Deterministic Leader Election in $O(D + \log n)$ Rounds with Messages of size $O(1)$

A. Casteigts, Y. Métivier, J.M. Robson, A. Zemmari

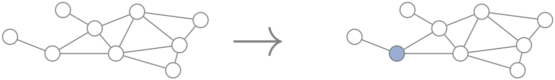
LaBRI - University of Bordeaux

DISC 2016, Paris

Leader election

“[Likely one of] the two most studied tasks in distributed computing literature” (Dinitz et al. JACM). More than 1500 papers published on this problem.

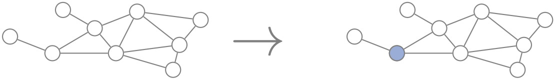
Def.: A distributed algorithm solves the election problem if it always terminates and in the final configuration exactly one process is marked as *elected* and all others are *non-elected*.



Leader election

"[Likely one of] the two most studied tasks in distributed computing literature" (Dinitz et al. JACM). More than 1500 papers published on this problem.

Def.: A distributed algorithm solves the election problem if it always terminates and in the final configuration exactly one process is marked as *elected* and all others are *non-elected*.



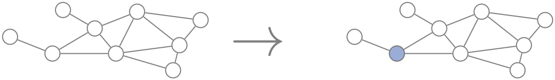
3 directions

- Feasibility of deterministic LE in anonymous graphs
- Complexity of deterministic LE with unique identifiers
- Complexity of randomized LE (in anonymous graphs)

Leader election

“[Likely one of] the two most studied tasks in distributed computing literature” (Dinitz et al. JACM). More than 1500 papers published on this problem.

Def.: A distributed algorithm solves the election problem if it always terminates and in the final configuration exactly one process is marked as *elected* and all others are *non-elected*.



3 directions

- Feasibility of deterministic LE in anonymous graphs
- Complexity of deterministic LE with unique identifiers
- Complexity of randomized LE (in anonymous graphs)

This talk.

Model and complexity measures

The model

- Synchronous rounds (and simultaneous wakeup)
  1. Send messages to neighbors
  2. Receive messages from neighbors
  3. Perform computation
- Possibility to send different messages to different neighbors
- Identifiers of size $O(\log n)$
- Messages of size $O(1)$

Model and complexity measures

The model

- Synchronous rounds (and simultaneous wakeup)
  1. Send messages to neighbors
  2. Receive messages from neighbors
  3. Perform computation
- Possibility to send different messages to different neighbors
- Identifiers of size $O(\log n)$
- Messages of size $O(1)$

Does this model have a name?

Model and complexity measures

The model

- Synchronous rounds (and simultaneous wakeup)
  1. Send messages to neighbors
  2. Receive messages from neighbors
  3. Perform computation
- Possibility to send different messages to different neighbors
- Identifiers of size $O(\log n)$
- Messages of size $O(1)$ *Does this model have a name?*

Bit complexity (3 possible measures)

1. Total number of bits exchanged [Van Leeuwen et al.'87]
2. Max number of bits per edge [Schneider and Waterhofer'11]
3. Bit round complexity [Kothapalli et al.'06]
   = #rounds with 1-bit message

Model and complexity measures

The model

- Synchronous rounds (and simultaneous wakeup)
  1. Send messages to neighbors
  2. Receive messages from neighbors
  3. Perform computation
- Possibility to send different messages to different neighbors
- Identifiers of size $O(\log n)$
- Messages of size $O(1)$ *Does this model have a name?*

Bit complexity (3 possible measures)

1. Total number of bits exchanged [Van Leeuwen et al.'87]
2. Max number of bits per edge [Schneider and Waterhofer'11]
3. Bit round complexity [Kothapalli et al.'06]
   = #rounds with 1-bit message

$\rightarrow$ Measure 3 captures both time and communication complexities (silence is not free)

Best bounds

	Time	# Messages	Message size	# Bit rounds
Awerbuch'87	$O(n)$	$\boldsymbol{\Theta(\vert E\vert + n\log n)}$	$O(\log n)$ bits	$O(n\log n)$
Peleg'90	$\boldsymbol{\Theta(D)}$	$O(D\vert E\vert)$	$O(\log n)$ bits	$O(D\log n)$

