Election in anonymous networks

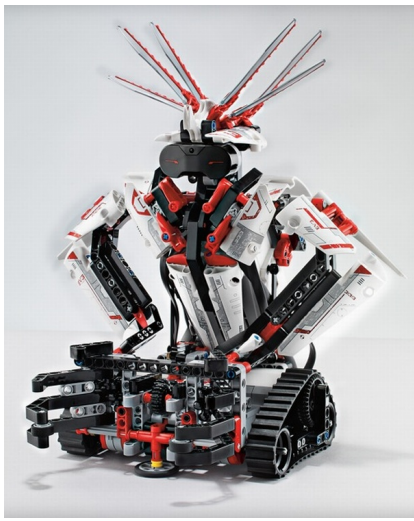
In an anonymous network, processes (and channels) have no unique id.

Processes may be anonymous for several reasons:

- Transmitting/storing id's is too expensive (IEEE 1394 bus).
- Processes don't want to reveal their id (security protocols).
- Absence of unique hardware id's (LEGO Mindstorms).

Question: Suppose there is one *leader*.

How can each process be provided with a unique id ?



Election in anonymous networks

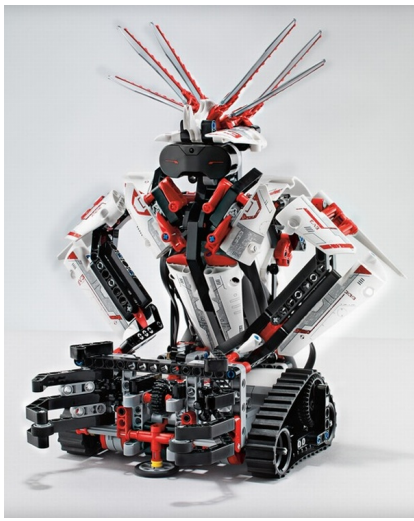
In an anonymous network, processes (and channels) have no unique id.

Processes may be anonymous for several reasons:

- Transmitting/storing id's is too expensive (IEEE 1394 bus).
- Processes don't want to reveal their id (security protocols).
- Absence of unique hardware id's (LEGO Mindstorms).

Question: Suppose there is one *leader*.

How can each process be provided with a unique id ?



Impossibility of election in anonymous rings

Theorem: There is no election algorithm for anonymous rings that always terminates.

Proof: Consider an anonymous ring of size N.

In a symmetric configuration, all processes are in the same state and all channels carry the same messages.

- There is a symmetric initial configuration.
- If γ_0 is symmetric and $\gamma_0 \rightarrow \gamma_1$, then there are transitions $\gamma_1 \rightarrow \gamma_2 \rightarrow \cdots \rightarrow \gamma_N$ with γ_N symmetric.

In a symmetric configuration there isn't one leader.

So there is an infinite computation in which no leader is elected.

Impossibility of election in anonymous rings

Theorem: There is no election algorithm for anonymous rings that always terminates.

Proof: Consider an anonymous ring of size N.

In a symmetric configuration, all processes are in the same state and all channels carry the same messages.

- There is a symmetric initial configuration.
- If γ_0 is symmetric and $\gamma_0 \rightarrow \gamma_1$, then there are transitions $\gamma_1 \rightarrow \gamma_2 \rightarrow \cdots \rightarrow \gamma_N$ with γ_N symmetric.

In a symmetric configuration there isn't one leader.

So there is an infinite computation in which no leader is elected.

Impossibility of election in anonymous rings

Theorem: There is no election algorithm for anonymous rings that always terminates.

Proof: Consider an anonymous ring of size N.

In a symmetric configuration, all processes are in the same state and all channels carry the same messages.

- There is a symmetric initial configuration.
- If γ_0 is symmetric and $\gamma_0 \to \gamma_1$, then there are transitions $\gamma_1 \to \gamma_2 \to \cdots \to \gamma_N$ with γ_N symmetric.

In a symmetric configuration there isn't one leader.

So there is an infinite computation in which no leader is elected.

Impossibility of election in anonymous rings

Theorem: There is no election algorithm for anonymous rings that always terminates.

Proof: Consider an anonymous ring of size N.

In a symmetric configuration, all processes are in the same state and all channels carry the same messages.

- There is a symmetric initial configuration.
- If γ_0 is symmetric and $\gamma_0 \to \gamma_1$, then there are transitions $\gamma_1 \to \gamma_2 \to \cdots \to \gamma_N$ with γ_N symmetric.

In a symmetric configuration there isn't one leader.

