Time and space in distributed computing

Tuomo Lempiäinen

Spring meeting of logicians

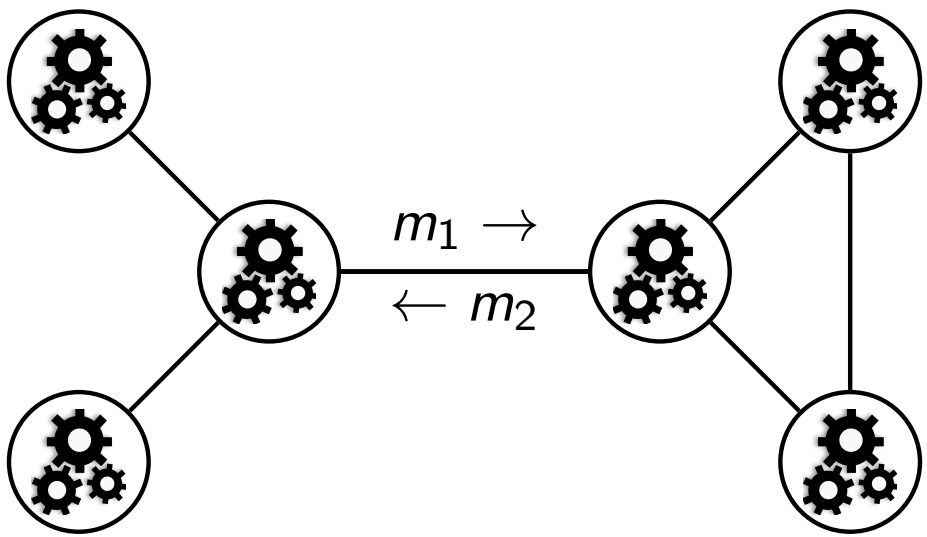
17 May 2019

Tampere

Outline

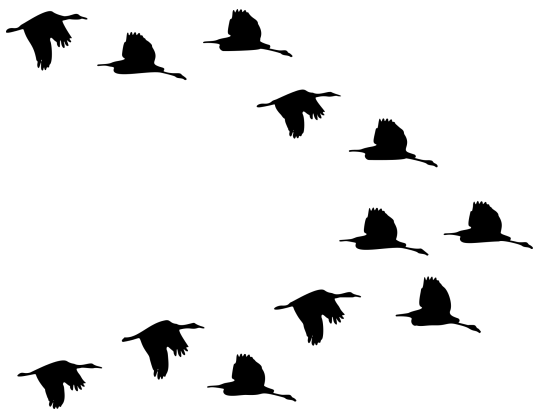
What is distributed computing?

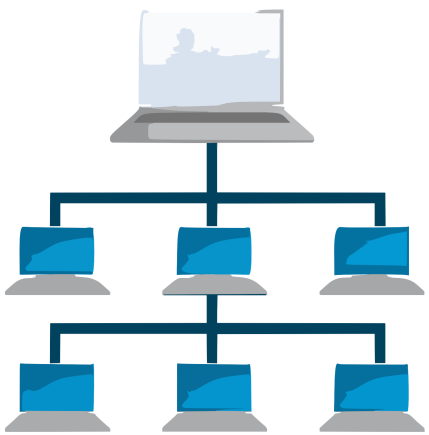
- In distributed computing, we study systems that consists of multiple *computational entities* or *nodes* that *communicate* with each other.

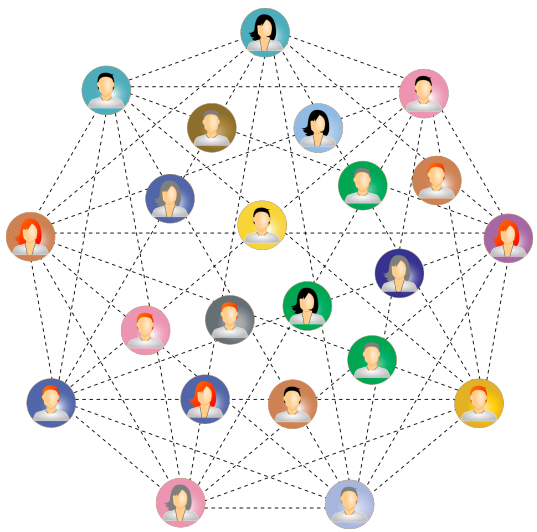


Distributed systems in the wild

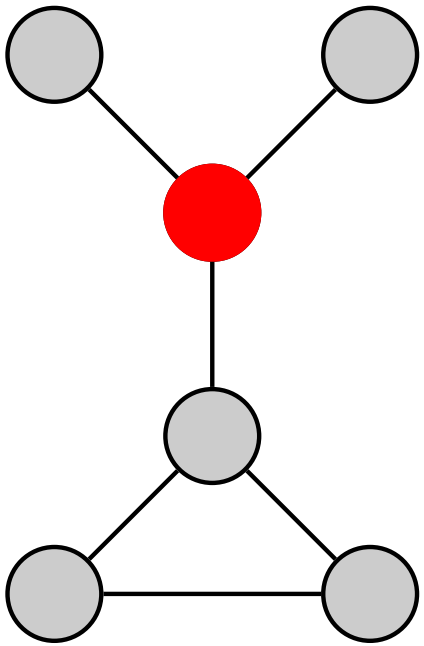
- Distributed computing is a very general concept.
- Similar principles apply to biological organisms, computer networks, human societies, ...