Can we be both optimal in time and messages?
(in our setting: deterministic algorithms without knowledge)

Best bounds

	Time	# Messages	Message size	# Bit rounds
Awerbuch'87	$O(n)$	$\boldsymbol{\Theta(\lvert E \rvert + n \log n)}$	$O(\log n)$ bits	$O(n \log n)$
Peleg'90	$\boldsymbol{\Theta(D)}$	$O(D \lvert E \rvert)$	$O(\log n)$ bits	$O(D \log n)$

Can we be both optimal in time and messages?
(in our setting: deterministic algorithms without knowledge)

$\rightarrow$ Not with $O(\log n)$-size messages [Kutten et al. 2015].

Best bounds

	Time	# Messages	Message size	# Bit rounds
Awerbuch'87	$O(n)$	$\boldsymbol{\Theta(\lvert E\rvert + n\log n)}$	$O(\log n)$ bits	$O(n\log n)$
Peleg'90	$\boldsymbol{\Theta(D)}$	$O(D\lvert E\rvert)$	$O(\log n)$ bits	$O(D\log n)$

Can we be both optimal in time and messages?
(in our setting: deterministic algorithms without knowledge)

$\rightarrow$ Not with $O(\log n)$-size messages [Kutten et al. 2015].

What about bit round complexity?

Best bounds

	Time	# Messages	Message size	# Bit rounds
Awerbuch'87	$O(n)$	$\mathbf{\Theta(m + n \log n)}$	$O(\log n)$ bits	$O(n \log n)$
Peleg'90	$\mathbf{\Theta(D)}$	$O(Dm)$	$O(\log n)$ bits	$O(D \log n)$
$\mathcal{STT}$	$O(D + \log n)$	$O((D + \log n)m)$	$O(1)$ bits	$\mathbf{\Theta(D + \log n)}$

Can we be both optimal in time and messages?
(in our setting: deterministic algorithms without knowledge)

$\rightarrow$ Not with $O(\log n)$-size messages [Kutten et al. 2015].

What about bit round complexity?

This paper: Algorithm $\mathcal{STT}$ + matching LB $\rightarrow$ opt. bit round complexity (utcf)
($\rightarrow$ bit round complexity captures both time and communication complexities)

Best bounds

	Time	# Messages	Message size	# Bit rounds
Awerbuch'87	$O(n)$	$\boldsymbol{\Theta(m + n\log n)}$	$O(\log n)$ bits	$O(n\log n)$
Peleg'90	$\boldsymbol{\Theta(D)}$	$O(Dm)$	$O(\log n)$ bits	$O(D\log n)$
$\mathcal{STT}$	$O(D + \log n)$	$O((D + \log n)m)$	$O(1)$ bits	$\boldsymbol{\Theta(D + \log n)}$

Can we be both optimal in time and messages?
(in our setting: deterministic algorithms without knowledge)

$\rightarrow$ Not with $O(\log n)$-size messages [Kutten et al. 2015].

What about bit round complexity?

This paper: Algorithm $\mathcal{STT}$ + matching LB $\rightarrow$ opt. bit round complexity (utcf)
($\rightarrow$ bit round complexity captures both time and communication complexities)

Lower bound — $\Omega(D + \log n)$ bit rounds

- $2\lceil \log_2((Id_{max} + 2)/3.5)\rceil = \Omega(\log n)$ bits — [Dinitz and Solomon'07]
- $\Omega(D)$ messages, even of size $O(\log n)$ — [Kutten et al.'15]

Algorithm $\mathcal{STT}$

The algorithm (no knowledge required on the graph)

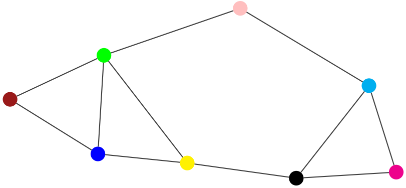
1. A spreading algorithm that broadcasts the largest id to each node
2. A spanning tree algorithm that is associated to the spreading actions;
3. A termination detection algorithm from the leaves up to the root (highest ID), which becomes elected when it detects termination.

Algorithm $\mathcal{STT}$

The algorithm (no knowledge required on the graph)

1. A spreading algorithm that broadcasts the largest id to each node
2. A spanning tree algorithm that is associated to the spreading actions;
3. A termination detection algorithm from the leaves up to the root (highest ID), which becomes elected when it detects termination.