So there is an infinite computation in which no leader is elected.

Probabilistic algorithms

In a probabilistic algorithm, a process may flip a coin, and perform an event based on the outcome of this coin flip.

Probabilistic algorithms where *all* computations terminate in a *correct* configuration aren't interesting.

Because letting the coin e.g. always flip heads yields a correct non-probabilistic algorithm.

Probabilistic algorithms

In a probabilistic algorithm, a process may flip a coin, and perform an event based on the outcome of this coin flip.

Probabilistic algorithms where *all* computations terminate in a *correct* configuration aren't interesting.

Because letting the coin e.g. always flip heads yields a correct non-probabilistic algorithm.

Las Vegas and Monte Carlo algorithms

A probabilistic algorithm is Las Vegas if:

- the probability that it terminates is greater than zero, and
- all terminal configurations are correct.

It is Monte Carlo if:

- it always terminates, and
- the probability that a terminal configuration is correct is greater than zero.

Las Vegas and Monte Carlo algorithms

A probabilistic algorithm is Las Vegas if:

- the probability that it terminates is greater than zero, and
- all terminal configurations are correct.

It is Monte Carlo if:

- it always terminates, and
- the probability that a terminal configuration is correct is greater than zero.

Questions

Even if the probability that a Las Vegas algorithm terminates is 1, this doesn't always imply termination. Why is that ?

Assume a Monte Carlo algorithm, and a (deterministic) algorithm to check whether a run of the Monte Carlo algorithm terminated correctly.

Give a Las Vegas algorithm that terminates with probability 1.

Questions

Even if the probability that a Las Vegas algorithm terminates is 1, this doesn't always imply termination. Why is that ?

Assume a Monte Carlo algorithm, and a (deterministic) algorithm to check whether a run of the Monte Carlo algorithm terminated correctly.

Give a Las Vegas algorithm that terminates with probability 1.

Itai-Rodeh election algorithm

Given an anonymous, directed ring; *all processes know the ring size N*.

We adapt the Chang-Roberts algorithm: each initiator sends out an id, and the largest id is the only one making a round trip.

Each initiator selects a random id from $\{1, \ldots, N\}$.

If several processes select the same largest id, then they start a new election round, *with a higher round number*.

Itai-Rodeh election algorithm

Given an anonymous, directed ring; *all processes know the ring size N*.

We adapt the Chang-Roberts algorithm: each initiator sends out an id, and the largest id is the only one making a round trip.

Each initiator selects a random id from $\{1, \ldots, N\}$.

Complication: Different processes may select the same id.

Solution: Each message is supplied with a hop count.

A message that arrives at its source has hop count N.

If several processes select the same largest id, then they start a new election round, *with a higher round number*.

Itai-Rodeh election algorithm

Given an anonymous, directed ring; *all processes know the ring size N*.

We adapt the Chang-Roberts algorithm: each initiator sends out an id, and the largest id is the only one making a round trip.

Each initiator selects a random id from $\{1, \ldots, N\}$.

Complication: Different processes may select the same id.

Solution: Each message is supplied with a hop count.

A message that arrives at its source has hop count N.

If several processes select the same largest id, then they start a new election round, *with a higher round number*.

Itai-Rodeh election algorithm

Initially, *initiators* are active in round 0, and *noninitiators* are passive.

Let p be *active*. At the start of election round n, p randomly selects id_p, sends $(n, id_p, 1, false)$, and waits for a message (n', i, h, b).

The 3^{rd} value is the hop count. The 4^{th} value signals if another process *with the same id* was encountered during the round trip.

- p gets (n', i, h, b) with $n' > n$, or $n' = n$ and $i > id_p$: it becomes *passive* and sends $(n', i, h + 1, b)$.
- p gets (n', i, h, b) with $n' < n$, or $n' = n$ and $i < id_p$: it *dismisses* the message.
- p gets (n, id_p, h, b) with $h < N$: it sends $(n, id_p, h + 1, true)$.
- p gets $(n, id_p, N, true)$: it proceeds to round $n + 1$.
- p gets $(n, id_p, N, false)$: it becomes the leader.

Passive processes pass on messages, increasing their hop count by one.

Itai-Rodeh election algorithm

Initially, *initiators* are active in round 0, and *noninitiators* are passive.

Let p be *active*. At the start of election round n, p randomly selects id_p, sends $(n, id_p, 1, false)$, and waits for a message (n', i, h, b).

The 3^{rd} value is the hop count. The 4^{th} value signals if another process *with the same id* was encountered during the round trip.

