On the Time Complexity of Distributed Topological Self-Stabilization

Andrea Richa

Joint work with Dominik Gall, Riko Jacob, Christian Scheideler, Stefan Schmid, and Hanjo Täubig

Arizona State University

LATIN 2010

Reminder: Purpose of a Model

A model should reflect reality

- accurately enough
- simply enough

to say something interesting.

The execution time of a parallel/distributed algorithm heavily depends on how parallelism is modeled:

- too loose: too many operations (some possibly conflicting) executed in one round
- too rigid: may “force” a sequential execution of the operations when parallelism could still be exploited

We present an execution framework for local topological algorithms, which sheds new light on the *achievable paralellism*

Reminder: Purpose of a Model

A model should reflect reality

- accurately enough
- simply enough

to say something interesting.

The execution time of a parallel/distributed algorithm heavily depends on how parallelism is modeled:

- too loose: too many operations (some possibly conflicting) executed in one round
- too rigid: may "force" a sequential execution of the operations when parallelism could still be exploited

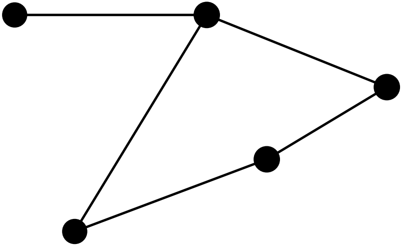
We present an execution framework for local topological algorithms, which sheds new light on the *achievable paralellism*