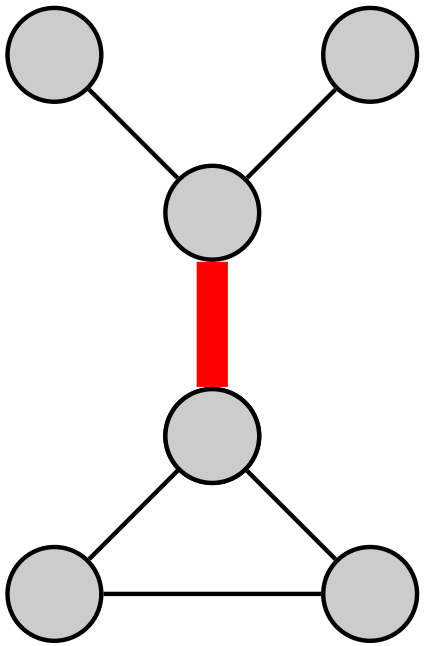


Message-passing models of distributed computing



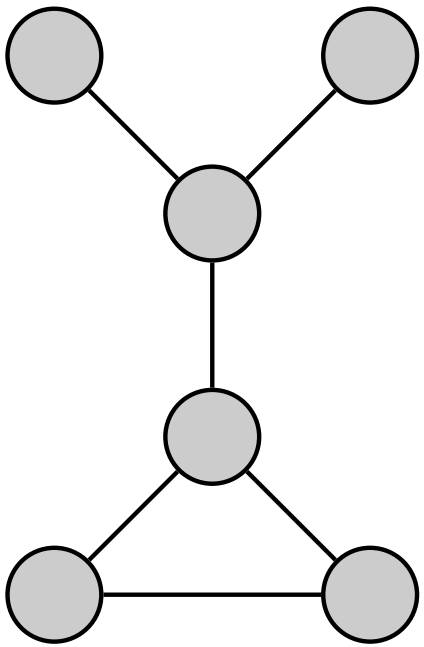
Nodes are computational units, edges are communication links.

Message-passing models of distributed computing



Nodes are computational units, edges are communication links.

Message-passing models of distributed computing

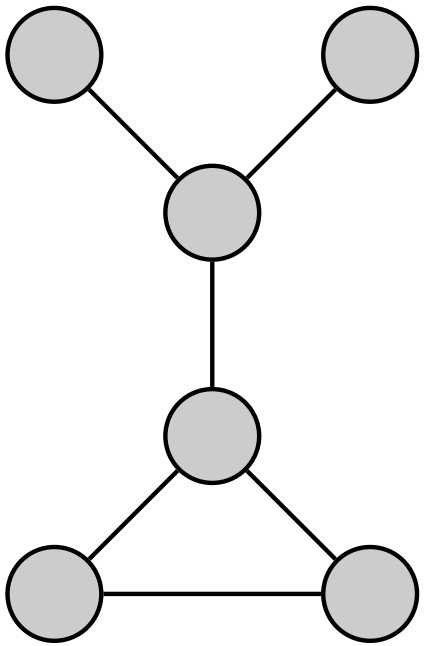


Nodes are computational units, edges are communication links.

Each node

- runs an identical algorithm,
- communicates with its neighbouring nodes,
- halts and produces its own local output.

Message-passing models of distributed computing

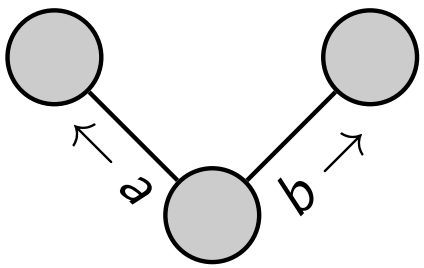


Nodes are computational units, edges are communication links.

Numerous models that differ in the details, e.g. how nodes can distinguish

- themselves from others,
- received messages.

Communication between nodes

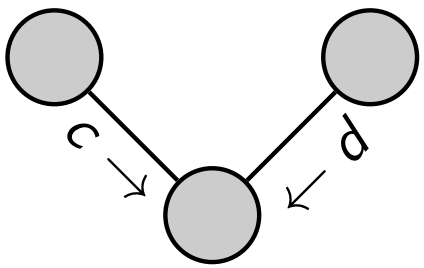


Synchronous communication rounds.

On each round, every node

1. sends messages to its neighbours,
2. receives messages from its neighbours,
3. updates its local state.

Communication between nodes

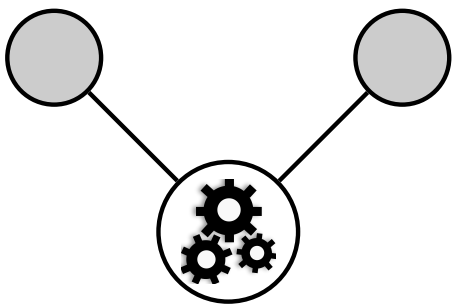


Synchronous communication rounds.

On each round, every node

1. sends messages to its neighbours,
2. receives messages from its neighbours,
3. updates its local state.

Communication between nodes

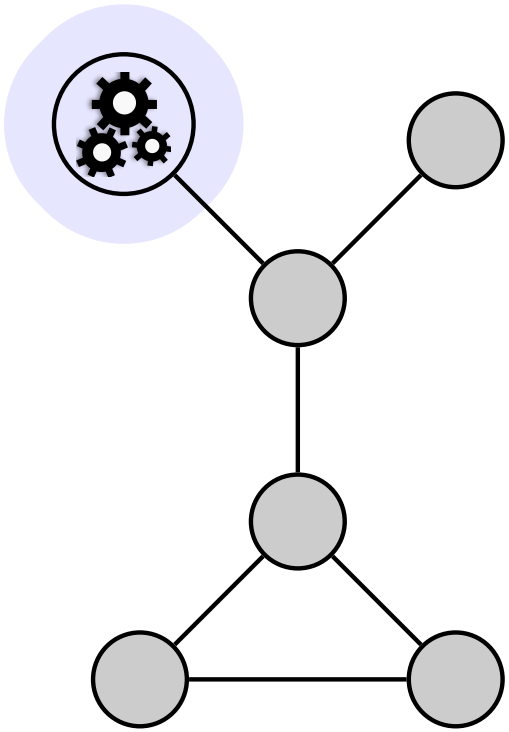


Synchronous communication rounds.