- p gets (n', i, h, b) with $n' > n$, or $n' = n$ and $i > id_p$: it becomes *passive* and sends $(n', i, h + 1, b)$.
- p gets (n', i, h, b) with $n' < n$, or $n' = n$ and $i < id_p$: it *dismisses* the message.
- p gets (n, id_p, h, b) with $h < N$: it sends $(n, id_p, h + 1, true)$.
- p gets $(n, id_p, N, true)$: it proceeds to round $n + 1$.
- p gets $(n, id_p, N, false)$: it becomes the leader.

Passive processes pass on messages, increasing their hop count by one.

Itai-Rodeh election algorithm

Initially, *initiators* are active in round 0, and *noninitiators* are passive.

Let p be *active*. At the start of election round n, p randomly selects id_p, sends $(n, id_p, 1, \mathit{false})$, and waits for a message (n', i, h, b).

The 3^{rd} value is the hop count. The 4^{th} value signals if another process *with the same id* was encountered during the round trip.

- p gets (n', i, h, b) with $n' > n$, or $n' = n$ and $i > id_p$: it becomes *passive* and sends $(n', i, h + 1, b)$.
- p gets (n', i, h, b) with $n' < n$, or $n' = n$ and $i < id_p$: it *dismisses* the message.
- p gets (n, id_p, h, b) with $h < N$: it sends $(n, id_p, h + 1, \mathit{true})$.
- p gets $(n, id_p, N, \mathit{true})$: it proceeds to round $n + 1$.
- p gets $(n, id_p, N, \mathit{false})$: it becomes the leader.

Passive processes pass on messages, increasing their hop count by one.

Itai-Rodeh election algorithm

Initially, *initiators* are active in round 0, and *noninitiators* are passive.

Let p be *active*. At the start of election round n, p randomly selects id_p, sends $(n, id_p, 1, \mathit{false})$, and waits for a message (n', i, h, b).

The 3^{rd} value is the hop count. The 4^{th} value signals if another process *with the same id* was encountered during the round trip.

- p gets (n', i, h, b) with $n' > n$, or $n' = n$ and $i > id_p$: it becomes *passive* and sends $(n', i, h + 1, b)$.
- p gets (n', i, h, b) with $n' < n$, or $n' = n$ and $i < id_p$: it *dismisses* the message.
- p gets (n, id_p, h, b) with $h < N$: it sends $(n, id_p, h + 1, \mathit{true})$.
- p gets $(n, id_p, N, \mathit{true})$: it proceeds to round $n + 1$.
- p gets $(n, id_p, N, \mathit{false})$: it becomes the leader.

Passive processes pass on messages, increasing their hop count by one.

Itai-Rodeh election algorithm

Initially, *initiators* are active in round 0, and *noninitiators* are passive.

Let p be *active*. At the start of election round n, p randomly selects id_p, sends $(n, id_p, 1, \mathit{false})$, and waits for a message (n', i, h, b).

The 3^{rd} value is the hop count. The 4^{th} value signals if another process *with the same id* was encountered during the round trip.

- p gets (n', i, h, b) with $n' > n$, or $n' = n$ and $i > id_p$: it becomes *passive* and sends $(n', i, h + 1, b)$.
- p gets (n', i, h, b) with $n' < n$, or $n' = n$ and $i < id_p$: it *dismisses* the message.
- p gets (n, id_p, h, b) with $h < N$: it sends $(n, id_p, h + 1, \mathit{true})$.
- p gets $(n, id_p, N, \mathit{true})$: it proceeds to round $n + 1$.
- p gets $(n, id_p, N, \mathit{false})$: it becomes the leader.

Passive processes pass on messages, increasing their hop count by one.

Itai-Rodeh election algorithm

Initially, *initiators* are active in round 0, and *noninitiators* are passive.

Let p be *active*. At the start of election round n, p randomly selects id_p, sends $(n, id_p, 1, \mathit{false})$, and waits for a message (n', i, h, b).

The 3^{rd} value is the hop count. The 4^{th} value signals if another process *with the same id* was encountered during the round trip.

- p gets (n', i, h, b) with $n' > n$, or $n' = n$ and $i > id_p$: it becomes *passive* and sends $(n', i, h + 1, b)$.
- p gets (n', i, h, b) with $n' < n$, or $n' = n$ and $i < id_p$: it *dismisses* the message.
- p gets (n, id_p, h, b) with $h < N$: it sends $(n, id_p, h + 1, \mathit{true})$.
- p gets $(n, id_p, N, \mathit{true})$: it proceeds to round $n + 1$.
- p gets $(n, id_p, N, \mathit{false})$: it becomes the leader.