Topological (Self-Stabilizing) Algorithm

System State
Graph G on n nodes

Local actions
change edges between neighbors

Goal
Achieve a specific target topology

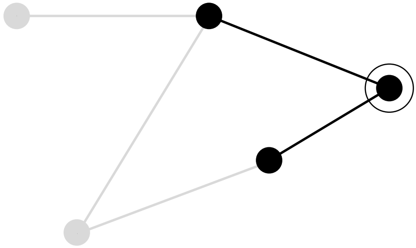


Topological (Self-Stabilizing) Algorithm

System State
Graph G on n nodes

Local actions
change edges between neighbors

Goal
Achieve a specific target topology



Topological (Self-Stabilizing) Algorithm

System State

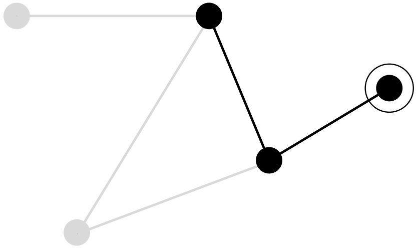
Graph G on n nodes

Local actions

change edges between neighbors

Goal

Achieve a specific target topology

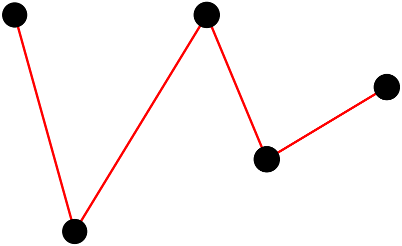


Topological (Self-Stabilizing) Algorithm

System State
Graph G on n nodes

Local actions
change edges between neighbors

Goal
Achieve a specific target topology



Topological (Self-Stabilizing) Algorithm

System State

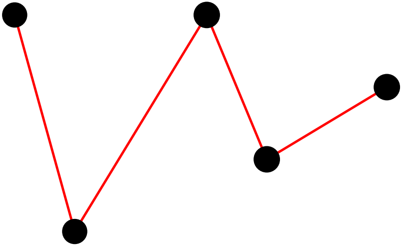
Graph *G* on *n* nodes

Local actions

change edges between neighbors

Goal

Achieve a specific target topology **FAST**



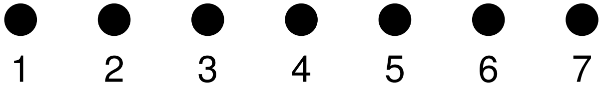
Outline

Linearization

Problem

Initial State Arbitrary connected graph on $\{1, \ldots, n\}$

Goal Edges of the form $\{i, i+1\}$

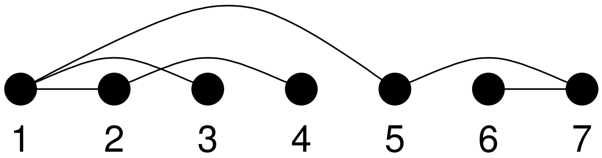


Linearization

Problem

Initial State Arbitrary connected graph on $\{1, \ldots, n\}$

Goal Edges of the form $\{i, i+1\}$

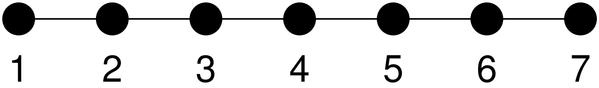


Linearization

Problem

Initial State Arbitrary connected graph on $\{1, \ldots, n\}$

Goal Edges of the form $\{i, i+1\}$



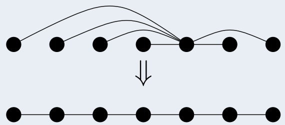
Linearization

Problem

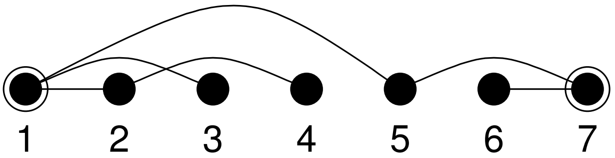
Initial State Arbitrary connected graph on $\{1, \ldots, n\}$

Goal Edges of the form $\{i, i+1\}$

Rule



$t = 0$



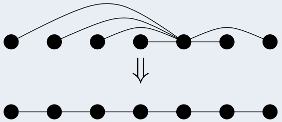
Linearization

Problem

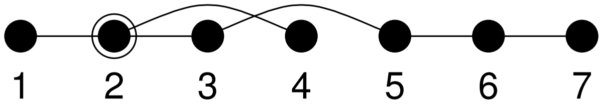
Initial State Arbitrary connected graph on $\{1, \ldots, n\}$

Goal Edges of the form $\{i, i+1\}$

Rule



$t = 1$



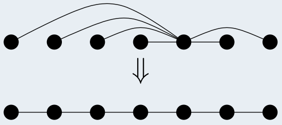
Linearization

Problem

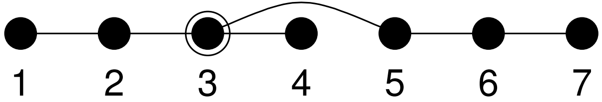
Initial State Arbitrary connected graph on $\{1, \ldots, n\}$

Goal Edges of the form $\{i, i+1\}$

Rule



$t = 2$



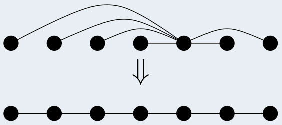
Linearization

Problem

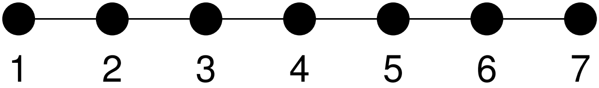
Initial State Arbitrary connected graph on $\{1, \ldots, n\}$

Goal Edges of the form $\{i, i+1\}$

Rule



$t = 3$



Known result

Theorem [Onus, Richa, Scheideler '06]

The presented linearization algorithm takes $\Theta(n)$ rounds

Lower bound

Upper bound

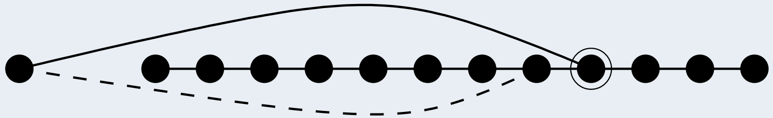
For every missing edge $\{k, k+1\}$:
Consider the length of the shortest interval $[i, j]$ s.t. k and $k+1$ are connected in the subgraph induced by $\{i, \ldots, k, \ldots, j\}$.
This length is reduced in every step by at least one.

