# On the existence of constant-space non-constant-time distributed algorithms

Tuomo Lempiäinen

(joint work with Jukka Suomela)

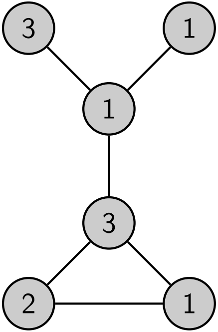
Helsinki Algorithms Seminar, Aalto University
1st December 2016

# Introduction

- We study distributed algorithms in bounded-degree graphs, with constant-size local input.
- Constant running time implies constant number of states.

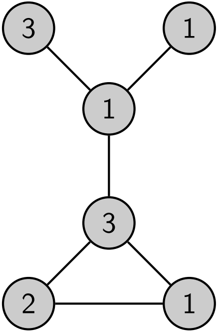
- What about the other direction?
- Does there exist a graph problem that can be solved in constant-space but requires more than constant time?
- If yes, in which class of graphs? (E.g. the class of path graphs would be trivial.)

# Model of computation



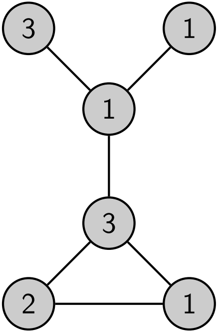

- A simple finite connected undirected graph, with constant-size local inputs.
- An identical deterministic state machine on each node.
- Computation proceeds in synchronous rounds:
  1. broadcast a message to neighbours,
  2. receive a set of messages,
  3. set a new state based on previous state and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

# 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.
- Each node eventually halts and produces an output.

# Complexity measures

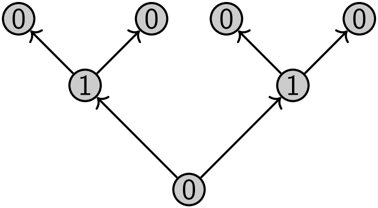



Given an algorithm (a state machine),

- its running time or time complexity is the number of communication rounds until all nodes have halted,
- its space complexity is the number of states that are visited at least once,

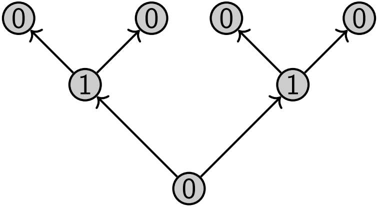
as a function of $n$, over all graphs of $n$ nodes and of maximum degree at most $\Delta$.

# Warm-up: count distance mod 2




- No local input; local outputs from $\{0, 1, \perp\}$.
- If the graph is a binary tree where each edge is directed towards the leaves, output the distance modulo 2 to the closest leaf node. Otherwise, output $\perp$.
- Edge directions can be encoded in the structure of the graph.

# Warm-up: count distance mod 2




- No local input; local outputs from $\{0, 1, \perp\}$.
- If the graph is a binary tree where each edge is directed towards the leaves, output the distance modulo 2 to the closest leaf node. Otherwise, output $\perp$.
- Edge directions can be encoded in the structure of the graph.
- If the graph is not of the desired type, at least one node can detect it locally and inform other nodes.
- The root node needs $\Theta(\log n)$ communication rounds until it knows its parity.

# Graphs of maximum degree 2

The following was already known:

**Theorem (Kuusisto 2014)**

*There exist a distributed algorithm that always halts but has a non-constant running time in the class of finite graphs of maximum degree 2.*

However, this algorithm has a non-constant space complexity.