On each round, every node

1. sends messages to its neighbours,
2. receives messages from its neighbours,
3. updates its local state.

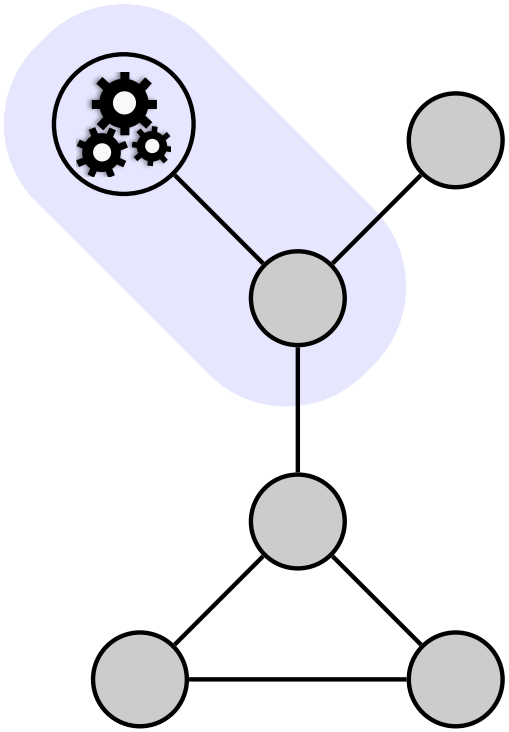
Communication between nodes



Round 1.

- Initially, nodes are only aware of themselves.
- By exchanging messages, nodes gain more information on the network structure.
- Number of rounds = distance.

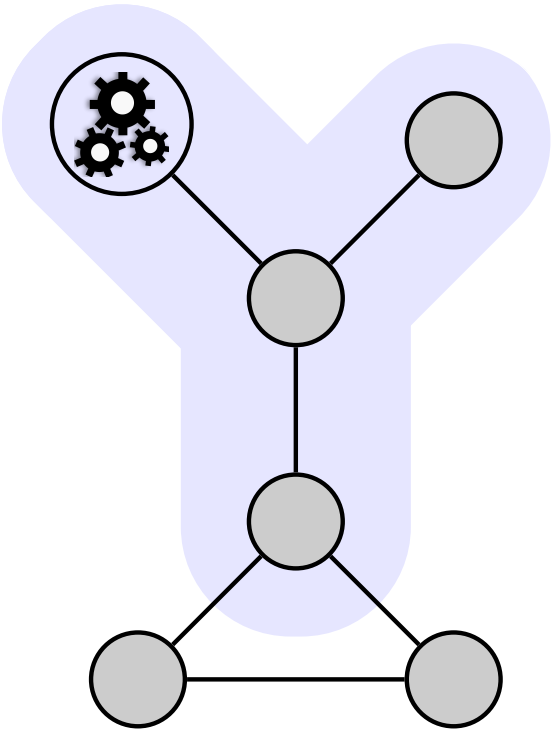
Communication between nodes



Round 2.

- Initially, nodes are only aware of themselves.
- By exchanging messages, nodes gain more information on the network structure.
- Number of rounds = distance.

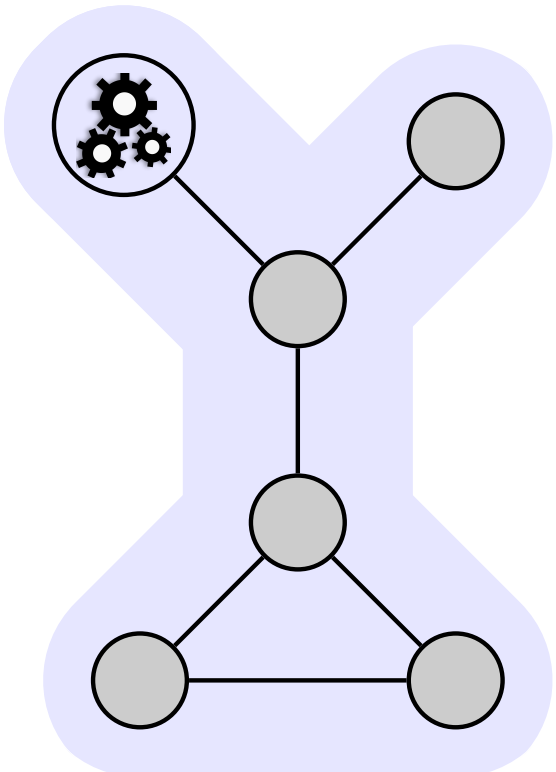
Communication between nodes



Round 3.

- Initially, nodes are only aware of themselves.
- By exchanging messages, nodes gain more information on the network structure.
- Number of rounds = distance.

Communication between nodes



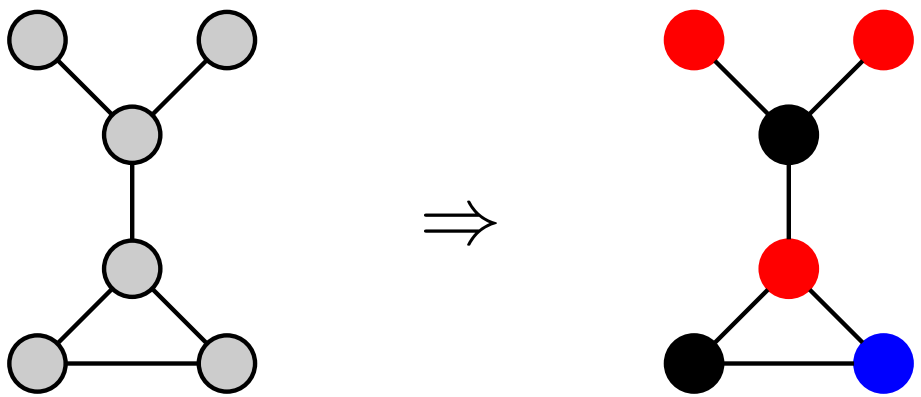
Round 4.