Known result

Theorem [Onus, Richa, Scheideler '06]

The presented linearization algorithm takes $\Theta(n)$ rounds

Lower bound



Upper bound

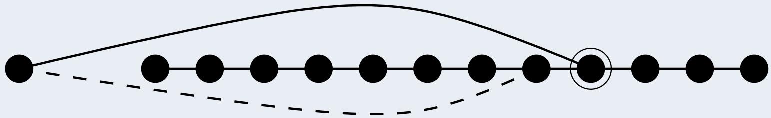
For every missing edge $\{k, k+1\}$:
Consider the length of the shortest interval $[i, j]$ s.t. k and $k+1$ are connected in the subgraph induced by $\{i, \ldots, k, \ldots, j\}$.
This length is reduced in every step by at least one.

Known result

Theorem [Onus, Richa, Scheideler '06]

The presented linearization algorithm takes $\Theta(n)$ rounds

Lower bound



Upper bound

For every missing edge $\{k, k+1\}$:
Consider the length of the shortest interval $[i, j]$ s.t. k and $k+1$ are connected in the subgraph induced by $\{i, \ldots, k, \ldots, j\}$.
This length is reduced in every step by at least one.

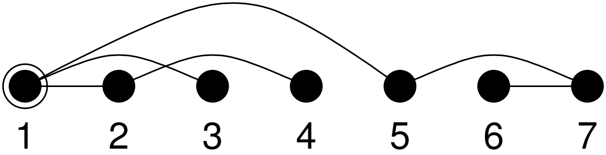
Criticism

High degree nodes

Executing the rule should take time proportional to the degree.

Concurrent access

Nodes should not participate “passively” in more than one execution of the rule.



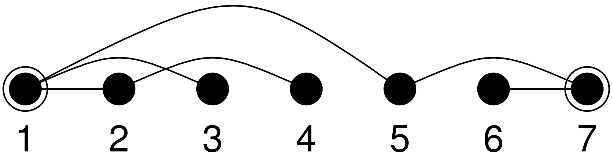
Criticism

High degree nodes

Executing the rule should take time proportional to the degree.

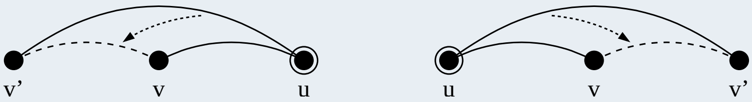
Concurrent access

Nodes should not participate “passively” in more than one execution of the rule.



Atomic Rule

Atomic algorithmic rule



Every such triple is considered independently

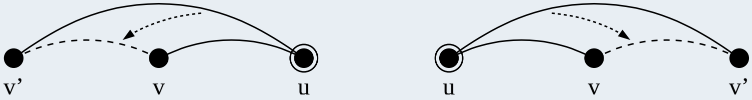
The previous algorithm: Example with 3 triples

Idea: Scheduler

In every round, a scheduler decides on the executed triples.
For example, an independent set of rules (matching)

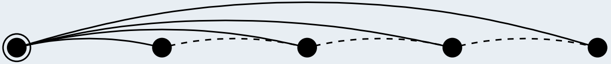
Atomic Rule

Atomic algorithmic rule



Every such triple is considered independently

The previous algorithm: Example with 3 triples

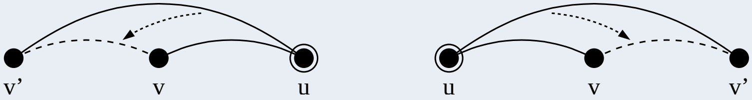


Idea: Scheduler

In every round, a scheduler decides on the executed triples.
For example, an independent set of rules (matching)

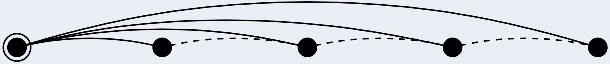
Atomic Rule

Atomic algorithmic rule



Every such triple is considered independently

The previous algorithm: Example with 3 triples

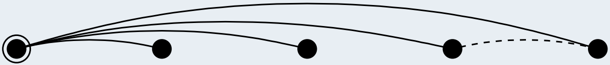


Idea: Scheduler

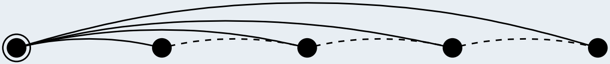
In every round, a scheduler decides on the executed triples.
For example, an independent set of rules (matching)