Passive processes pass on messages, increasing their hop count by one.

Itai-Rodeh election algorithm - Correctness

Question: How can an infinite computation occur ?

Correctness: The Itai-Rodeh election algorithm is Las Vegas.

Eventually one leader is elected, with probability 1.

Without rounds, the algorithm would be flawed.

Example:

Average-case message complexity: $O(N \log N)$

Itai-Rodeh election algorithm - Correctness

Question: How can an infinite computation occur ?

Correctness: The Itai-Rodeh election algorithm is Las Vegas.

Eventually one leader is elected, with probability 1.

Without rounds, the algorithm would be flawed.

Example:

Average-case message complexity: $O(N \log N)$

Itai-Rodeh election algorithm - Correctness

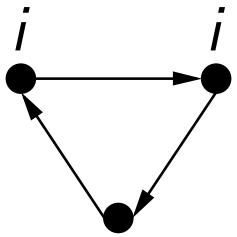
Question: How can an infinite computation occur ?

Correctness: The Itai-Rodeh election algorithm is Las Vegas.

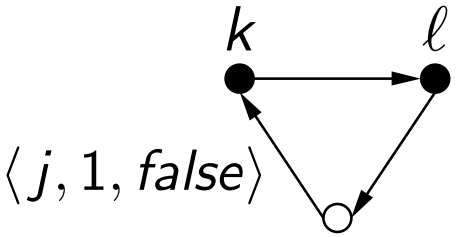
Eventually one leader is elected, with probability 1.

Without rounds, the algorithm would be flawed.

Example:



$i > j$



$j > k, \ell$

Average-case message complexity: $O(N \log N)$

Itai-Rodeh election algorithm - Correctness

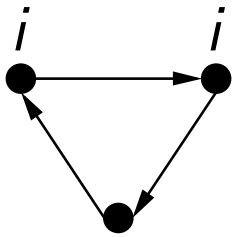
Question: How can an infinite computation occur ?

Correctness: The Itai-Rodeh election algorithm is Las Vegas.

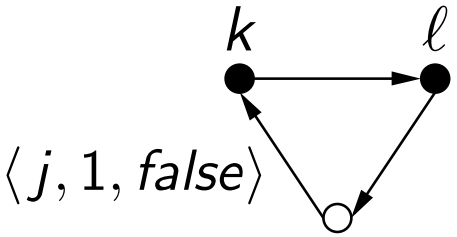
Eventually one leader is elected, with probability 1.

Without rounds, the algorithm would be flawed.

Example:



$i > j$



$j > k, \ell$

Average-case message complexity: $O(N \log N)$

Questions

How can the echo algorithm with extinction be adapted to obtain an election algorithm for arbitrary anonymous networks ?

What may happen if different processes select the same id ?

How can a process, when its wave completes, determine whether other processes selected the same id in the current round ?

You may assume that all processes know the network size.

Questions

How can the echo algorithm with extinction be adapted to obtain an election algorithm for arbitrary anonymous networks ?

What may happen if different processes select the same id ?

How can a process, when its wave completes, determine whether other processes selected the same id in the current round ?

You may assume that all processes know the network size.

Questions

How can the echo algorithm with extinction be adapted to obtain an election algorithm for arbitrary anonymous networks ?

What may happen if different processes select the same id ?

How can a process, when its wave completes, determine whether other processes selected the same id in the current round ?

You may assume that all processes know the network size.

Election in arbitrary anonymous networks

The echo algorithm with extinction, with random selection of id's, can be used for election in anonymous undirected networks in which *all processes know the network size*.

Initially, initiators are active in round 0, and noninitiators are passive.

Each active process selects a random id, and starts a wave, tagged with its id and round number 0.

Let process p in wave i of round n be hit by wave j of round n':

- If $n' > n$, or $n' = n$ and $j > i$, then p adopts wave j of round n', and treats the message according to the echo algorithm.
- If $n' < n$, or $n' = n$ and $j < i$, then p dismisses the message.
- If $n' = n$ and $j = i$, then p treats the message according to the echo algorithm.

Election in arbitrary anonymous networks

The echo algorithm with extinction, with random selection of id's, can be used for election in anonymous undirected networks in which *all processes know the network size*.

Initially, initiators are active in round 0, and noninitiators are passive.

Each active process selects a random id, and starts a wave, tagged with its id and round number 0.