- Initially, nodes are only aware of themselves.
- By exchanging messages, nodes gain more information on the network structure.
- Number of rounds = distance.

Graph problems

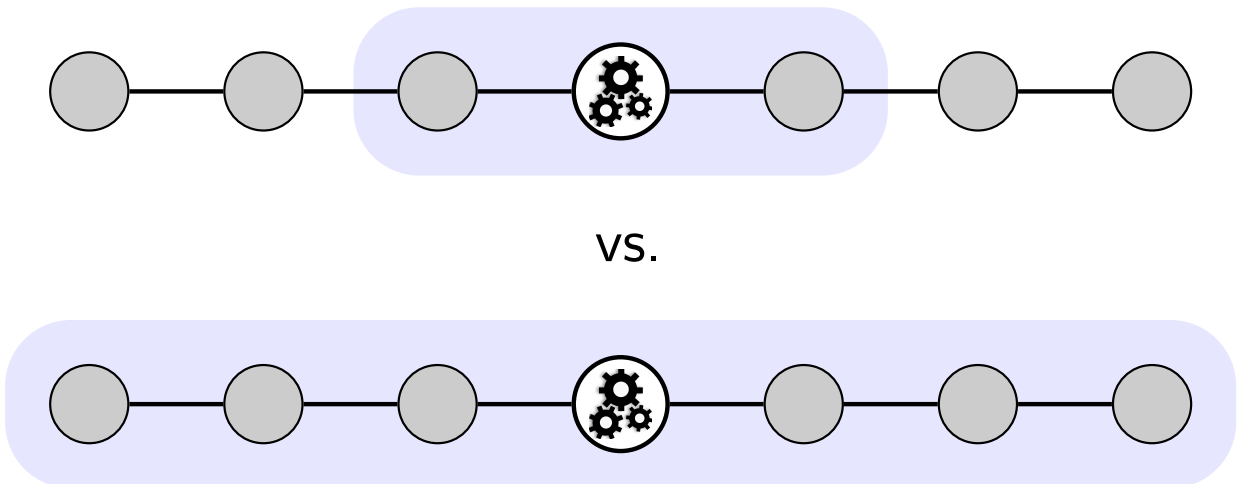
- Problem instance = communication graph (+ local input labelling).

- Example: proper node 3-colouring.
- Each node has to halt and output its own colour.



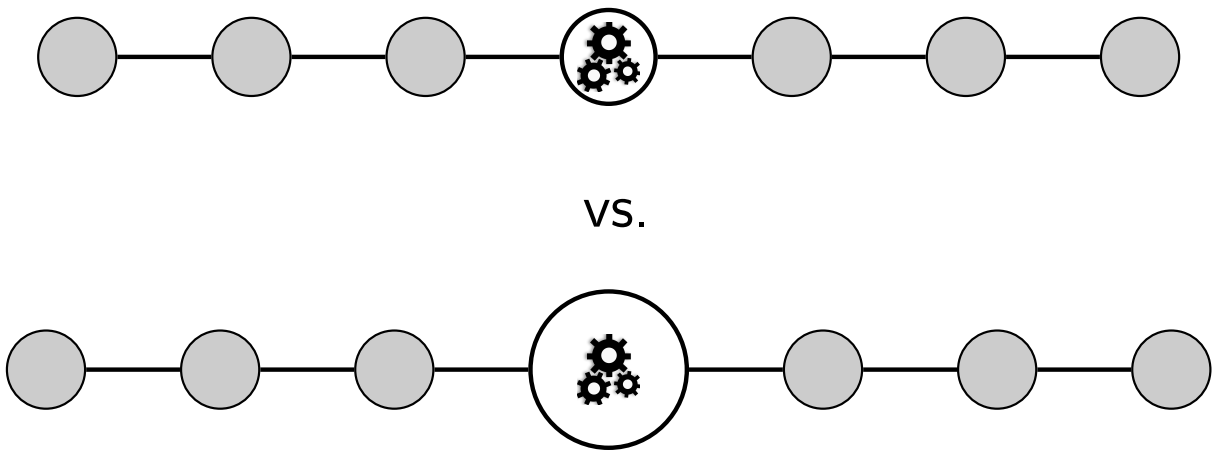
Computational complexity

- *Time complexity*: the number of communication rounds until all nodes have halted.



Computational complexity

- *Space complexity*: the number of bits needed to encode all the states that are visited at least once.



Computational complexity

- *Time complexity*: the number of communication rounds until all nodes have halted.
- *Space complexity*: the number of bits needed to encode all the states that are visited at least once.
- . . . as a function of n, over all graphs of n nodes.

Research on distributed time complexity

- The most-studied setting:
  - LOCAL model of computing: nodes have unique identifiers.
  - Locally-checkable labelling (LCL) problems: solutions can be verified in constant time.

Research on distributed time complexity

- The most-studied setting:
  - LOCAL model of computing: nodes have unique identifiers.
  - Locally-checkable labelling (LCL) problems: solutions can be verified in constant time.
- Cycle and path graphs:
  - Initiated by Naor and Stockmeyer (1995).
  - Chang, Kopelowitz, Pettie (2016): only $O(1)$, $\Theta(\log^* n)$ and $\Theta(n)$ are possible.
- General bounded-degree graphs:
  - Lots of progress recently.
  - Complexities between $\omega(\log^* n)$ and $o(n)$.
  - A gap between $\omega(\log^* n)$ and $o(\log n)$.
  - Complexity $\Theta(n^{1/k})$ for all k.
  - ...

What about space complexity?

- A well-established topic in centralised complexity theory.
- For example, NP $\subseteq$ PSPACE $\subseteq$ EXP.