Variants of the Algorithm

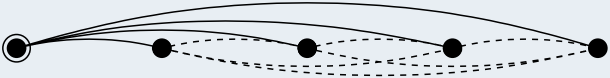
Only longest: LinMax



All neighbors: LinNgb



All possible triples: LinAll



Framework for different models of execution

Rounds

Execution progresses in synchronous rounds;
these are counted

Scheduler

In every round, the set of active rules is determined, and a scheduler decides which rules are executed in this round

Matching

The rules define hyperedges on 3 nodes.
A matching is a set of hyperedges that do not share nodes.

Framework for different models of execution

Rounds

Execution progresses in synchronous rounds;
these are counted

Scheduler

In every round, the set of active rules is determined, and a scheduler decides which rules are executed in this round

Matching

The rules define hyperedges on 3 nodes.
A matching is a set of hyperedges that do not share nodes.

Framework for different models of execution

Rounds

Execution progresses in synchronous rounds;
these are counted

Scheduler

In every round, the set of active rules is determined, and a scheduler decides which rules are executed in this round

Matching

The rules define hyperedges on 3 nodes.
A matching is a set of hyperedges that do not share nodes.

Schedulers: What happens in one round?

Best Case Scheduler

Algorithm chooses a matching

Realistic parallelism, but centrally coordinated

Randomized Scheduler

Takes random maximal matching

Worst Case Scheduler

An adversary chooses dominating set of actions

Very realistic, even if not synchronized

All rules [Onus, Richa, Scheideler '06]

Fire all allowed actions; insertion is stronger than deletion

Unrealistic, easy to analyze.

Antichain [Critical Path, Blumofe, Leiserson '93]

For some fixed serial execution, the next antichain

Established, but actually not really parallel

Schedulers: What happens in one round?

Best Case Scheduler

Algorithm chooses a matching

Realistic parallelism, but centrally coordinated

Randomized Scheduler

Takes random maximal matching

Worst Case Scheduler

An adversary chooses dominating set of actions

Very realistic, even if not synchronized

All rules [Onus, Richa, Scheideler '06]

Fire all allowed actions; insertion is stronger than deletion

Unrealistic, easy to analyze.

Antichain [Critical Path, Blumofe, Leiserson '93]

For some fixed serial execution, the next antichain

Established, but actually not really parallel

Schedulers: What happens in one round?

Best Case Scheduler

Algorithm chooses a matching

Realistic parallelism, but centrally coordinated

Randomized Scheduler

Takes random maximal matching

Worst Case Scheduler

An adversary chooses dominating set of actions

Very realistic, even if not synchronized

All rules [Onus, Richa, Scheideler '06]

Fire all allowed actions; insertion is stronger than deletion

Unrealistic, easy to analyze.

Antichain [Critical Path, Blumofe, Leiserson '93]

For some fixed serial execution, the next antichain

Established, but actually not really parallel

Schedulers: What happens in one round?

Best Case Scheduler

Algorithm chooses a matching

Realistic parallelism, but centrally coordinated

Randomized Scheduler

Takes random maximal matching

Worst Case Scheduler

An adversary chooses dominating set of actions

Very realistic, even if not synchronized

All rules [Onus, Richa, Scheideler '06]

Fire all allowed actions; insertion is stronger than deletion

Unrealistic, easy to analyze.

Antichain [Critical Path, Blumofe, Leiserson '93]

For some fixed serial execution, the next antichain

Established, but actually not really parallel

Schedulers: What happens in one round?

Best Case Scheduler

Algorithm chooses a matching

Realistic parallelism, but centrally coordinated

Randomized Scheduler

Takes random maximal matching

Worst Case Scheduler

An adversary chooses dominating set of actions

Very realistic, even if not synchronized

All rules [Onus, Richa, Scheideler ’06]

Fire all allowed actions; insertion is stronger than deletion

---

Unrealistic, easy to analyze.

