GIAN Course on Distributed Network Algorithms

Self-Stabilization
We have seen: LOCAL algorithms can be run in asynchronous environments. Today: LOCAL algorithms can also be made very robust, namely self-stabilizing!
Example: A Fault-Tolerant Concert for the Mayor

Musicians are arranged in a graph. Can see neighbors only!
Example: A Fault-Tolerant Concert for the Mayor

Setting:
- Play “happy birthday” again and again
- Wind changes pages
- Musicians can only observe immediate neighbors

Goal:
- When wind stops, harmonize eventually!
Example: A Fault-Tolerant Concert for the Mayor

**Setting:**
- Play “happy birthday” again and again
- Wind changes pages
- Musicians can only observe immediate neighbors

**Goal:**
- When wind stops, harmonize eventually!
Example: A Fault-Tolerant Concert for the Mayor

Setting:

- Play “happy birthday” again and again
- Wind changes pages
- Musicians can only observe immediate neighbors

Goal:

- When wind stops, harmonize eventually!

Algorithm to achieve this?

Idea 1: If out of sync, just change to the page of a nearby player!
But what if the neighbour does the same with his neighbor? E.g., me?? May never converge!

**Idea 1:** If out of sync, just change to the page of a nearby player!

**Setting:**
- Play “happy birthday” again and again
- Wind changes pages
- Musicians can only observe immediate neighbors

**Goal:**
- When wind stops, harmonize eventually!
Example: A Fault-Tolerant Concert for the Mayor

Setting:
- Play “happy birthday” again and again
- Wind changes pages
- Musicians can only observe immediate neighbors

Goal:
- When wind stops, harmonize eventually!

Algorithm to achieve this?

Idea 1: If out of sync, just change to the page of a nearby player!

Idea 2: Go to start when asynchrony detected!

But what if the neighbour does the same with his neighbor? E.g., me?? May never converge!
Example: A Fault-Tolerant Concert for the Mayor

Setting:
- Play “happy birthday” again and again
- Wind changes pages
- Musicians can only observe immediate neighbors

Goal:
- When wind stops, harmonize eventually!

Algorithm to achieve this?

Idea 1: If out of sync, just change to the page of a nearby player!

But what if the neighbour does the same with his neighbor? E.g., me?? May never converge!

Idea 2: Go to start when asynchrony detected!

But players further away detect it later and restart later! May never converge…
Self-stabilizing algorithms pioneered by Dijkstra (1973): for example self-stabilizing mutual exclusion.

“I regard this as Dijkstra’s most brilliant work. Self-stabilization is a very important concept in fault tolerance.”

Leslie Lamport (PODC 1983)
A distributed system is self-stabilizing if, starting from an \textit{arbitrary initial state}, it is guaranteed to converge to a \textit{legitimate state}. If the system is in a legitimate state, it is guaranteed to remain there, provided that \textit{no further faults happen}. A state is legitimate if the state satisfies the specifications of the distributed system.
The Classic: Self-Stabilizing Token Ring

Heard of token-ring networks?
A **Token Ring** network is a local area network (LAN) in which all computers are connected in a **ring** or star topology and a bit- or token-passing scheme is used in order to prevent the collision of data between two computers that want to send messages at the same time.
The Classic: Self-Stabilizing Token Ring

(Eventual) goal: A single token, circulating. E.g., mutual exclusion!
The Classic: Self-Stabilizing Token Ring

(Eventual) goal: A single token, circulating. E.g., mutual exclusion!

Assume: ring orientation is given.
The Classic: Self-Stabilizing Token Ring

(Eventual) goal: A single token, circulating. E.g., mutual exclusion!

Assume: ring orientation is given.

Assume: leader node given, and n known.
Note, given orientation, we can use the notion of child and parent.
Adversary Model

Adversary may add and remove many tokens *anytime*!
Adversary Model

Possible initial configuration!
Distributed algorithm that self-stabilizes to a single rotating token?

Adversary may add and remove many tokens anytime!

Possible initial configuration!
Idea: Each node is in a state $s \in S = \{1, \ldots, n\}$. Each node informs its child continuously about its state.
Idea: Each node is in a state $S=\{1,\ldots,n\}$. Each node informs its child continuously about its state.
Our self-stabilizing algorithm will ensure that eventually there are only two numbers: the changing point denotes the token location!

**Idea:** Each node is in a state $S=\{1, \ldots, n\}$. Each node informs its child continuously about its state.
Example

Idea: Each node is in a state $S=\{1,\ldots,n\}$. Each node informs its child continuously about its state.

Our self-stabilizing algorithm will ensure that eventually there are only two numbers: the changing point denotes the token location!
Our self-stabilizing algorithm will ensure that eventually there are only two numbers: the changing point denotes the token location!

