed collection of input signals (a.k.a. alphabet)
- $\text{state}(t+1) \longleftarrow \text{state}(t), \text{signal}(t)$

Finite state machines (a.k.a. automata)

- A fixed collection of states
- A fixed collection of input signals (a.k.a. alphabet)
- $\text{state}(t+1) \longleftarrow \text{state}(t), \text{signal}(t)$
  - determined by transition function

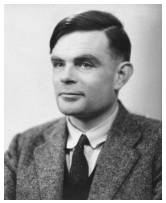
Finite state machines (a.k.a. automata)

- A fixed collection of states
- A fixed collection of input signals (a.k.a. alphabet)
- $\text{state}(t+1) \longleftarrow \text{state}(t), \text{signal}(t)$
  - determined by transition function
- Computational power $\ll$



Finite state machines (a.k.a. automata)

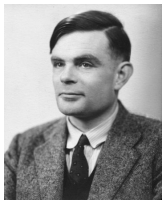
- A fixed collection of states
- A fixed collection of input signals (a.k.a. alphabet)
- state$(t+1) \longleftarrow$ state(t), signal(t)
  - determined by transition function
- Computational power $\ll$



Cell enzymes "programmed" to implement an FSM
[Benenson, Paz-Elizur, Adar, Keinan, Livneh, Shapiro 01]

Finite state machines (a.k.a. automata)

- A fixed collection of states
- A fixed collection of input signals (a.k.a. alphabet)
- state$(t+1) \longleftarrow$ state(t), signal(t)
  - determined by transition function
- Computational power $\ll$



Cell enzymes "programmed" to implement an FSM
[Benenson, Paz-Elizur, Adar, Keinan, Livneh, Shapiro 01]

Perhaps we should aim for a network of FSMs?

Cellular automata

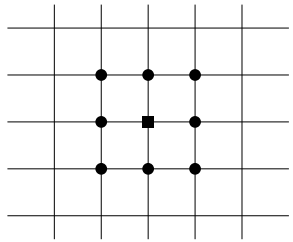
Cellular automata

Infinite grid of FSMs

Cellular automata

Infinite grid of FSMs

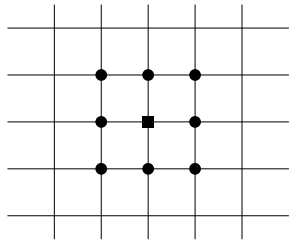
$q_{x,y}(t+1) \longleftarrow q_{x,y}(t), \{q_{x',y'}(t) : \text{grid neighbors } (x', y')\}$



Cellular automata

Infinite grid of FSMs

$q_{x,y}(t+1) \longleftarrow q_{x,y}(t), \{q_{x',y'}(t) : \text{grid neighbors } (x', y')\}$

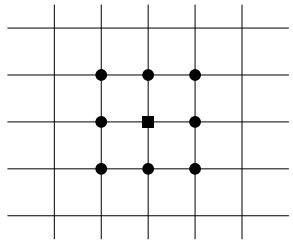


Typical question: How an initial (finite) configuration evolves?

Cellular automata

Infinite grid of FSMs

$q_{x,y}(t+1) \longleftarrow q_{x,y}(t), \{q_{x',y'}(t) : \text{grid neighbors } (x', y')\}$



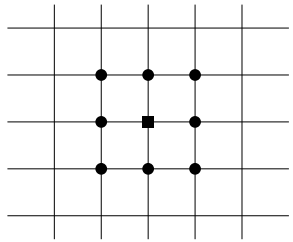
Typical question: How an initial (finite) configuration evolves?

Invented by   (crystal growth, self-replicating systems)

Cellular automata

Infinite grid of FSMs

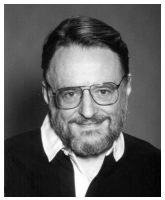
$q_{x,y}(t+1) \longleftarrow q_{x,y}(t), \{q_{x',y'}(t) : \text{grid neighbors } (x', y')\}$



Typical question: How an initial (finite) configuration evolves?

Invented by   (crystal growth, self-replicating systems)

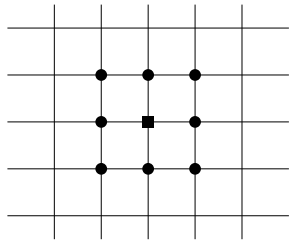
Game of life



Cellular automata

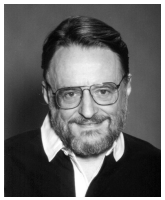
Infinite grid of FSMs

$q_{x,y}(t+1) \longleftarrow q_{x,y}(t), \{q_{x',y'}(t) : \text{grid neighbors } (x', y')\}$



Typical question: How an initial (finite) configuration evolves?