# The main result

**Theorem**

*There exists a graph decision problem $P$ and a constant-space distributed algorithm $A$ such that*

- *algorithm $A$ solves problem $P$,*
- *$P$ requires at least a linear running time.*

# Preliminaries

- The *Thue–Morse sequence* is a sequence over $\{0, 1\}$ obtained by
  - starting with 0,
  - appending the Boolean complement of the sequence obtained so far.
- First steps:

  0
  01
  0110
  01101001
  0110100110010110
  $\vdots$

# Preliminaries

- The *Thue–Morse sequence* is a sequence over $\{0,1\}$ obtained by
    - starting with 0,
    - appending the Boolean complement of the sequence obtained so far.
- First steps:

$$\begin{array}{l} 0 \\ 01 \\ 0110 \\ 01101001 \\ 0110100110010110 \\ \vdots \end{array}$$

- Interesting property: does not contain any cubes, i.e. subwords $xxx$ for any $x \in \{0,1\}^*$

# Preliminaries

- An equivalent definition by a Lindenmayer system:
  - variables: 0, 1
  - constants: none
  - start: 0
  - production rules: $(0 \mapsto 01)$, $(1 \mapsto 10)$

# The decision problem

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

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

# The decision problem

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

- An instance is a yes-instance if and only if
  - the graph is a path,
  - first parts of the local inputs define a consistent orientation: ABCABCABC. . . ,
  - second parts of the local inputs define a *valid* word over $\{0, 1, \_\}$.
- Valid words are defined recursively as follows:
  - _0_ is valid,
  - if $x$ is valid and $y$ is obtained from $x$ by applying substitutions $(0 \mapsto 0\_1\_1\_0)$ and $(1 \mapsto 1\_0\_0\_1)$ to each occurrence of 0 and 1, then $y$ is valid.

# The algorithm

- Denote the end of the path by $|$.
- Denote one or more $x$'s by $x+$.

- Each node $v$ does the following:
  1. Verify the orientation: 3 different symbols from $\{\mathsf{A}, \mathsf{B}, \mathsf{C}, |\}$ can be found within the radius-1 neighbourhood of $v$; otherwise abort.
  2. Verify the word locally: radius-1 neighbourhood is in $\{|\_0,\ 0\_0,\ 1\_1,\ 0\_1,\ \_0\_,\ \_1\_\}$; otherwise abort.
  3. ...
- Aborting means that the node sends message "abort" to its neighbours, halts and outputs no. Whenever the node receives such a message, it passes it on, halts and outputs "no".

# The algorithm

- Each node $v$ does the following:
  3. Set current symbol $c(v)$ to be the local input from $\{0, 1, \_\}$. Repeat the following steps:
     1. Gather two buffers, L and R. Initially, broadcast _ if $c(v) = \_$, otherwise $c(v)+$. If you receive L from the left, send $r(L, c(v))$ to the right, where $r(L, c(v)) = L$ if $L = Ac(v)+$ for some $A$, otherwise $r(L, c(v)) = L\_$ if $c(v) = \_$, otherwise $r(L, c(v)) = Lc(v)+$. Handle R similarly. Continue until both L and R contain eight _'s (or an end-of-the-path marker |). This can be done in constant space.

## The algorithm

- Each node $v$ does the following:
  3. Set current symbol $c(v)$ to be the local input from $\{0, 1, \_\}$. Repeat the following steps:
     1. Gather two buffers, L and R. Initially, broadcast _ if $c(v) = \_$, otherwise $c(v)+$. If you receive L from the left, send $r(L, c(v))$ to the right, where $r(L, c(v)) = L$ if $L = Ac(v)+$ for some $A$, otherwise $r(L, c(v)) = L\_$ if $c(v) = \_$, otherwise $r(L, c(v)) = Lc(v)+$. Handle R similarly. Continue until both L and R contain eight _'s (or an end-of-the-path marker $|$). This can be done in constant space.