- In the distributed setting, constant-space computation has been studied (e.g. cellular automata)
- . . . but the relationship between space and time is mostly an unexplored area.

Case study: constant space and non-constant time

Lempiäinen & Suomela

Constant space and non-constant time in distributed computing

Proc. 21st International Conference on Principles of Distributed Systems (OPODIS 2017), Lisbon, Portugal

Time vs. space in the distributed setting

- A message-passing model.

- Constant time complexity $\Rightarrow$ constant space complexity.
- Does the converse hold?

Time vs. space in the distributed setting

- A message-passing model.

- Constant time complexity $\Rightarrow$ constant space complexity.
- Does the converse hold?

- More specifically: does there exist a distributed graph problem that is
  - solvable in constant space,
  - not solvable in constant time?

Time vs. space in the distributed setting

- A message-passing model.

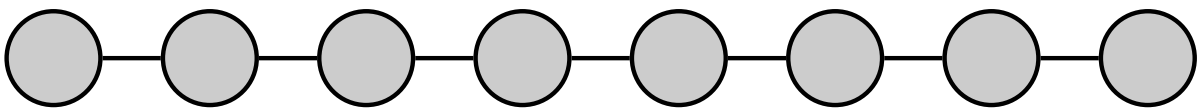
- Constant time complexity $\Rightarrow$ constant space complexity.
- Does the converse hold?
- More specifically: does there exist a distributed graph problem that is
  - solvable in constant space,
  - not solvable in constant time?

- Our result: YES, constant space and constant time can be separated!

What are the right assumptions? (1/2)

- Easy to construct constant-space non-constant-time problems if
  - promise that the graph is a path, or
  - nodes do not need to halt.

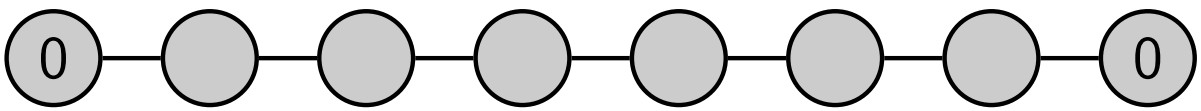
- Count the distance modulo 2 to the nearest degree-1 node:



What are the right assumptions? (1/2)

- Easy to construct constant-space non-constant-time problems if
  - promise that the graph is a path, or
  - nodes do not need to halt.

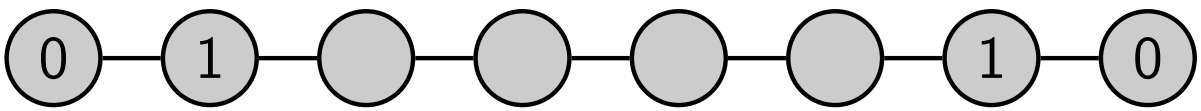
- Count the distance modulo 2 to the nearest degree-1 node:



What are the right assumptions? (1/2)

- Easy to construct constant-space non-constant-time problems if
  - promise that the graph is a path, or
  - nodes do not need to halt.

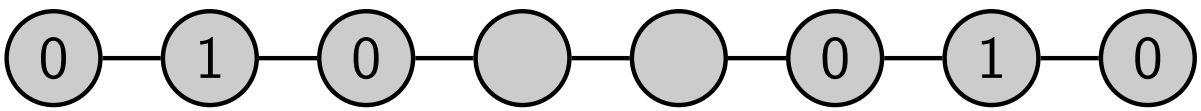
- Count the distance modulo 2 to the nearest degree-1 node:



What are the right assumptions? (1/2)

- Easy to construct constant-space non-constant-time problems if
  - promise that the graph is a path, or
  - nodes do not need to halt.

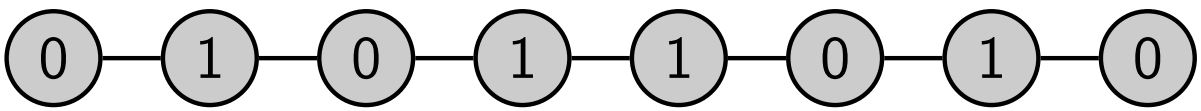
- Count the distance modulo 2 to the nearest degree-1 node:



What are the right assumptions? (1/2)

- Easy to construct constant-space non-constant-time problems if
  - promise that the graph is a path, or
  - nodes do not need to halt.

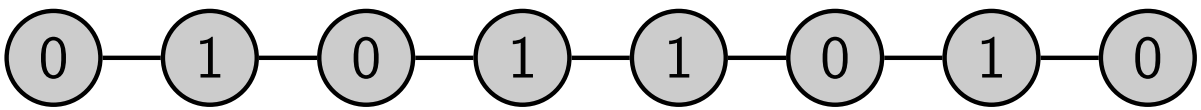
- Count the distance modulo 2 to the nearest degree-1 node:



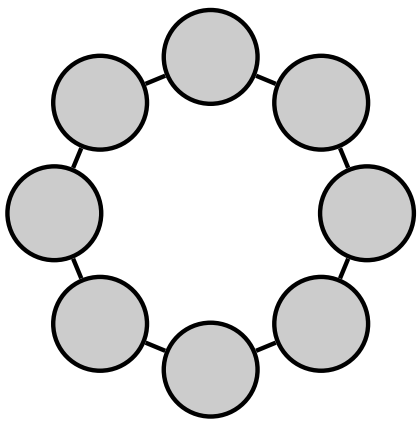
What are the right assumptions? (1/2)

- Easy to construct constant-space non-constant-time problems if
  - promise that the graph is a path, or
  - nodes do not need to halt.