Invented by   (crystal growth, self-replicating systems)

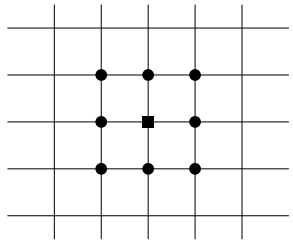
Game of life 

Digital physics  

Cellular automata

Infinite grid of FSMs

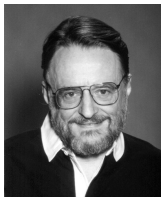
$q_{x,y}(t+1) \longleftarrow q_{x,y}(t), \{q_{x',y'}(t) : \text{grid neighbors } (x', y')\}$



Typical question: How an initial (finite) configuration evolves?

Invented by   (crystal growth, self-replicating systems)

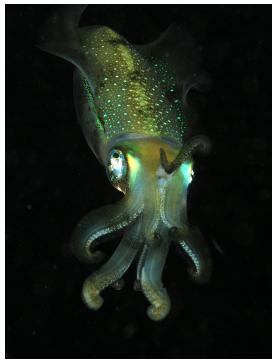
Game of life



Digital physics

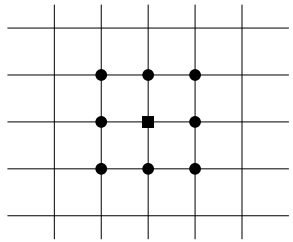
Biological processes

Cellular automata

Infinite grid of FSMs

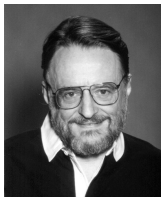
$q_{x,y}(t+1) \longleftarrow q_{x,y}(t), \{q_{x',y'}(t) : \text{grid neighbors } (x', y')\}$



Typical question: How an initial (finite) configuration evolves?

Invented by   (crystal growth, self-replicating systems)

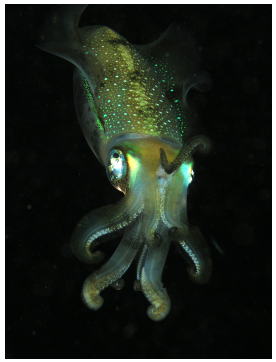
Game of life



Digital physics

Biological processes

Highly regular topology

1. Cells as computing devices
2. Abstract distributed computing models
3. Networked finite state machines
4. Results
   - MIS algorithm
5. Conclusions

nFSM

- Every node is a FSM

nFSM

- Every node is a FSM
- Communication based on transmissions: same message delivered to all neighbors

nFSM

- Every node is a FSM
- Communication based on transmissions:
  same message delivered to all neighbors
- Message is a letter in a constant-size communication alphabet Σ

nFSM

- Every node is a FSM
- Communication based on transmissions:
  same message delivered to all neighbors
- Message is a letter in a constant-size communication alphabet Σ
- Node u has a port corresponding to each $v \in N(u)$
  - Stores the last message $\sigma \in \Sigma$ delivered from v

nFSM

- Every node is a FSM
- Communication based on transmissions:
  same message delivered to all neighbors
- Message is a letter in a constant-size communication alphabet Σ
- Node u has a port corresponding to each $v \in N(u)$
  - Stores the last message $\sigma \in \Sigma$ delivered from v
- In each step, u decides on next state and which letter to transmit based on its current state and letters currently stored in its ports

nFSM

- Every node is a FSM
- Communication based on transmissions:
  same message delivered to all neighbors
- Message is a letter in a constant-size communication alphabet Σ
- Node u has a port corresponding to each $v \in N(u)$
  - Stores the last message $\sigma \in \Sigma$ delivered from v
- In each step, u decides on next state and which letter to transmit based on its current state and letters currently stored in its ports
- Problem:
  - # possible signals = # port configurations = $|\Sigma|^{\deg(u)}$

nFSM

- Every node is a FSM
- Communication based on transmissions:
  same message delivered to all neighbors
- Message is a letter in a constant-size communication alphabet Σ
- Node u has a port corresponding to each $v \in N(u)$
  - Stores the last message $\sigma \in \Sigma$ delivered from v
- In each step, u decides on next state and which letter to transmit based on its current state and letters currently stored in its ports
- Problem:
  - # possible signals = # port configurations = $|\Sigma|^{\deg(u)}$
  - Should be fixed in a FSM!

The one-two-many principle

- Node u cares only about the number π_σ of appearances of each $\sigma \in \Sigma$ in its ports (currently)

The one-two-many principle

- Node u cares only about the number π_σ of appearances of each $\sigma \in \Sigma$ in its ports (currently)
- # possible signals $= \binom{\deg(u)+|\Sigma|-1}{|\Sigma|-1} = \mathrm{poly}(\deg(u))$

The one-two-many principle

- Node u cares only about the number π_σ of appearances of each $\sigma \in \Sigma$ in its ports (currently)
- # possible signals $= \binom{\deg(u)+|\Sigma|-1}{|\Sigma|-1} = \mathrm{poly}(\deg(u))$
- π_σ calculated by the one-two-many principle:

The one-two-many principle

- Node u cares only about the number π_σ of appearances of each $\sigma \in \Sigma$ in its ports (currently)
- $\#$ possible signals $= \binom{\deg(u)+|\Sigma|-1}{|\Sigma|-1} = \mathrm{poly}(\deg(u))$
- π_σ calculated by the one-two-many principle:
  isolated cultures developed counting systems that don't go beyond 2

Warlpiri (Australia)

Piraha (the Amazon)

The one-two-many principle

- Node u cares only about the number π_σ of appearances of each $\sigma \in \Sigma$ in its ports (currently)
- $\#$ possible signals $= \binom{\deg(u)+|\Sigma|-1}{|\Sigma|-1} = \mathrm{poly}(\deg(u))$
- π_σ calculated by the one-two-many principle:
  isolated cultures developed counting systems that don't go beyond 2

Warlpiri (Australia)

Piraha (the Amazon)

- Constant bounding parameter $b \in \mathbb{Z}_{>0}$ (property of the algorithm)

The one-two-many principle

- Node u cares only about the number π_σ of appearances of each $\sigma \in \Sigma$ in its ports (currently)
- # possible signals $= \binom{\deg(u)+|\Sigma|-1}{|\Sigma|-1} = \mathrm{poly}(\deg(u))$
- π_σ calculated by the one-two-many principle:
  isolated cultures developed counting systems that don't go beyond 2

Warlpiri (Australia)

Piraha (the Amazon)

  - Constant bounding parameter $b \in \mathbb{Z}_{>0}$ (property of the algorithm)
  - u can distinguish between $\pi_\sigma = 0, 1, \ldots, b-1$, or $\pi_\sigma \geq b$

The one-two-many principle

- Node u cares only about the number π_σ of appearances of each $\sigma \in \Sigma$ in its ports (currently)
- # possible signals $= \binom{\deg(u)+|\Sigma|-1}{|\Sigma|-1} = \mathrm{poly}(\deg(u))$
- π_σ calculated by the one-two-many principle:
  isolated cultures developed counting systems that don't go beyond 2

Warlpiri (Australia)

Piraha (the Amazon)

  - Constant bounding parameter $b \in \mathbb{Z}_{>0}$ (property of the algorithm)
  - u can distinguish between $\pi_\sigma = 0, 1, \ldots, b-1$, or $\pi_\sigma \geq b$
  - # possible signals $= (b+1)^{|\Sigma|}$

The one-two-many principle

- Node u cares only about the number π_σ of appearances of each $\sigma \in \Sigma$ in its ports (currently)
- # possible signals $= \binom{\deg(u)+|\Sigma|-1}{|\Sigma|-1} = \mathrm{poly}(\deg(u))$
- π_σ calculated by the one-two-many principle:
  isolated cultures developed counting systems that don't go beyond 2

Warlpiri (Australia)

Piraha (the Amazon)

  - Constant bounding parameter $b \in \mathbb{Z}_{>0}$ (property of the algorithm)
  - u can distinguish between $\pi_\sigma = 0, 1, \ldots, b-1$, or $\pi_\sigma \geq b$
  - # possible signals $= (b+1)^{|\Sigma|}$
- FSM's transition function: $\delta : Q \times \{0, 1, \ldots, b\}^\Sigma \to Q \times \Sigma$

The one-two-many principle

- Node u cares only about the number π_σ of appearances of each $\sigma \in \Sigma$ in its ports (currently)
- # possible signals $= \binom{\deg(u)+|\Sigma|-1}{|\Sigma|-1} = \mathrm{poly}(\deg(u))$
- π_σ calculated by the one-two-many principle:
  isolated cultures developed counting systems that don't go beyond 2

Warlpiri (Australia)

Piraha (the Amazon)

  - Constant bounding parameter $b \in \mathbb{Z}_{>0}$ (property of the algorithm)
  - u can distinguish between $\pi_\sigma = 0, 1, \ldots, b-1$, or $\pi_\sigma \geq b$
  - # possible signals $= (b+1)^{|\Sigma|}$
- FSM's transition function: $\delta : Q \times \{0, 1, \ldots, b\}^\Sigma \to 2^{Q \times \Sigma}$

Crux of the model

Crux of the model

- Applicable to arbitrary network topologies

Crux of the model

- Applicable to arbitrary network topologies
- Nodes run the same (randomized) protocol

Crux of the model

- Applicable to arbitrary network topologies
- Nodes run the same (randomized) protocol
- All parameters of the protocol are constants, independent of any feature of the input graph (including $\deg(u)$):

Crux of the model

- Applicable to arbitrary network topologies
- Nodes run the same (randomized) protocol
- All parameters of the protocol are constants, independent of any feature of the input graph (including $\deg(u)$):
  - number of states

Crux of the model

- Applicable to arbitrary network topologies
- Nodes run the same (randomized) protocol
- All parameters of the protocol are constants, independent of any feature of the input graph (including $\deg(u)$):
  - number of states
  - size of alphabet Σ