# The algorithm

- Each node $v$ does the following:
  3. Set current symbol $c(v)$ to be the local input from $\{0, 1, \_\}$. Repeat the following steps:
     1. Gather two buffers, L and R. Initially, broadcast $\_$ if $c(v) = \_$, otherwise $c(v)+$. If you receive L from the left, send $r(L, c(v))$ to the right, where $r(L, c(v)) = L$ if $L = Ac(v)+$ for some $A$, otherwise $r(L, c(v)) = L\_$ if $c(v) = \_$, otherwise $r(L, c(v)) = Lc(v)+$. Handle R similarly. Continue until both L and R contain eight $\_$'s (or an end-of-the-path marker $|$). This can be done in constant space.
     2. If L$c(v)$R matches $|\_0 + \_|$ or $|\_0 + \_1 + \_1 + \_0 + \_|$, halt and output yes.
     3. Apply the following substitution to the word L$c(v)$R:
        $\_0{+}\_1{+}\_1{+}\_0{+}\_1{+}\_0{+}\_0{+}\_1{+}\_ \mapsto \_0{+}00{+}00{+}00{+}\_1{+}11{+}11{+}11{+}\_$.
        If the pattern matches in several positions, and they result in different new symbols for node $v$, abort. If the pattern does not match, abort. Otherwise, update $c(v)$ according to the substitution.

  This constitutes one *phase* in the execution.

# The algorithm

We call the sequence of all the current symbols $c(v)$ a *configuration*.

**Lemma**

*Assume that in the current configuration, each maximal subword of 0's or 1's is of length $\ell$. If the algorithm is executed for one phase and no node aborts, in the resulting configuration the length is $4\ell + 3$.*

This guarantees that

- each phase completes in a finite amount of time,
- nodes agree on when to start a new phase.

It also follows that the algorithm always halts in finite graphs.

# Accepting a yes-instance

We call a word $\_x_1^i\_x_2^i\_ \ldots \_x_p^i\_$ a *padded Thue–Morse word of length $p$* if $x_1 x_2 \ldots x_p$ is a prefix of the Thue–Morse sequence.

**Lemma**

*If the current configuration is a padded Thue–Morse word of length $4^k$ and the algorithm is executed for one phase without aborting, the resulting configuration is a padded Thue–Morse word of length $4^{k-1}$.*

From this we can derive that in a yes-instance, each node eventually outputs "yes".

# Rejecting a no-instance

**Lemma**

*In a no-instance, each node eventually outputs "no".*

*Proof idea:*

- Assume for a contradiction that a no-instance gets accepted.
- If there is a yes-instance of the same size, it also gets accepted.
- Consider the first phase after which the configurations are identical in both cases.
- In the previous configurations, there were two different subwords that were replaced by $\_0+\_1+\_$. This can be shown to be a contradiction.
- Cycle graphs and paths of wrong length can also be shown to be rejected.

# Running time

- Let $\ell$ be the length of maximal subwords of 0's or 1's. Gathering the buffers takes $8\ell + 8 = 8(\ell + 1)$ rounds.
- Recall the lemma: the length of the maximal subwords increases from $\ell$ to $4\ell + 3$ in one phase.
- There are roughly $\frac{1}{2} \log n$ phases before halting.
- The running time is thus $8(1 + 1) + 8(7 + 1) + 8(31 + 1) + \cdots + 8(2^{2((\log n)/2)-1}) \leq 8n$ rounds.

# Running time

- Let $\ell$ be the length of maximal subwords of 0's or 1's. Gathering the buffers takes $8\ell + 8 = 8(\ell + 1)$ rounds.
- Recall the lemma: the length of the maximal subwords increases from $\ell$ to $4\ell + 3$ in one phase.
- There are roughly $\frac{1}{2} \log n$ phases before halting.
- The running time is thus $8(1+1) + 8(7+1) + 8(31+1) + \cdots + 8(2^{2((\log n)/2)-1}) \leq 8n$ rounds.
- This is asymptotically tight.

# Conclusion

- We presented graph problems with
  - logarithmic (maximum degree 3) and
  - linear (maximum degree 2)

  time complexity, when restricted to constant space.

- Possible future direction: other time complexity classes under the constant-space assumption?

# Conclusion

- We presented graph problems with
  - logarithmic (maximum degree 3) and
  - linear (maximum degree 2)

  time complexity, when restricted to constant space.

- Possible future direction: other time complexity classes under the constant-space assumption?

Thanks!