Let process p in wave i of round n be hit by wave j of round n':

- If $n' > n$, or $n' = n$ and $j > i$, then p adopts wave j of round n', and treats the message according to the echo algorithm.
- If $n' < n$, or $n' = n$ and $j < i$, then p dismisses the message.
- If $n' = n$ and $j = i$, then p treats the message according to the echo algorithm.

Election in arbitrary anonymous networks

The echo algorithm with extinction, with random selection of id's, can be used for election in anonymous undirected networks in which *all processes know the network size*.

Initially, initiators are active in round 0, and noninitiators are passive.

Each active process selects a random id, and starts a wave, tagged with its id and round number 0.

Let process p in wave i of round n be hit by wave j of round n':

- If $n' > n$, or $n' = n$ and $j > i$, then p adopts wave j of round n', and treats the message according to the echo algorithm.
- If $n' < n$, or $n' = n$ and $j < i$, then p dismisses the message.
- If $n' = n$ and $j = i$, then p treats the message according to the echo algorithm.

Las Vegas and Monte Carlo algorithms

A probabilistic algorithm is Las Vegas if:

- the probability that it terminates is greater than zero, and
- all terminal configurations are correct.

It is Monte Carlo if:

- it always terminates, and
- the probability that a terminal configuration is correct is greater than zero.

Computing the size of a network

Theorem: There is no Las Vegas algorithm to compute the size of an anonymous ring.

This implies that there is no Las Vegas algorithm for election in an anonymous ring *if processes don't know the ring size*.

Because when a leader is known, the network size can be computed using a centralized wave algorithm with the leader as initiator.

Computing the size of a network

Theorem: There is no Las Vegas algorithm to compute the size of an anonymous ring.

This implies that there is no Las Vegas algorithm for election in an anonymous ring *if processes don't know the ring size*.

Because when a leader is known, the network size can be computed using a centralized wave algorithm with the leader as initiator.

Impossibility of computing anonymous network size

Theorem: There is no Las Vegas algorithm to compute the size of an anonymous ring.

Proof: Consider an anonymous, directed ring $p_0, \ldots, p_{N-1}$.

Suppose a computation C of a (probabilistic) ring size algorithm terminates with the correct outcome N.

Consider the ring $p_0, \ldots, p_{2N-1}$.

Let each event at a p_i in C be executed concurrently at p_i and p_{i+N}.

This computation terminates with the incorrect outcome N.

Question

Give an (always correctly terminating) algorithm for computing the network size of anonymous, acyclic networks.

Question

Give an (always correctly terminating) algorithm for computing the network size of anonymous, acyclic networks.

Answer: Use the tree algorithm, whereby each process reports the size of its subtree to its parent.

IEEE 1394 election algorithm

The IEEE 1394 standard is a serial multimedia bus.

It connects digital devices,
which can be added/removed dynamically.

Transmitting/storing id's is too expensive,
so the network is anonymous.

The network size is unknown to the processes.

The tree algorithm for undirected, acyclic networks is used.

Networks that contain a cycle give a time-out.

IEEE 1394 election algorithm

The IEEE 1394 standard is a serial multimedia bus.

It connects digital devices,
which can be added/removed dynamically.

Transmitting/storing id's is too expensive,
so the network is anonymous.

The network size is unknown to the processes.

The tree algorithm for undirected, acyclic networks is used.

Networks that contain a cycle give a time-out.

IEEE 1394 election algorithm

The IEEE 1394 standard is a serial multimedia bus.

It connects digital devices,
which can be added/removed dynamically.

Transmitting/storing id's is too expensive,
so the network is anonymous.

The network size is unknown to the processes.

The tree algorithm for undirected, acyclic networks is used.

Networks that contain a cycle give a time-out.

IEEE 1394 election algorithm

When a process has one possible parent, it sends a parent request to this neighbor. If the request is accepted, an ack is sent back.

The last two parentless processes can send parent requests to each other simultaneously. This is called root contention.

Each of the two processes in root contention randomly decides to either *immediately send* a parent request again, or to *wait some time* for a parent request from the other process.

Question: Is it optimal for performance to give probability 0.5 to both sending immediately and waiting for some time?

IEEE 1394 election algorithm

When a process has one possible parent, it sends a parent request to this neighbor. If the request is accepted, an ack is sent back.

The last two parentless processes can send parent requests to each other simultaneously. This is called root contention.

Each of the two processes in root contention randomly decides to either *immediately send* a parent request again, or to *wait some time* for a parent request from the other process.

Question: Is it optimal for performance to give probability 0.5 to both sending immediately and waiting for some time?