Crux of the model

- Applicable to arbitrary network topologies
- Nodes run the same (randomized) protocol
- All parameters of the protocol are constants, independent of any feature of the input graph (including $\deg(u)$):
  - number of states
  - size of alphabet Σ
  - bounding parameter b

Crux of the model

- Applicable to arbitrary network topologies
- Nodes run the same (randomized) protocol
- All parameters of the protocol are constants, independent of any feature of the input graph (including $\deg(u)$):
  - number of states
  - size of alphabet Σ
  - bounding parameter b
  - size of the description of $\delta : Q \times \{0, 1, \ldots, b\}^{|\Sigma|} \rightarrow 2^{Q \times \Sigma}$

Crux of the model

- Applicable to arbitrary network topologies
- Nodes run the same (randomized) protocol
- All parameters of the protocol are constants, independent of any feature of the input graph (including $\deg(u)$):
  - number of states
  - size of alphabet Σ
  - bounding parameter b
  - size of the description of $\delta : Q \times \{0, 1, \ldots, b\}^{|\Sigma|} \rightarrow 2^{Q \times \Sigma}$
- A genuine FSM!

Crux of the model

- Applicable to arbitrary network topologies
- Nodes run the same (randomized) protocol
- All parameters of the protocol are constants, independent of any feature of the input graph (including $\deg(u)$):
  - number of states
  - size of alphabet Σ
  - bounding parameter b
  - size of the description of $\delta : Q \times \{0, 1, \ldots, b\}^{|\Sigma|} \rightarrow 2^{Q \times \Sigma}$
- A genuine FSM!
- Fully asynchronous environment

Crux of the model

- Applicable to arbitrary network topologies
- Nodes run the same (randomized) protocol
- All parameters of the protocol are constants, independent of any feature of the input graph (including $\deg(u)$):
  - number of states
  - size of alphabet Σ
  - bounding parameter b
  - size of the description of $\delta : Q \times \{0, 1, \ldots, b\}^{|\Sigma|} \rightarrow 2^{Q \times \Sigma}$
- A genuine FSM!
- Fully asynchronous environment
- The biological angle:

Crux of the model

- Applicable to arbitrary network topologies
- Nodes run the same (randomized) protocol
- All parameters of the protocol are constants, independent of any feature of the input graph (including $\deg(u)$):
  - number of states
  - size of alphabet Σ
  - bounding parameter b
  - size of the description of $\delta : Q \times \{0, 1, \ldots, b\}^{|\Sigma|} \rightarrow 2^{Q \times \Sigma}$
- A genuine FSM!
- Fully asynchronous environment
- The biological angle:
  - one-two-many counting $=$
    discrete analogue for detecting different concentration levels

1. Cells as computing devices
2. Abstract distributed computing models
3. Networked finite state machines
4. Results
   - MIS algorithm
5. Conclusions

Performance measure

Run-time:

- # time units until all nodes terminate

Performance measure

Run-time:

- # time units until all nodes terminate
- Efficient algorithm $= \log^{O(1)} n$ run-time [Linial 92]

Performance measure

Run-time:

- # time units until all nodes terminate
- Efficient algorithm $= \log^{O(1)} n$ run-time [Linial 92]
- Las Vegas algorithms, irrevocable output

Performance measure

Run-time:

- # time units until all nodes terminate
- Efficient algorithm $= \log^{O(1)} n$ run-time [Linial 92]
- Las Vegas algorithms, irrevocable output
- Run-time bounds hold in expectation and w.h.p.

Efficient algorithms

Efficient algorithms

- Maximal Independent Set in arbitrary graphs
  - run-time $= O(\log^2 n)$

Efficient algorithms

- Maximal Independent Set in arbitrary graphs
  - run-time $= O(\log^2 n)$
- Maximal 2-hop Independent Set in arbitrary graphs
  - run-time $= O(\log^2 n)$

Efficient algorithms

- Maximal Independent Set in arbitrary graphs
  - run-time $= O(\log^2 n)$
- Maximal 2-hop Independent Set in arbitrary graphs
  - run-time $= O(\log^2 n)$
- Coloring bounded degree graphs with $\Delta + 1$ colors
  - run-time $= O(\log n)$

Efficient algorithms

- Maximal Independent Set in arbitrary graphs
  - run-time $= O(\log^2 n)$
- Maximal 2-hop Independent Set in arbitrary graphs
  - run-time $= O(\log^2 n)$
- Coloring bounded degree graphs with $\Delta + 1$ colors
  - run-time $= O(\log n)$
- 2-hop Coloring bounded degree graphs with $\Delta^2 + 1$ colors
  - run-time $= O(\log n)$

Efficient algorithms

- Maximal Independent Set in arbitrary graphs
  - run-time $= O(\log^2 n)$
- Maximal 2-hop Independent Set in arbitrary graphs
  - run-time $= O(\log^2 n)$
- Coloring bounded degree graphs with $\Delta + 1$ colors
  - run-time $= O(\log n)$
- 2-hop Coloring bounded degree graphs with $\Delta^2 + 1$ colors
  - run-time $= O(\log n)$
- Coloring arbitrary trees with 3 colors
  - run-time $= O(\log n)$