- Count the distance modulo 2 to the nearest degree-1 node:



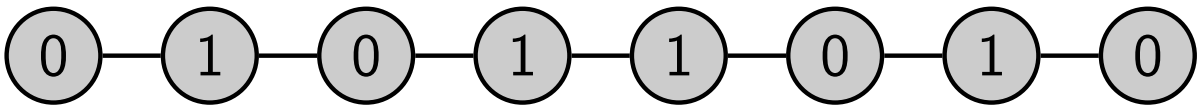
- But what if the input is a cycle?



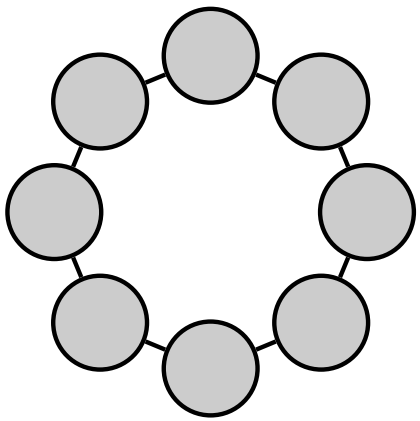
What are the right assumptions? (1/2)

- Easy to construct constant-space non-constant-time problems if
  - promise that the graph is a path, or
  - nodes do not need to halt.

- Count the distance modulo 2 to the nearest degree-1 node:



- But what if the input is a cycle?



- Our result does not require any promises about the input.

What are the right assumptions? (2/2)

- To achieve a strong separation result, we want a graph problem Π that
  - is solvable in constant space in a very weak model of computation,
  - cannot be solved in constant time even in a very strong model.

- Hence, we will present an algorithm for Π in a very weak model of computation:
  - no unique IDs,
  - no randomness,
  - only constant-size local inputs,
  - only weak communication capabilities.

Model of computation

Let $G = (V, E)$ be a graph. An *input* for G is a function $f\colon V \to I$, where I is a finite set.

A *distributed state machine* is a tuple $\mathcal{A} = (S, H, \sigma_0, M, \mu, \sigma)$, where

- S is a set of states,
- $H \subseteq S$ is a finite set of halting states,
- $\sigma_0\colon \mathbb{N} \times I \to S$ is an initialisation function,
- M is a set of possible messages,
- $\mu\colon S \to M$ is a function that constructs the outgoing messages,
- $\sigma\colon S \times \mathcal{P}(M) \to S$ is a function that defines the state transitions, so that $\sigma(h, \mathcal{M}) = h$ for each $h \in H$ and $\mathcal{M} \in \mathcal{P}(M)$.

Model of computation

The *execution* of $\mathcal{A}$ on (G, f):

- The state of the system in round $r \in \mathbb{N}$ is $x_r \colon V \to S$.
- Set $x_0(v) = \sigma_o(\deg(v), f(v))$ for each $v \in V$.
- Let $A_{r+1}(v) = \{\mu(x_r(u)) : u \in N(v)\}$ denote the set of messages received by node v in round $r + 1$.
- The new state of each $v \in V$ is $x_{r+1}(v) = \sigma(x_r(v), A_{r+1}(v))$.

Complexity measures

- The *running time* of $\mathcal{A}$ on (G, f) is the smallest $t \in \mathbb{N}$ for which $x_t(v) \in H$ holds for all $v \in V$.
- The output of $\mathcal{A}$ on (G, f) is $x_t \colon V \to H$, where t is the running time.
- The *space usage* of $\mathcal{A}$ on (G, f) is

$$\left\lceil \log_2 \left| \{ x_r(v) \in S \,:\, r \in [0, t] \text{ and } v \in V \} \right| \right\rceil,$$

where $t \in \mathbb{N}$ is the running time of $\mathcal{A}$ on (G, f).

Complexity measures

- The *running time* of $\mathcal{A}$ on (G, f) is the smallest $t \in \mathbb{N}$ for which $x_t(v) \in H$ holds for all $v \in V$.
- The output of $\mathcal{A}$ on (G, f) is $x_t \colon V \to H$, where t is the running time.
- The *space usage* of $\mathcal{A}$ on (G, f) is

$$\left\lceil \log_2 \left| \{ x_r(v) \in S \,:\, r \in [0, t] \text{ and } v \in V \} \right| \right\rceil,$$

  where $t \in \mathbb{N}$ is the running time of $\mathcal{A}$ on (G, f).

- The constant-time version of this model is captured by the *basic modal logic* (Hella et al. 2012).

Our main result

Problem

Construct a graph problem Π *such that*

1. *there exists a constant-space algorithm* $\mathcal{A}$ *that halts and solves* Π *in all (finite, simple, and connected) graphs, and*
2. Π *is not solvable by any constant-time algorithm.*

Theorem

There does exist a decision graph problem Π *that satisfies the above requirements (1) and (2).*

Our main result

Problem

Construct a graph problem Π *such that*

1. *there exists a constant-space algorithm* $\mathcal{A}$ *that halts and solves* Π *in all (finite, simple, and connected) graphs, and*
2. Π *is not solvable by any constant-time algorithm.*

Theorem (Stronger result)

There does exist a decision graph problem Π *that satisfies the above requirements (1) and (2), and that is not solvable by any sublinear-time algorithm even in the class of graphs of maximum degree 2.*

An intriguing binary sequence

- The *Thue–Morse sequence* is the infinite sequence (over $\{0, 1\}$) whose prefixes T_i of length 2^i are defined as follows:
  - start with $T_0 = 0$,
  - obtain T_i from T_{i-1} by mapping $0 \mapsto 01$ and $1 \mapsto 10$.

- First steps:

$$T_0 = 0$$
$$T_1 = 01$$
$$T_2 = 0110$$
$$T_3 = 01101001$$
$$T_4 = 0110100110010110$$
$$\vdots$$

An intriguing binary sequence

- The *Thue–Morse sequence* is the infinite sequence (over $\{0, 1\}$) whose prefixes T_i of length 2^i are defined as follows:
  - start with $T_0 = 0$,
  - obtain T_i from T_{i-1} by mapping $0 \mapsto 01$ and $1 \mapsto 10$.