This general principle is folklore. Takes $O(D)$ time in the CONGEST model.

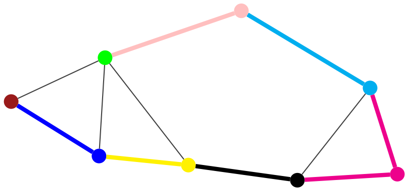


Algorithm $\mathcal{STT}$

The algorithm (no knowledge required on the graph)

1. A spreading algorithm that broadcasts the largest id to each node
2. A spanning tree algorithm that is associated to the spreading actions;
3. A termination detection algorithm from the leaves up to the root (highest ID), which becomes elected when it detects termination.

This general principle is folklore. Takes $O(D)$ time in the CONGEST model.

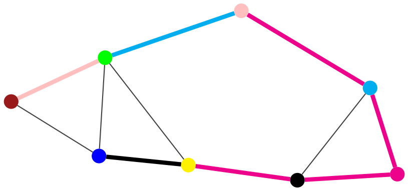


Algorithm $\mathcal{STT}$

The algorithm (no knowledge required on the graph)

1. A spreading algorithm that broadcasts the largest id to each node
2. A spanning tree algorithm that is associated to the spreading actions;
3. A termination detection algorithm from the leaves up to the root (highest ID), which becomes elected when it detects termination.

This general principle is folklore. Takes $O(D)$ time in the CONGEST model.

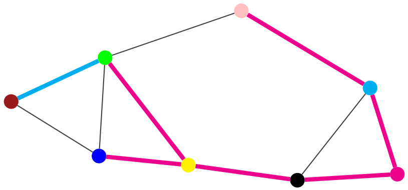


Algorithm $\mathcal{STT}$

The algorithm (no knowledge required on the graph)

1. A spreading algorithm that broadcasts the largest id to each node
2. A spanning tree algorithm that is associated to the spreading actions;
3. A termination detection algorithm from the leaves up to the root (highest ID), which becomes elected when it detects termination.

This general principle is folklore. Takes $O(D)$ time in the CONGEST model.

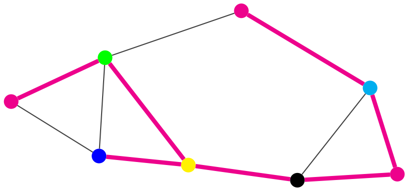


Algorithm $\mathcal{STT}$

The algorithm (no knowledge required on the graph)

1. A spreading algorithm that broadcasts the largest id to each node
2. A spanning tree algorithm that is associated to the spreading actions;
3. A termination detection algorithm from the leaves up to the root (highest ID), which becomes elected when it detects termination.

This general principle is folklore. Takes $O(D)$ time in the CONGEST model.

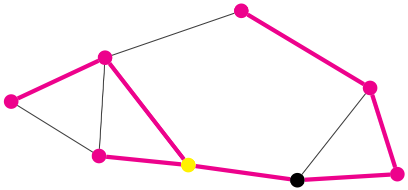


Algorithm $\mathcal{STT}$

The algorithm (no knowledge required on the graph)

1. A spreading algorithm that broadcasts the largest id to each node
2. A spanning tree algorithm that is associated to the spreading actions;
3. A termination detection algorithm from the leaves up to the root (highest ID), which becomes elected when it detects termination.

This general principle is folklore. Takes $O(D)$ time in the CONGEST model.

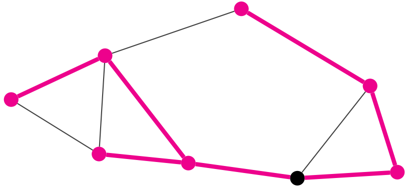


Algorithm $\mathcal{STT}$

The algorithm (no knowledge required on the graph)

1. A spreading algorithm that broadcasts the largest id to each node
2. A spanning tree algorithm that is associated to the spreading actions;
3. A termination detection algorithm from the leaves up to the root (highest ID), which becomes elected when it detects termination.

This general principle is folklore. Takes $O(D)$ time in the CONGEST model.