Efficient algorithms

- Maximal Independent Set in arbitrary graphs
  - run-time $= O(\log^2 n)$
- Maximal 2-hop Independent Set in arbitrary graphs
  - run-time $= O(\log^2 n)$
- Coloring bounded degree graphs with $\Delta + 1$ colors
  - run-time $= O(\log n)$
- 2-hop Coloring bounded degree graphs with $\Delta^2 + 1$ colors
  - run-time $= O(\log n)$
- Coloring arbitrary trees with 3 colors
  - run-time $= O(\log n)$
- Maximal Matching in arbitrary graphs (small model modification)
  - run-time $= O(\log^2 n)$

Additional general results

Theorem (Synchronizer)

Every nFSM algorithm designed to operate in a synchronous environment can be simulated in an asynchronous environment with a constant multiplicative run-time overhead.

Theorem (Computability)

In terms of their computational power, nFSM algorithms are (almost) equivalent to randomized linear-space Turing machines.

1 Cells as computing devices

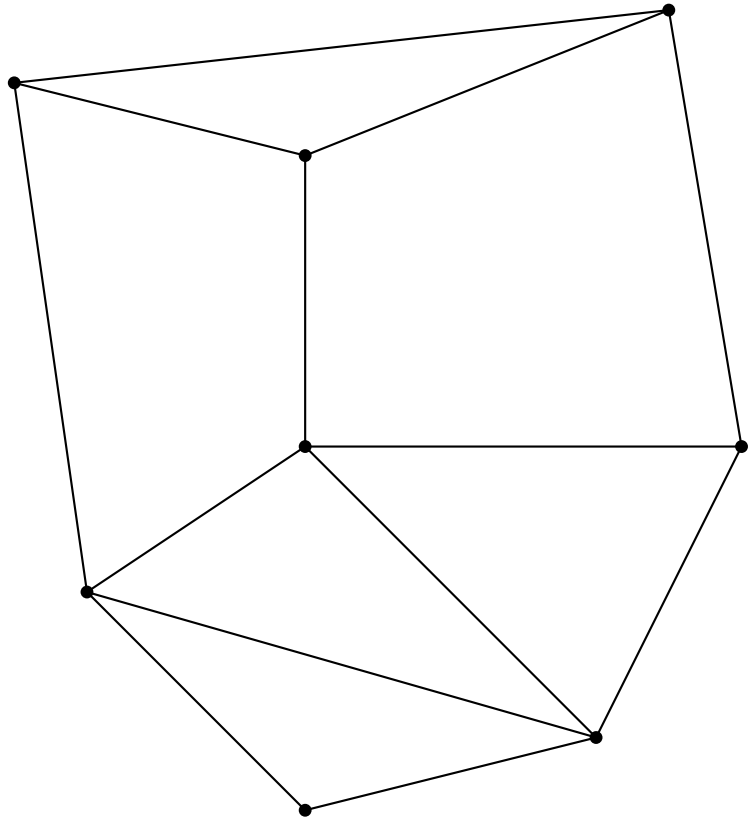
2 Abstract distributed computing models

3 Networked finite state machines

4 Results
- MIS algorithm

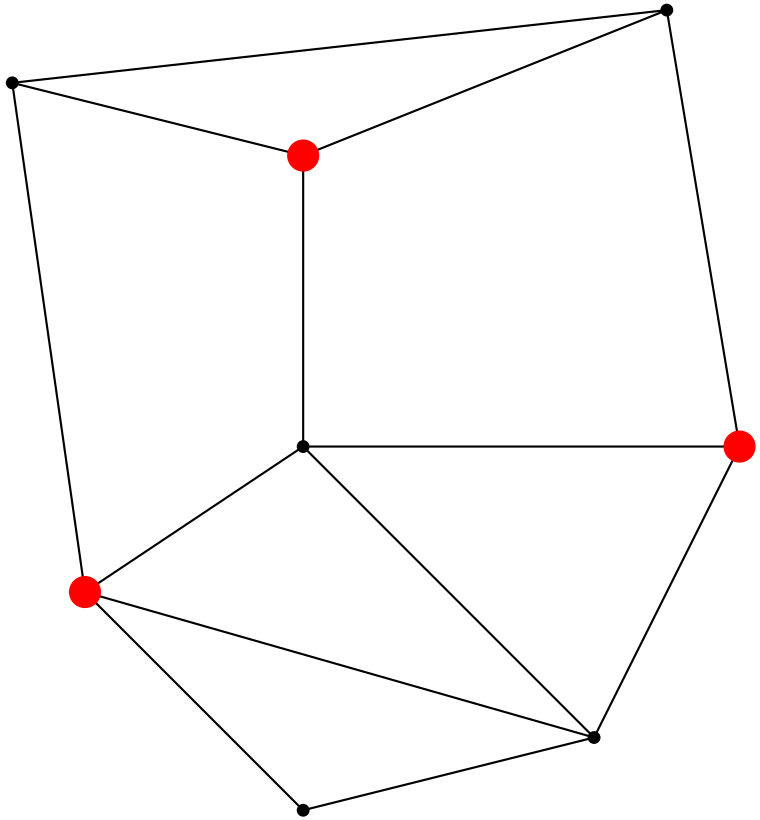
5 Conclusions

The MIS problem



The MIS problem

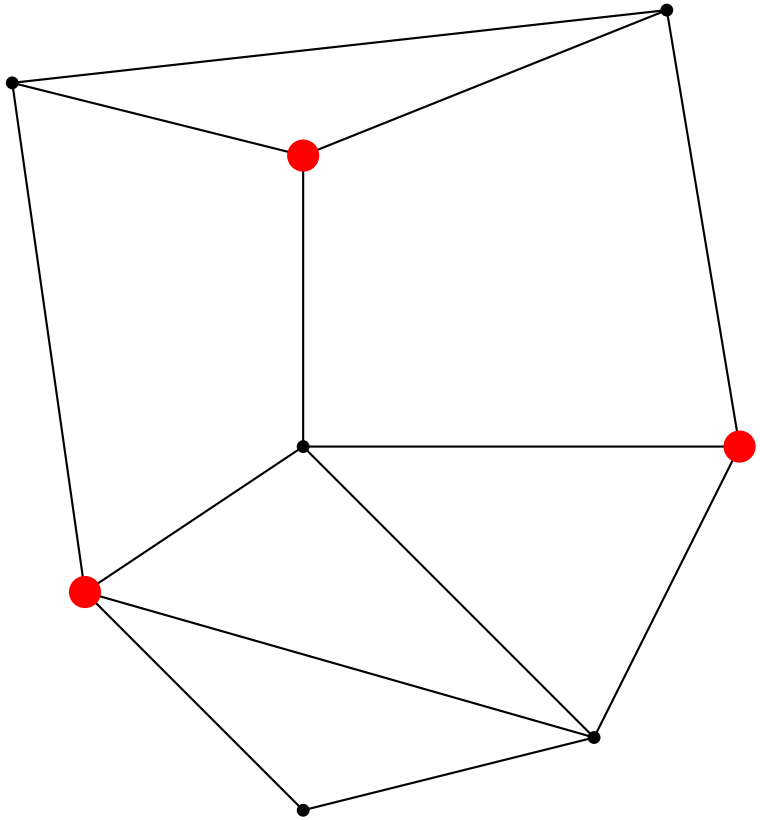
Independent set: set of nodes with no neighbors



The MIS problem

Independent set: set of nodes with no neighbors

maximal independent set (MIS): cannot be extended

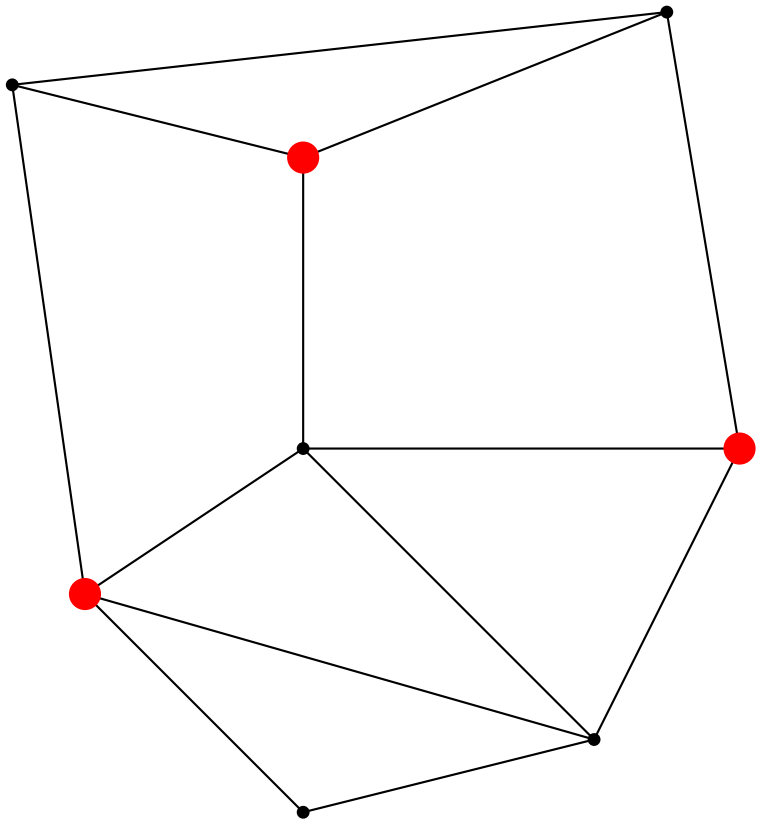


The MIS problem

Independent set: set of nodes with no neighbors

maximal independent set (MIS): cannot be extended

The MIS problem:
input: (arbitrary) network
output: MIS



MIS under nFSM — difficulties

MIS under nFSM — difficulties

- Existing MIS algorithms rely on grouping steps into phases: u competes with $N(u)$ over joining the MIS