**Example**

If \( v = v_0 \) then

If \( S(v) = S(c) \) then

\[
S(v) := S(v) + 1 \pmod{n}
\]

End If

Else \( S(v) := S(c) \)
Example

If $S(v_0) = S(c)$: increment!

Our self-stabilizing algorithm will ensure that eventually there are only two numbers: the changing point denotes the token location!

Token Ring

If $v = v_0$ then
   If $S(v) = S(c)$ then
      $S(v) := S(v) + 1 \pmod{n}$
   End If
Else $S(v) := S(c)$
Our self-stabilizing algorithm will ensure that eventually there are two numbers: the changing point denotes the token location!

Example

If $v = v_0$ then

If $S(v) = S(c)$ then

$S(v) := S(v) + 1 \pmod n$

and If

else $S(v) := S(c)$

Simply forward the value!
Example

If \( S(v_0) = S(c) \):
- increment!

Simply forward the value!

The leader chooses next ID, everyone else simply simply forwards!

Simply forward the value!

Token Ring

If \( v = v_0 \) then
- \( S(v) := S(v) + 1 \pmod{n} \)

If \( S(v) = S(c) \) then
- \( S(v) := S(c) \)

Our self-stabilizing algorithm will ensure that eventually there are two numbers: the changing point denotes the token location.
The algorithm stabilizes correctly.

1. Eventually, each node just copies from its child: same value throughout the ring.
Token Ring

The algorithm stabilizes correctly.

1. eventually, each node just copies from its child: same value throughout the ring.
Token Ring

The algorithm stabilizes correctly.

1. eventually, each node just copies from its child: same value throughout the ring.
Token Ring

The algorithm stabilizes correctly.

1. eventually, each node just copies from its child: same value throughout the ring.
Token Ring

The algorithm stabilizes correctly.

1. eventually, each node just copies from its child: same value throughout the ring.

2. Root chooses next larger value only once even the last child received the old leader value.
**Eventually:**

- The leader will reach a state $s$ that no other node had at time $t_0$. (There are $n$ nodes and $n$ states.)
- Then one node after the other will learn the current state of the leader.
- The leader itself does not push the next value until the previous value travelled the entire ring!
- At most one node active at any time: Token passed implicitly with the switching state.

larger value only once even the last child received the old leader value.
Eventually:

- The leader will reach a state \( s \) that no other node had at time \( t_0 \). (There are \( n \) nodes and \( n \) states.)
- Then one node after the other will learn the current state of the leader.
- The leader itself does not push the next value until the previous value travelled the entire ring!
- At most one node active at any time: Token passed implicitly with the switching state.

So the system “stabilizes” after at most \( n \) time units after the leader increased the value: from then on, a unique “value change” cycles the ring.
Self-Stabilizing Independent Sets

How to design self-stabilizing Maximal Independent Sets?
How to design self-stabilizing Maximal Independent Sets?

Remember algorithm:
Join MIS if all higher-ID neighbors did not.
Self-Stabilizing Independent Sets

How to design self-stabilizing Maximal Independent Sets?

Remember algorithm:
Join MIS if all higher-ID neighbors did not.

Idea: Make it self-stabilizing by executing this continuously!
Self-Stabilizing Independent Sets

Assume: node have unique IDs

Independent Sets

Every node $v$ executes the following code:

1: do atomically (forever)
2: Leave MIS if a neighbor with a larger ID is in the MIS
3: Join MIS if no neighbor with larger ID joins MIS
4: Send (node ID, MIS or not MIS) to all neighbors
5: end do
Self-Stabilizing Independent Sets

Assume: node have unique IDs

Independent Sets

Every node \( v \) executes the following code:

1: do atomically (forever)
2: Leave MIS if a neighbor with a larger ID is in the MIS
3: Join MIS if no neighbor with larger ID is in MIS
4: Send (node ID, MIS or not MIS) to all neighbors
5: end do

Why does it work? For same reason as before: eventually, highest-ID node will make decision, then its neighbors, then…
Self-Stabilizing Independent Sets

Assume: node have unique IDs

**Independent Sets**

Every node $v$ executes the following code:

1. do atomically (forever)
2. Leave MIS if a neighbor with a larger ID is in the MIS
3. Join MIS if no neighbor with larger ID joins MIS
4. Send (node ID, MIS or not MIS) to all neighbors
5. end do

Why does it work? For same reason as before: eventually, highest-ID node will make decision, then its neighbors, then…

Can we make any LOCAL algorithm self-stabilizing? E.g., coloring, matching, …?
Self-Stabilizing Independent Sets

Assume: node have unique IDs

Independent Sets

Every node $v$ executes the following code:

1: do atomically (forever)
2: Leave MIS if a neighbor with a larger ID is in the MIS
3: Join MIS if no neighbor with larger ID joins MIS
4: Send (node ID, MIS or not MIS) to all neighbors
5: end do

Why does it work? For same reason as before: eventually, highest-ID node will make decision, then its neighbors, then…

Can we make any LOCAL algorithm self-stabilizing? E.g., coloring, matching, …?