- First steps:

$$T_0 = 0$$
$$T_1 = 01$$
$$T_2 = 0110$$
$$T_3 = 01101001$$
$$T_4 = 0110100110010110$$
$$\vdots$$

- Interesting properties:
  - For each $i \in \mathbb{N}$, T_{2i} is a palindrome.
  - The sequence does not contain any cubes, i.e. subwords XXX for any $X \in \{0, 1\}^*$.

An intriguing binary sequence

- The *Thue–Morse sequence* is the infinite sequence (over $\{0, 1\}$) whose prefixes T_i of length 2^i are defined as follows:
  - start with $T_0 = 0$,
  - obtain T_i from T_{i-1} by mapping $0 \mapsto 01$ and $1 \mapsto 10$.

- First steps:

$$T_0 = 0$$
$$T_1 = 01$$
$$T_2 = 0110$$
$$T_3 = 01101001$$
$$T_4 = 0110100110010110$$
$$\vdots$$

- The sequence was used previously in distributed computing by Kuusisto (2014).

Towards a decision graph problem

- Could we separate paths labelled with a prefix T_i from all other paths and cycles by a distributed algorithm?

- The recursive definition of Thue–Morse can be applied backwards
  $\Rightarrow$ Given sequence T_i, get back to $T_0 = 0$.
- $\ldots T_i T_i T_i \ldots$ does not appear in the Thue–Morse sequence
  $\Rightarrow$ A cycle graph looks different from a path graph.

- A promising idea:
  - Yes-instance: a path labelled with a prefix of the Thue–Morse sequence.
  - No-instance: anything else.

Formalising the idea: the graph problem Π (1/2)

- Define the set of *valid* words over $\{0, 1, \_\}$:
  - _0_ is valid,
  - if X is valid and Y is obtained from X by mapping $0 \mapsto 0\_1\_1\_0$ and $1 \mapsto 1\_0\_0\_1$, then Y is valid.

- The valid words are prefixes of length 4^k of the Thue–Morse sequence, with a separator _ added at the beginning, between each pair of consecutive symbols, and at the end.

The decision graph problem Π (2/2)

- Local inputs from $\{A, B, C\} \times \{0, 1, \_\}$.
- Local outputs from $\{\text{yes}, \text{no}\}$.

- An instance is a yes-instance if and only if
  - the graph is a path graph,
  - the first parts of the local inputs define a consistent orientation for the path: $\ldots$ABCABCABC$\ldots$,
  - the second parts of the local inputs define a *valid* word over $\{0, 1, \_\}$.

The algorithm: a high-level idea (1/2)

In each node v of G:

1. Verify degree and orientation: if $\deg(v) \in \{1, 2\}$ and the orientation is locally consistent, continue; otherwise, reject.
   $\Rightarrow$ G is essentially an oriented path, with a port-numbering.

The algorithm: a high-level idea (1/2)

In each node v of G:

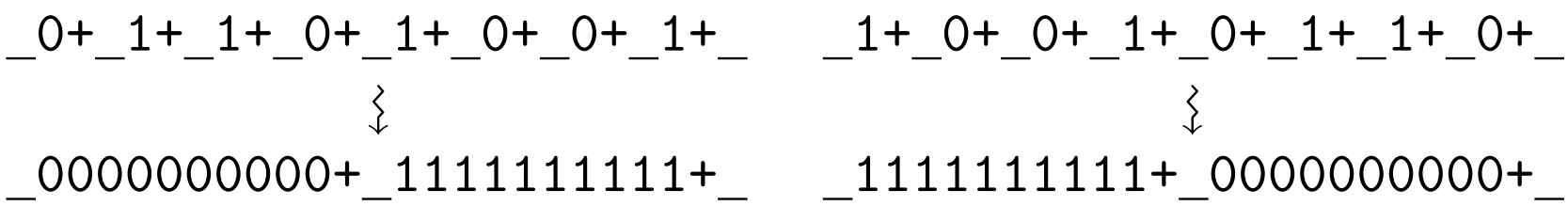
1. Verify degree and orientation: if $\deg(v) \in \{1, 2\}$ and the orientation is locally consistent, continue; otherwise, reject.
   $\Rightarrow$ G is essentially an oriented path, with a port-numbering.

2. Verify the input word locally: if every other label is from $\{0, 1\}$ and every other label is _, continue; otherwise, reject.
   $\Rightarrow$ Copy the input label as the *current label* of v.
   $\Rightarrow$ Maintain an invariant: always a separator _ at some finite distance.

The algorithm: a high-level idea (2/2)

In each node v of G:

3. Apply the recursive definition of Thue–Morse backwards:



If the pattern does not match or the new label for v is ambiguous, reject; otherwise, repeat.

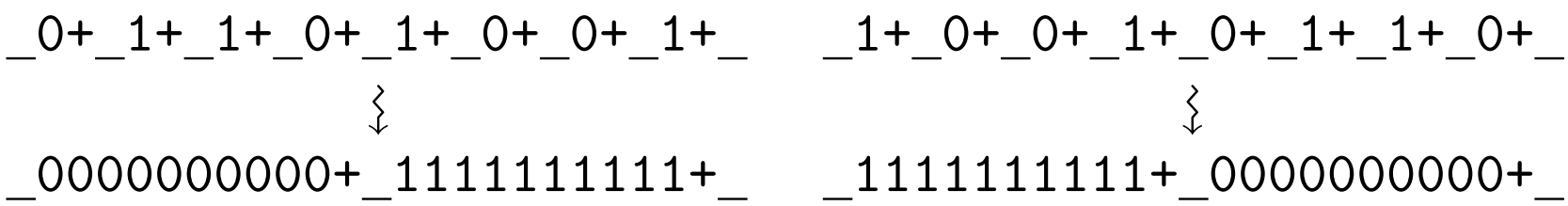
$\Rightarrow$ The invariant is maintained.