MIS under nFSM — difficulties

- Existing MIS algorithms rely on grouping steps into phases: u competes with $N(u)$ over joining the MIS
- Require either
  - calculations with super-constant variables
  - independent communication with each neighbor
  - messages of logarithmic size

MIS under nFSM — difficulties

- Existing MIS algorithms rely on grouping steps into phases: u competes with $N(u)$ over joining the MIS
- Require either
  - calculations with super-constant variables
  - independent communication with each neighbor
  - messages of logarithmic size
- Idea: transmit $O(1)$ bits per step
  - logarithmically long phases

MIS under nFSM — difficulties

- Existing MIS algorithms rely on grouping steps into phases: u competes with $N(u)$ over joining the MIS
- Require either
  - calculations with super-constant variables
  - independent communication with each neighbor
  - messages of logarithmic size
- Idea: transmit $O(1)$ bits per step
  - logarithmically long phases
- Problem:
  - u must count the steps in a phase (deciding when it ends)

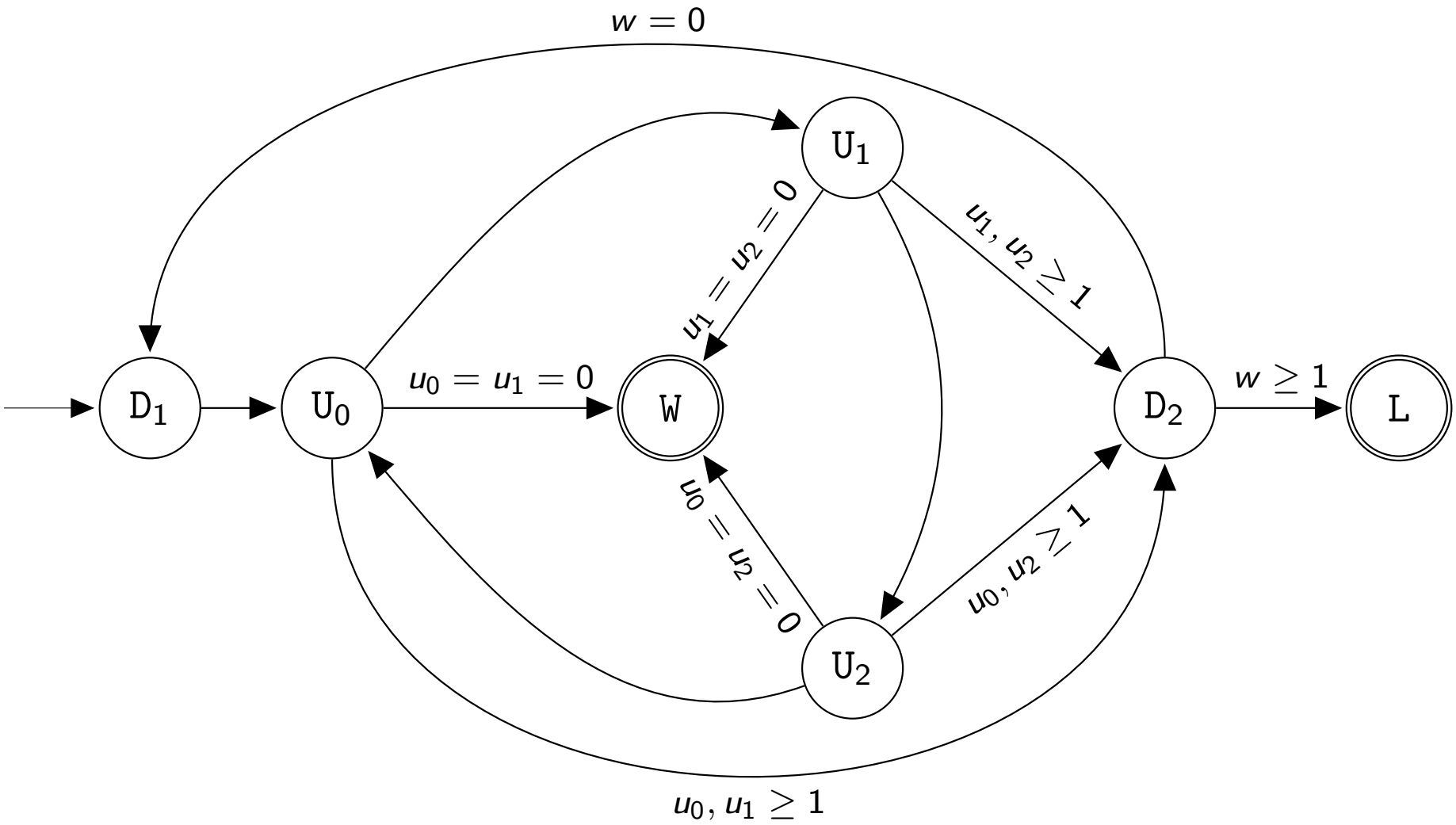
MIS under nFSM — difficulties

- Existing MIS algorithms rely on grouping steps into phases: u competes with $N(u)$ over joining the MIS
- Require either
  - calculations with super-constant variables
  - independent communication with each neighbor
  - messages of logarithmic size
- Idea: transmit $O(1)$ bits per step
  - logarithmically long phases
- Problem:
  - u must count the steps in a phase (deciding when it ends)
  - phases must be aligned to guarantee fair competition

MIS under nFSM — difficulties

- Existing MIS algorithms rely on grouping steps into phases: u competes with $N(u)$ over joining the MIS
- Require either
  - calculations with super-constant variables
  - independent communication with each neighbor
  - messages of logarithmic size
- Idea: transmit $O(1)$ bits per step
  - logarithmically long phases
- Problem:
  - u must count the steps in a phase (deciding when it ends)
  - phases must be aligned to guarantee fair competition
- How can we decide if u joins MIS without long aligned phases?

MIS under nFSM — solution



MIS under nFSM — solution

- Relax requirement that phase is aligned and of predetermined length

MIS under nFSM — solution

- Relax requirement that phase is aligned and of predetermined length
- Tournament:
  - length determined probabilistically
  - "softly" aligned
  - maintained under nFSM

MIS under nFSM — solution

- Relax requirement that phase is aligned and of predetermined length
- Tournament:
  - length determined probabilistically
  - "softly" aligned
  - maintained under nFSM
- Prove:
  1. Amortized length of a tournament is $O(\log n)$ w.h.p.

MIS under nFSM — solution

- Relax requirement that phase is aligned and of predetermined length
- Tournament:
  - length determined probabilistically
  - "softly" aligned
  - maintained under nFSM
- Prove:
  1. Amortized length of a tournament is $O(\log n)$ w.h.p.