Yes! Automatic transformation.
Transforming Local Algorithms

Transformation

Given:
Deterministic k-round LOCAL algorithm A.

Output:
k-round self-stab LOCAL algorithm, i.e.:
- if the adversary does not corrupt the system for k time units, the solution is stable
- if the adversary does not corrupt any node or message closer than distance k from a node u, node u will be stable (locality)
Given:

Deterministic k-round LOCAL algorithm A.

A.k.a. *local checking*. Proof by induction: after $t_0$, round 1 variables and messages will be correct, then round 2 variables and messages, then ...

- if the adversary does not corrupt the system for $k$ time units, the solution is stable
- if the adversary does not corrupt any node or message closer than distance $k$ from a node $u$, node $u$ will be stable (*locality*)
Given:

Deterministic k-round LOCAL algorithm A.

A.k.a. *local checking*. Proof **by induction**: after t0, round 1 variables and messages will be correct, then round 2 variables and messages, then …

- if the adversary does not corrupt the system for k time units, the solution is stable
- if the adversary does not corrupt any node or message closer

It is automatic: from Art to Craft!
Sometimes stabilization is not to a fixed state but to a cyclic state! E.g., token ring. Here comes another example!
Advanced Stabilization

In a little town, each evening citizens call their friends to ask whether they vote for **Democrats or Republicans**. Then they decide themselves for **majority** (assume odd number of friends). Does this system «converge» or «stabilize»?
In a little town, each evening citizens call their friends to ask whether they vote for Democrats or Republicans. Then they decide themselves for the majority (assume odd number of friends). Does this system «converge» or «stabilize»?
Example

t:

\[
\begin{array}{c}
\text{t:} \\
\text{t+1:}
\end{array}
\]

- [Diagram of connected nodes at time t and t+1]
Example

t:

\[ t: \]

\[ t + 1: \]
Example

majority of red...

... so red.

$t$: 

$t+1$: 
What do you think?

- Does eventually everybody vote for the same party?
- Will each citizen eventually stay with the same party?
- Will citizens who stayed with the same party for some time, stay with that party forever?
- And if their friends also constantly root for the same party?
- Will this beast stabilize at all? 😊
What do you think?

- Does eventually everybody vote for the same party?
- Will each citizen eventually stay with the same party?
- Will citizens who stayed with the same party for some time, stay with that party forever?
- And if their friends also constantly root for the same party?
- Will this beast stabilize at all?

No, no, no!
Democrats / Republicans

Eventually each citizen will vote for the same party every other day.

At least one can show this:
Some kind of convergence...
Why?

Democrats / Republicans

Eventually each citizen will vote for the same party every other day.

Why?
Democrats / Republicans

Eventually each citizen will vote for the same party every other day.

Why?

Represent friendship as bidirected edges.

Define bad edge: points to node which does not follow the advisor’s opinion on next day!

Good or bad?
Democrats / Republicans

Eventually each citizen will vote for the same party every other day.

Why?

Represent friendship as bidirected edges.

Define bad edge: points to node which does not follow the advisor’s opinion on next day!
Democrats / Republicans

Eventually each citizen will vote for the same party every other day.

- Consider a citizen c (Democrat) with g good and b bad out-edges on a day t (= will be c resp. not c at t+1)
- Degree of citizen c is hence g+b.
Eventually each citizen will vote for the same party every other day.

- Consider a citizen c (Democrat) with $g$ good and $b$ bad out-edges on a day $t$ (= will be c resp. not c at $t+1$)
- Degree of citizen c is hence $g+b$.

What happens in round $t+1$? How many neighbors root for which party?
Democrats / Republicans

Eventually each citizen will vote for the same party every other day.

- Consider a citizen \(c\) (Democrat) with \(g\) good and \(b\) bad out-edges on a day \(t\) (= will be \(c\) resp. not \(c\) at \(t+1\))
- Degree of citizen \(c\) is hence \(g+b\).

What happens in round \(t+1\)? How many neighbors root for which party?
Democrats / Republicans

Eventually each citizen will vote for the same party every other day.

- Consider a citizen c (Democrat) with g good and b bad out-edges on a day t (= will be c resp. not c at t+1)
- Degree of citizen c is hence g+b.