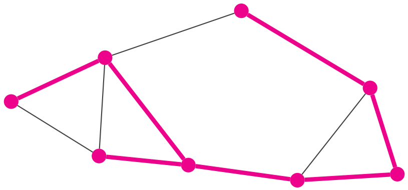


Algorithm $\mathcal{STT}$

The algorithm (no knowledge required on the graph)

1. A spreading algorithm that broadcasts the largest id to each node
2. A spanning tree algorithm that is associated to the spreading actions;
3. A termination detection algorithm from the leaves up to the root (highest ID), which becomes elected when it detects termination.

This general principle is folklore. Takes $O(D)$ time in the CONGEST model.

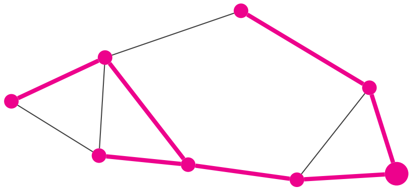


Algorithm $\mathcal{STT}$

The algorithm (no knowledge required on the graph)

1. A spreading algorithm that broadcasts the largest id to each node
2. A spanning tree algorithm that is associated to the spreading actions;
3. A termination detection algorithm from the leaves up to the root (highest ID), which becomes elected when it detects termination.

This general principle is folklore. Takes $O(D)$ time in the CONGEST model.

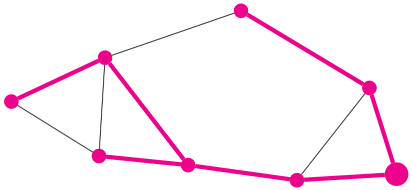


Algorithm $\mathcal{STT}$

The algorithm (no knowledge required on the graph)

1. A spreading algorithm that broadcasts the largest id to each node
2. A spanning tree algorithm that is associated to the spreading actions;
3. A termination detection algorithm from the leaves up to the root (highest ID), which becomes elected when it detects termination.

This general principle is folklore. Takes $O(D)$ time in the CONGEST model.



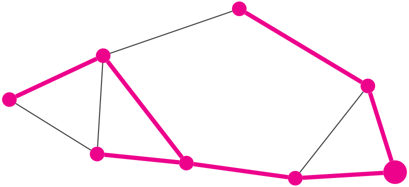
Trivial adaptation in $O(D \log n)$ bit rounds.

Algorithm $\mathcal{STT}$

The algorithm (no knowledge required on the graph)

1. A spreading algorithm that broadcasts the largest id to each node
2. A spanning tree algorithm that is associated to the spreading actions;
3. A termination detection algorithm from the leaves up to the root (highest ID), which becomes elected when it detects termination.

This general principle is folklore. Takes $O(D)$ time in the CONGEST model.



Trivial adaptation in $O(D \log n)$ bit rounds.

$\rightarrow \mathcal{STT}$ takes it down to $O(D + \log n)$ bit rounds.

The main trick...

α-encoding of the identifiers

Given *Id*, we define $\alpha(Id)$ as the unary representation of the binary length of *Id*, followed by 0, followed by the binary representation of *Id*, i.e.:

$$\alpha(Id) = base1(|base2(Id)|) \cdot 0 \cdot base2(Id).$$

Ex. $Id = 25 \stackrel{2}{=} 11001$, then $\alpha(Id) = 11111011001$

The main trick...

α-encoding of the identifiers

Given *Id*, we define $\alpha(Id)$ as the unary representation of the binary length of *Id*, followed by 0, followed by the binary representation of *Id*, i.e.:

$$\alpha(Id) = base1(|base2(Id)|) \cdot 0 \cdot base2(Id).$$

Ex. $Id = 25 \stackrel{2}{=} 11001$, then $\alpha(Id) = 11111011001$

Consequence:

$$Id_u < Id_v \Leftrightarrow \alpha(Id_u) \prec \alpha(Id_v)$$

Order induced by the first distinct letter.

The main trick...

α-encoding of the identifiers

Given *Id*, we define $\alpha(Id)$ as the unary representation of the binary length of *Id*, followed by 0, followed by the binary representation of *Id*, i.e.:

$$\alpha(Id) = base1(|base2(Id)|) \cdot 0 \cdot base2(Id).$$

Ex. $Id = 25 \stackrel{2}{=} 11001$, then $\alpha(Id) = 11111011001$

Consequence:

$$Id_u < Id_v \Leftrightarrow \alpha(Id_u) \prec \alpha(Id_v)$$

Order induced by the first distinct letter.

