Constant Space and Non-Constant Time in Distributed Computing

Tuomo Lempiäinen and Jukka Suomela

Aalto University, Finland

OPODIS
20th December 2017
Lisbon, Portugal

Time complexity versus space complexity

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

- What can be said in the distributed setting?
- A message-passing model:
  - problem instance = communication graph + local inputs,
  - time = number of synchronous communication rounds,
  - space = number of bits per node needed to represent the states.

Time vs. space in the distributed setting

- A message-passing model:
  - problem instance = communication graph + local inputs,
  - time = number of synchronous communication rounds,
  - space = number of bits per node needed to represent the states.
- Constant time complexity $\Rightarrow$ constant space complexity.
- Does the converse hold?

Time vs. space in the distributed setting

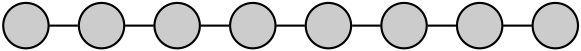
- A message-passing model:
  - problem instance = communication graph + local inputs,
  - time = number of synchronous communication rounds,
  - space = number of bits per node needed to represent the states.
- Constant time complexity $\Rightarrow$ constant space complexity.
- Does the converse hold?
- 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:
  - problem instance = communication graph + local inputs,
  - time = number of synchronous communication rounds,
  - space = number of bits per node needed to represent the states.
- 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)

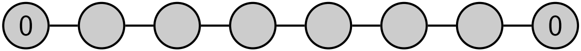
- Easy to construct constant-space non-constant-time problems if
  - promise that the graph is a path, or
  - nodes do not need to halt.
- Count the distance modulo 2 to the nearest degree-1 node:



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.

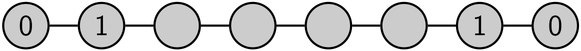
- Count the distance modulo 2 to the nearest degree-1 node:



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.

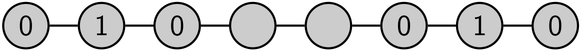
- Count the distance modulo 2 to the nearest degree-1 node:



What are the right assumptions? (1/2)

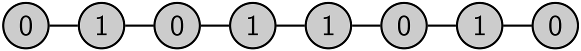
- Easy to construct constant-space non-constant-time problems if
  - promise that the graph is a path, or
  - nodes do not need to halt.

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



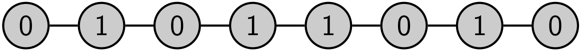
What are the right assumptions? (1/2)

- 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:

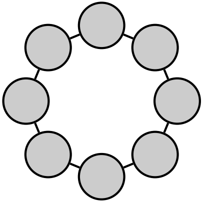


What are the right assumptions? (1/2)

- 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:

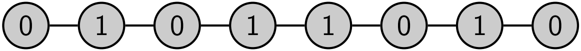


- But what if the input is a cycle?

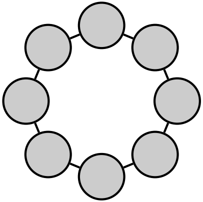


What are the right assumptions? (1/2)

- 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?

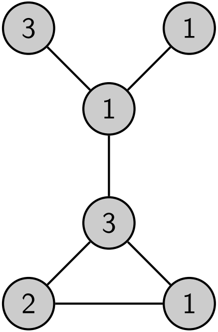


- Our result does not require any promises about the input.

What are the right assumptions? (2/2)

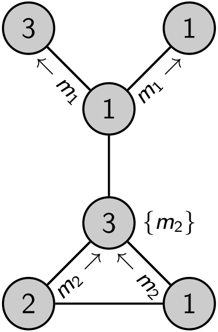
- To achieve a strong separation result, we want a graph problem Π that
  - is solvable in constant space in a very weak model of computation,
  - cannot be solved in constant time even in a very strong model.
- Hence, we will present an algorithm for Π in a very weak model of computation:
  - no unique IDs,
  - no randomness,
  - only constant-size local inputs,
  - only weak communication capabilities.

Model of computation



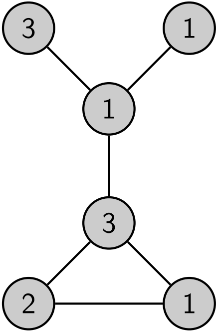
- A simple finite connected undirected graph, with constant-size local inputs.
- An identical deterministic state machine on each node.

Model of computation



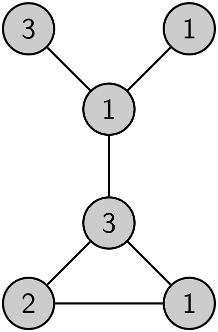
- A simple finite connected undirected graph, with constant-size local inputs.
- An identical deterministic state machine on each node.
- Computation proceeds in synchronous rounds:
  1. broadcast a message to neighbours,
  2. receive a set of messages,
  3. set a new state based on previous state and received messages.

Model of computation



- A simple finite connected undirected graph, with constant-size local inputs.
- An identical deterministic state machine on each node.
- Computation proceeds in synchronous rounds:
    1. broadcast a message to neighbours,
    2. receive a set of messages,
    3. set a new state based on previous state and received messages.
- In all graphs, each node eventually halts and produces an output.

Complexity measures



Given an algorithm (a state machine), its

- running time or time complexity is the number of communication rounds until all nodes have halted,
- space complexity is 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.

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:

$$
\begin{aligned}
T_0 &= 0 \\
T_1 &= 01 \\
T_2 &= 0110 \\
T_3 &= 01101001 \\
T_4 &= 0110100110010110 \\
&\vdots
\end{aligned}
$$

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:

$$\begin{aligned} T_0 &= 0 \\ T_1 &= 01 \\ T_2 &= 0110 \\ T_3 &= 01101001 \\ T_4 &= 0110100110010110 \\ &\vdots \end{aligned}$$

- 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 $X \in \{0, 1\}^*$.

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 $T_0 = 0$.
- $\dots T_i T_i T_i \dots$ 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 $\{yes, no\}$.

- 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:

$$\_0+\_1+\_1+\_0+\_1+\_0+\_0+\_1+\_ \quad \_1+\_0+\_0+\_1+\_0+\_1+\_1+\_0+\_$$
$$\rightsquigarrow$$
$$\_000000000+\_111111111+\_ \quad \_111111111+\_000000000+\_$$

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:

   _0+_1+_1+_0+_1+_0+_0+_1+_   _1+_0+_0+_1+_0+_1+_1+_0+_

   ⇝   ⇝

   _000000000+_111111111+_   _111111111+_000000000+_

   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:

_0_1_1_0_1_0_0_1_1_0_0_1_0_1_1_0

⇓ (unambiguous substitutions)

_0000000_1111111_1111111_0000000

⇓ (no matches)

reject

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

- 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 (locally checkable labelling) problems

  with the above properties.

Conclusion

- 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 (locally checkable labelling) problems

  with the above properties.

Thanks! Questions?