On day t+1, g friends of c root for the Democrats, and b friends root for the Republicans. And in evening of t+1, c will receives g recommendations for Democrats, and b for Republicans.
Democrats / Republicans

Eventually each citizen will vote for the same party every other day.

- Consider a citizen c (Democrat) with g good and b bad out-edges on a day t (= will be c

Degree of citizen c is hence g+b.

What happens in round t+2?

On day t+1, g friends of c root for the Democrats, and b friends root for the Republicans. And in evening of t+1, c will receives g recommendations for Democrats, and b for Republicans.
Democrats / Republicans

Eventually each citizen will vote for the same party every other day.

If $g > b$ at $t$, then $v$ will vote in time $t+2$ as in time $t$ again, otherwise the opposite!

What happens in round $t+2$?

On day $t+1$, $g$ friends of $c$ root for the Democrats, and $b$ friends root for the Republicans. And in evening of $t+1$, $c$ will receives $g$ recommendations for Democrats, and $b$ for Republicans.
Democrats / Republicans

Eventually each citizen will vote for the same party every other day.

Consider a citizen c (Democrat) with g good and b bad out-edges on a day t (= will be c resp. not c at t+1)

Degree of citizen c is hence g + b.

On day t+1, g friends of c root for the Democrats, and b friends root for the Republicans. And in evening of t+1, c will receives g recommendations for Democrats, and b for Republicans.

What happens in round t+2?

If g > b at t, then c will vote in time t+2 as in time t again, otherwise the opposite!

In other words, if it was g < b, the number of bad edges incident will reduce at time t+2: c changes the party!
Example:

\( g > b: \) stay in same party

The number of bad edges stays the same!

\( b > g: \) change to opposite party

The number of bad edges decreases!
Example:

$g > b$: stay in same party

$\text{The number of bad edges stays the same!}$

$\text{The number of bad edges does never increase. Thus, it will at some point converge to a certain value. From then on, user will vote for the same party every second day. A complex “convergence”!}$

$b > g$: change to opposite party

$\text{The number of bad edges decreases!}$
Related to Conway’s Game of Life

- Turing-complete game: LIFE
- 2d cell grid, each cell dead or alive
- Every cell interacts with its eight neighbors:
  - Any live cell with fewer than two live neighbors dies (loneliness).
  - Any live cell with more than three live neighbors dies, as if by overcrowding.
  - Any live cell with >2 live neighbors lives on to the next generation.

Can model complex behavior: gun + glider:
End of Lecture
Democrats / Republicans

Eventually each citizen will vote for the same party every other day.

Continued…: if \( g > b \)
- At day \( t+1 \), (blue) citizen \( c \):
  - \( g > b \) neighbors blue, \( b \) red
- So citizen \( c \) will be blue (still/again) at \( t+2 \)
- So \( b \) (red) neighbors pointing to \( c \) are bad at \( t+1 \) (from neighbor’s perspective), since \( c \) will be blue at \( t+2 \).

Bad out-edges of \( c \) at time \( t \) will be bad edges to \( c \) at time \( t+1 \)! Total number of bad edges remains the same. (No matter what color of \( c \) is at time \( t+1 \).)
Democrats / Republicans

Eventually each citizen will vote for the same party every other day.

Continued…: if $b > g$

- At day $t+1$, (blue) citizen $c$:
  - $b > g$ neighbors blue, $g$ red

- So citizen $c$ will be red at $t+2$

- So $g < b$ (blue) neighbors pointing to $c$ are bad at $t+1$ (from neighbor’s perspective), since $c$ will be red at $t+2$.

Bad out-edges of $c$ at time $t$ will be good edges to $c$ at time $t+1$! Total number of bad edges decreases. (No matter what color of $c$ is at time $t+1$.)
Democrats / Republicans

Eventually each citizen will vote for the same party every other day.

Continued:

- In both cases, the number of bad edges does not increase.

- In fact, it decreases if any node switches the party.

- Since the number of bad edges cannot be negative, the system will stabilize for a certain number of bad edges.

- Once number of bad edges stabilized, each node either stabilizes to a party or switches back and forth between times t and t+2.

QED
Discussion

- How to do it for randomized algorithms?
  - Do not know k, the number of rounds!
  - But can just simulate more rounds, no problem.
  - Careful about adversary: should not compromise randomness of choices (e.g., have nodes produce random bits until it’s what he wanted)
  - Problem: can also not just stick to given random choices once and forever! Adversary may have corrupted the variables before.

- Some additional memory overhead, but usually bearable.
  - Memory overhead depends on k, the number of rounds, which is low.

- Good for mobile environments: if k-neighborhood does not change, nothing changes