Antichain [Critical Path, Blumofe, Leiserson ’93]

For some fixed serial execution, the next antichain

Established, but actually not really parallel

Algorithm LinAll with best case scheduler

Theorem

There exists a scheduler such that for any connected initial graph G_0 the algorithm LinAll converges in $O(n \log n)$ steps.

Proof

Define potential $\sum_{(i,j)\in E} |j - i|$
Scheduler: choose matching greedily according to potential ("longest and second longest edge on highest degree node").
One matching reduces potential by factor $1 - 1/\Theta(n)$.
Round k can happen if $n^3(1 - 1/cn)^k \geq n - 1$; solve for k

Summary of Linearization Results

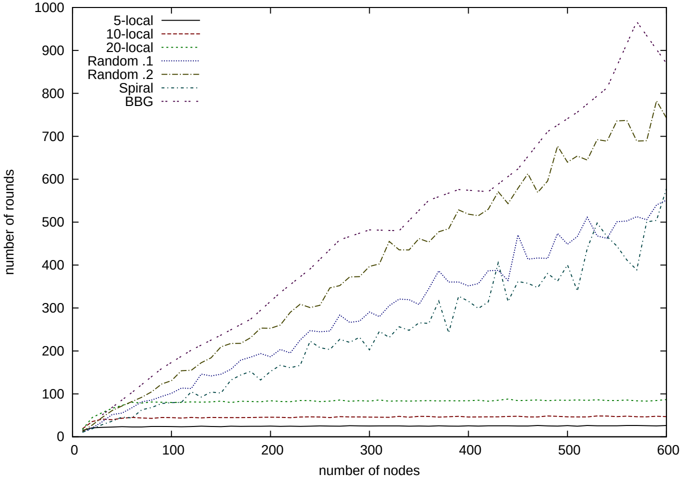
Summary

algorithm	scheduler	time	work
*	*	$\Omega(n)$	$\Omega(n)$
LinNgb	all	$O(n)$	$O(n^2)$
LinAll	best-case	$O(n \log n)$	$O(n^2 \log n)$
LinAll	worst-case	$O(n^2 \log n)$	$O(n^3)$
LinAll	critical-path	$\Theta(n^3)$	$\Theta(n^3)$
LinMax	worst-case	$\Theta(n^2)$	$\Theta(n^2)$
LinMax	critical-path	$O(n^2)$	$O(n^2)$

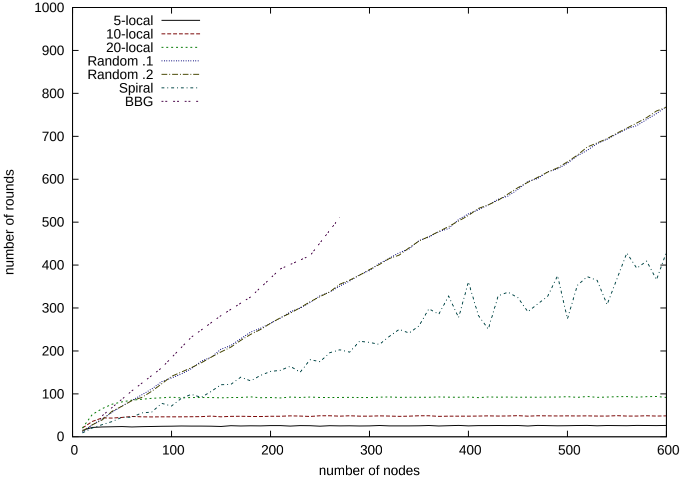
Open Questions

LinMax or LinAll with best-case or randomized scheduler $\Theta(n)$?

Experimental Results: LinAll, Randomized Scheduler



Experimental Results: LinMax, Randomized Scheduler



More future work

- Can we devise local, distributed algorithms that implement (approximations of) the proposed schedulers?

Thank you!
Questions?