$\rightarrow$ Decidable based on prefixes, no need to wait...

The Spreading Algorithm $\mathcal{S}$

Pipelining the identifiers

Each node maintains the largest prefix of (encoded) identifier known so far, based on that of neighbors. Here are the round actions:

1. Update local largest prefix (e.g. based on that of neighbors)
2. Send signals indicating how the prefix was updated (or not)
3. Receive signals from neighbors
4. Update local copy of each neighbor's largest prefix

The Spreading Algorithm $\mathcal{S}$

Pipelining the identifiers

Each node maintains the largest prefix of (encoded) identifier known so far, based on that of neighbors. Here are the round actions:

1. Update local largest prefix (e.g. based on that of neighbors)
2. Send signals indicating how the prefix was updated (or not)
3. Receive signals from neighbors
4. Update local copy of each neighbor's largest prefix

Exchanged signals (constant-size)

$signal \in \{append0, append1, delete1, delete2, delete3, change, null\}$
meaning that the largest known prefix was updated by:

- appending 0 or 1
- deleting one, two or three letters
- changing the last letter from 0 to 1
- leaving it unchanged

Update rules for prefix

The largest known prefix, denoted p here, is updated in each round based on current status (active or follower) and neighbors largest known prefixes (denoted p_n here) learnt through signals.

Update rules (in order of priority, only one per round)

Variables z, x, and y denote words.

(1.1) if $p = zx$ and $\exists p_n = z$ with delete signal, then delete x from p (up to $\underline{3}$ letters)

(1.2) if $p = z0x$ ($x \neq \epsilon$) and $\exists p_n = z1y$, then delete $|x|$ letters from p

(2) If $p = z0$ and $\exists p_n = z1y$, then $p \leftarrow z1$ and *status* $\leftarrow$ *follower*

(3) If $p = z$ and $\exists p_n = z1x$, then append 1 to p

(4) If $p = z$ and $\exists p_n = z0x$, then append 0 to p

(5) If *active*, then append the next bit of $\alpha(id)$ to p

(6) *null action*

Update rules for prefix

The largest known prefix, denoted p here, is updated in each round based on current status (active or follower) and neighbors largest known prefixes (denoted p_n here) learnt through signals.

Update rules (in order of priority, only one per round)

Variables z, x, and y denote words.

(1.1) if $p = zx$ and $\exists p_n = z$ with delete signal, then delete x from p (up to <u>3</u> letters)

(1.2) if $p = z0x$ ($x \neq \epsilon$) and $\exists p_n = z1y$, then delete $|x|$ letters from p

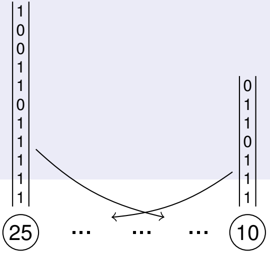
(2) If $p = z0$ and $\exists p_n = z1y$, then $p \leftarrow z1$ and *status* $\leftarrow$ *follower*

(3) If $p = z$ and $\exists p_n = z1x$, then append 1 to p

(4) If $p = z$ and $\exists p_n = z0x$, then append 0 to p

(5) If *active*, then append the next bit of $\alpha(id)$ to p

(6) *null action*



Resulting properties

Lemma 8

Let u and v be two neighbors and p_u and p_v their prefixes. The only possible cases after a round (up to renaming) are:

1. $p_u = p_v$
2. $p_u = p$ and $p_v = pw$ with $1 \leq |w| \leq 2$
3. $p_u = p0$ and $p_v = p1a$
4. $p_u = p1$ and $p_v = p0w$ and $|w| \leq 3$
5. $p_u = p$ and $p_v = pw$ and $3 \leq |w| \leq 6$ and u performed a delete

where p and w are words and a is either 0 or 1.

Resulting properties

Lemma 8

Let u and v be two neighbors and p_u and p_v their prefixes. The only possible cases after a round (up to renaming) are:

1. $p_u = p_v$
2. $p_u = p$ and $p_v = pw$ with $1 \leq |w| \leq 2$
3. $p_u = p0$ and $p_v = p1a$
4. $p_u = p1$ and $p_v = p0w$ and $|w| \leq 3$
5. $p_u = p$ and $p_v = pw$ and $3 \leq |w| \leq 6$ and u performed a delete

where p and w are words and a is either 0 or 1.