  2. Guarantee fair competition $\Longrightarrow$
     const fraction of the edges is removed with const probability $\Longrightarrow$
     $O(\log n)$ tournaments w.h.p.

1. Cells as computing devices
2. Abstract distributed computing models
3. Networked finite state machines
4. Results
   - MIS algorithm
5. Conclusions

Summary

- Abstract model for network of FSMs

Summary

- Abstract model for network of FSMs
- Fundamental DC problems admit efficient algorithms

Summary

- Abstract model for network of FSMs
- Fundamental DC problems admit efficient algorithms
- Suitable to biological cellular networks
  - Local computation, communication, asynchrony

Summary

- Abstract model for network of FSMs
- Fundamental DC problems admit efficient algorithms
- Suitable to biological cellular networks
  - Local computation, communication, asynchrony
  - Also networks of man made nano-devices

Summary

- Abstract model for network of FSMs
- Fundamental DC problems admit efficient algorithms
- Suitable to biological cellular networks
  - Local computation, communication, asynchrony
  - Also networks of man made nano-devices
- Another model: population protocols (pairwise interactions, eventual correctness)

Summary

- Abstract model for network of FSMs
- Fundamental DC problems admit efficient algorithms
- Suitable to biological cellular networks
  - Local computation, communication, asynchrony
  - Also networks of man made nano-devices
- Another model: population protocols (pairwise interactions, eventual correctness)
- Reasonable constants (MIS: $|Q| = |\Sigma| = 7$, $b = 1$)

Summary

- Abstract model for network of FSMs
- Fundamental DC problems admit efficient algorithms
- Suitable to biological cellular networks
  - Local computation, communication, asynchrony
  - Also networks of man made nano-devices
- Another model: population protocols (pairwise interactions, eventual correctness)
- Reasonable constants (MIS: $|Q| = |\Sigma| = 7$, $b = 1$)
- Open problem: dynamic environment

Summary

- Abstract model for network of FSMs
- Fundamental DC problems admit efficient algorithms
- Suitable to biological cellular networks
  - Local computation, communication, asynchrony
  - Also networks of man made nano-devices
- Another model: population protocols (pairwise interactions, eventual correctness)
- Reasonable constants (MIS: $|Q| = |\Sigma| = 7$, $b = 1$)
- Open problem: dynamic environment
- Joint research project with Jara Uitto and Roger Wattenhofer

Summary

- Abstract model for network of FSMs
- Fundamental DC problems admit efficient algorithms
- Suitable to biological cellular networks
  - Local computation, communication, asynchrony
  - Also networks of man made nano-devices
- Another model: population protocols (pairwise interactions, eventual correctness)
- Reasonable constants (MIS: $|Q| = |\Sigma| = 7$, $b = 1$)
- Open problem: dynamic environment
- Joint research project with Jara Uitto and Roger Wattenhofer

תודה רבה