$\Rightarrow$ The word encoded in the path goes consistently from T_{2j} to $T_{2(j-1)}$.

The algorithm: a high-level idea (2/2)

In each node v of G:

3. Apply the recursive definition of Thue–Morse backwards:



   If the pattern does not match or the new label for v is ambiguous, reject; otherwise, repeat.
   $\Rightarrow$ The invariant is maintained.
   $\Rightarrow$ The word encoded in the path goes consistently from T_{2j} to $T_{2(j-1)}$.

4. If the word matches |_0+_| or |_0+_1+_1+_0+_|, accept.
   (Here | denotes the end of the path.)

Examples (1/3)

- Path graph, yes-instance:

_0_1_1_0_1_0_0_1_1_0_0_1_0_1_1_0_

$\Downarrow$ (unambiguous substitutions)

_0000000_1111111_1111111_0000000_

$\Downarrow$

accept

Examples (2/3)

- Path graph, no-instance:

$$\_0\_1\_1\_0\_1\_0\_0\_1\_1\_0\_1\_0\_0\_1\_$$

$$\Downarrow$$

$$\_0000000\_1111111\_\ldots$$

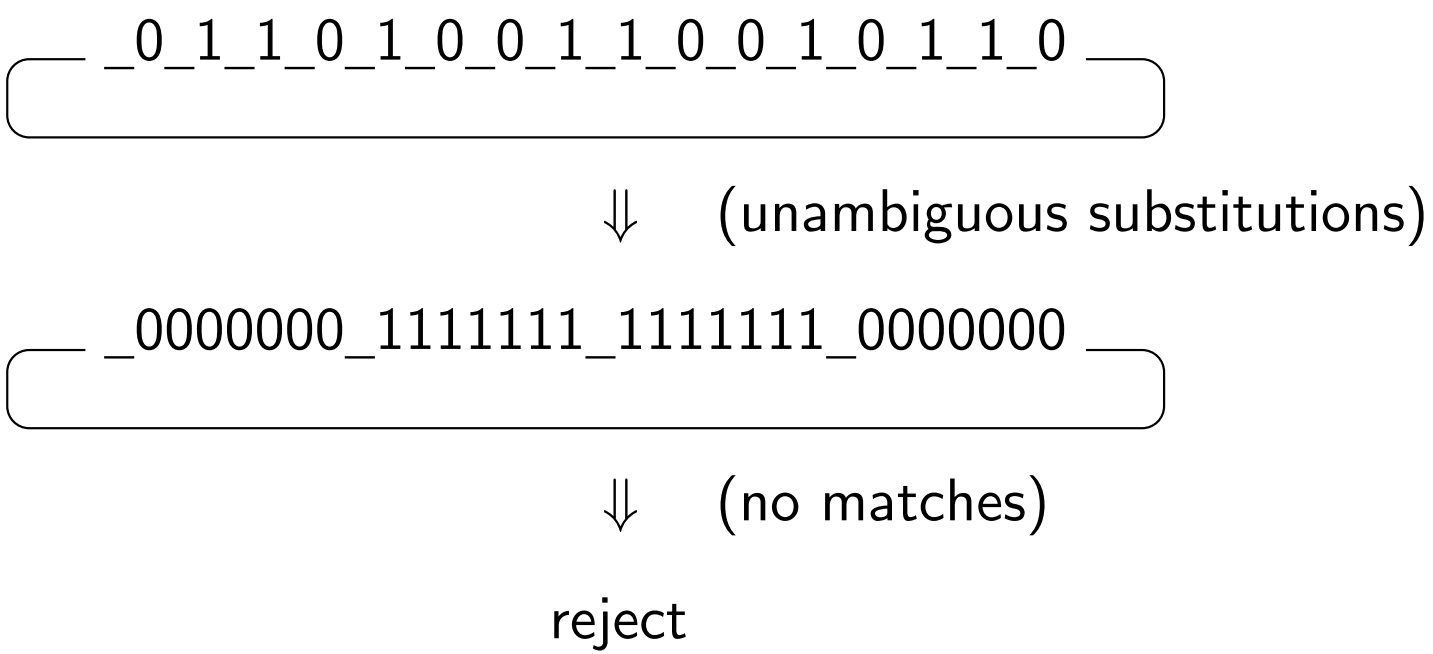
$$\ldots\_0000000\_1111111\_$$

$$\Downarrow \quad \text{(ambiguous substitutions)}$$

reject

Examples (3/3)

- Cycle graph:



Complexity

- The substitutions involve constant number of blocks separated by _'s
  $\Rightarrow$ constant space is enough.

- Need to receive information from the other end of the path
  $\Rightarrow$ $\Omega(n)$ time is needed – even if we have unique IDs or randomness.

- Substitution phase i takes $O(c^i)$ rounds (c constant), $O(\log n)$ phases
  $\Rightarrow$ $O(n)$ time is enough.

Conclusion

- Distributed time complexity is now a well-established topic.
- Research on distributed space complexity is still in its infancy.
- We proved a strong separation between constant space and constant time by introducing a graph problem that
  - can be solved in constant space in a very limited model,
  - requires linear time in strong models (e.g. LOCAL with randomness).
- However, our problem is highly artificial. It is open, whether there exist
  - natural graph problems, or
  - LCL problems

  with the above properties.

Conclusion

- Distributed time complexity is now a well-established topic.
- Research on distributed space complexity is still in its infancy.
- We proved a strong separation between constant space and constant time by introducing a graph problem that
  - can be solved in constant space in a very limited model,
  - requires linear time in strong models (e.g. LOCAL with randomness).
- However, our problem is highly artificial. It is open, whether there exist
  - natural graph problems, or
  - LCL problems

  with the above properties.

Thanks!