Theorem 10

After at most $6D + |\alpha(Id_{max})|$ rounds, the spreading is complete and all nodes know $\alpha(Id_{max})$.

Resulting properties

Lemma 8

Let u and v be two neighbors and p_u and p_v their prefixes. The only possible cases after a round (up to renaming) are:

1. $p_u = p_v$
2. $p_u = p$ and $p_v = pw$ with $1 \leq |w| \leq 2$
3. $p_u = p0$ and $p_v = p1a$
4. $p_u = p1$ and $p_v = p0w$ and $|w| \leq 3$
5. $p_u = p$ and $p_v = pw$ and $3 \leq |w| \leq 6$ and u performed a delete

where p and w are words and a is either 0 or 1.

Theorem 10

After at most $6D + |\alpha(Id_{max})|$ rounds, the spreading is complete and all nodes know $\alpha(Id_{max})$.

$\rightarrow O(D + \log n)$ bit rounds.

Spanning tree construction ($\rightarrow \mathcal{ST}$)

Occurs in parallel of spreading

- If u applies rule 2 relative to v, then v becomes u's parent.
- If u applies rule 3 relative to v, then v becomes (or remains) u's parent.
- If u applies rule 4 relative to v, then v becomes (or remains) u's parent.

Spanning tree construction ($\rightarrow \mathcal{ST}$)

Occurs in parallel of spreading

- If *u* applies rule 2 relative to *v*, then *v* becomes *u*'s parent.
- If *u* applies rule 3 relative to *v*, then *v* becomes (or remains) *u*'s parent.
- If *u* applies rule 4 relative to *v*, then *v* becomes (or remains) *u*'s parent.

$\rightarrow$ Additional signals (constant-size).

Termination ($\rightarrow \mathcal{STT}$)

Recursive termination proceeds from the leaves up to the root, according to the following rule.

Termination rule

If it holds that

1. u is a follower
2. u's prefix is well-formed and identical to that of neighbors
3. for every child v of u, $term_v = \texttt{true}$
4. $term_u = \texttt{false}$

then $term_u \leftarrow \texttt{true}$ and u notifies its parent

Termination ($\rightarrow \mathcal{STT}$)

Recursive termination proceeds from the leaves up to the root, according to the following rule.

Termination rule

If it holds that

1. u is a follower
2. u's prefix is well-formed and identical to that of neighbors
3. for every child v of u, $term_v = \texttt{true}$
4. $term_u = \texttt{false}$

then $term_u \leftarrow \texttt{true}$ and u notifies its parent

$\rightarrow$ Additional signals (constant-size).

Termination ($\rightarrow \mathcal{STT}$)

Recursive termination proceeds from the leaves up to the root, according to the following rule.

Termination rule

If it holds that

1. u is a follower
2. u's prefix is well-formed and identical to that of neighbors
3. for every child v of u, $term_v = \texttt{true}$
4. $term_u = \texttt{false}$

then $term_u \leftarrow \texttt{true}$ and u notifies its parent

$\rightarrow$ Additional signals (constant-size).

Remark: $term_u$ may wrongly be `true` due to local maxima and then re-become `false` (this is fine...). However, when an active node is notified by all its children, it becomes elected and correctly decides termination.

Conclusion

$\mathcal{STT}$ runs in $\Theta(D + \log n)$ bit rounds

1. Spreading phase in $O(D + \log n)$
2. Spanning tree in parallel of spreading
3. Termination detection costs $O(D)$ additional rounds

\+ Lower bound $\Omega(D + \log n)$

Open questions

- How about weaker models? E.g. beeping models?
- Is the technique useful for other problems?

Conclusion

$\mathcal{STT}$ runs in $\Theta(D + \log n)$ bit rounds

1. Spreading phase in $O(D + \log n)$
2. Spanning tree in parallel of spreading
3. Termination detection costs $O(D)$ additional rounds

\+ Lower bound $\Omega(D + \log n)$

Open questions

- How about weaker models? E.g. beeping models?
- Is the technique useful for other problems?

Thank you.