Towards *Terminator 2*:

**Self-stabilizing and Distributed Topological Graph Linearization**

Stefan Schmid

*Joint work with:*
- Dominik Gall
- Riko Jacob
- Andrea Richa
- Christian Scheideler
- Hanjo Täubig

**Wroclaw Information Technology Initiative (2008)**
Goal: *Terminator 2!*
Towards Terminator 2:

**Self-stabilizing and Distributed Topological Graph Linearization**

Stefan Schmid

*Joint work with:*

Dominik Gall
Riko Jacob
Andrea Richa
Christian Scheideler
Hanjo Täubig

Wroclaw Information Technology Initiative (2008)
Self-Stabilization (1)

• Important concept in fault-tolerance

• A self-stabilizing system (eventually) ends up in a correct state...

• ... independently of the initial state.

„All the designs I was familiar with were not self-stabilizing in the sense that when once (erroneously) in an illegitimate state, they could – and usually did! – remain so forever.“

E. W. Dijkstra (1974)
Self-Stabilization (2)

• Model: Adversary can disturb the computations (shared variables in system state) arbitrarily

• Once the changes are over, algorithm converges towards desired state
Towards *Terminator 2:*

Self-stabilizing and Distributed Topological Graph Linearization

Stefan Schmid

*Joint work with:*
Dominik Gall
Riko Jacob
Andrea Richa
Christian Scheideler
Hanjo Täubig

Wroclaw Information Technology Initiative (2008)
Graph Linearization

- **INPUT**: Arbitrary connected graph
  - nodes have *arbitrary IDs*
Graph Linearization

- OUTPUT: Sorted graph
Towards *Terminator 2*:

Self-stabilizing and **Distributed** Topological Graph Linearization

Stefan Schmid

*Joint work with:*
Dominik Gall
Riko Jacob
Andrea Richa
Christian Scheideler
Hanjo Täubig

Wroclaw Information Technology Initiative (2008)
Every node runs its own program:
- each pair of nodes \((u,v)\) shares a Boolean variable \(e(u,v)\) („edge“)
- program of the node consists of variables and actions
- an action is of the form:
  
  \[
  \text{<name> : <guard> => <commands>}
  \]

  - Guard: predicate over the local and shared variables of node
  - Commands: sequence of commands involving any local or shared variables of the node itself or its neighbors
  - An action is enabled if guard is true
Towards *Terminator 2*: Self-stabilizing and Distributed Topological Graph Linearization

Stefan Schmid

Joint work with:
Dominik Gall
Riko Jacob
Andrea Richa
Christian Scheideler
Hanjo Täubig

Focus on scalability!

Wroclaw Information Technology Initiative (2008)
Scalability

• Self-stabilizing algorithm: terminates \textit{eventually}

• But what about \textit{convergence time}?

• Analysis of synchronous model
  - total number of \textit{rounds} (after adversarial change) = \textit{execution time}

• What can be done in one round?

\begin{quote}
For scalability reasons, a node should not be involved in too many changes per round!
\end{quote}
Talk Outline
1. Two Distributed Algorithms for Graph Linearization
2. Model for Time Complexity of Convergence
3. Analysis and Simulation
4. Conclusion
Two Algorithms
Basic Linearization Step

• A basic linearization step involves a node triple

• Observe: Connectivity is preserved
**LIN\_all and LIN\_max**

- LIN\_all proposes *all possible triples* to the scheduler (for node $u$)

  \[
  \text{linearize left}(v, w) : (v, w \in u.L \land w < v < u) \rightarrow e(u, w) := 0, e(v, w) := 1
  \]

  \[
  \text{linearize right}(v, w) : (v, w \in u.R \land u < v < w) \rightarrow e(u, w) := 0, e(v, w) := 1
  \]

- LIN\_max proposes the *furthest triple* on each side (for node $u$)

  \[
  \text{linearize left}(v, w):
  \quad (v, w \in u.L) \land w < v < u \land \exists x \in u.L \setminus \{w\} : x < v \rightarrow e(u, w) := 0, e(v, w) := 1
  \]

  \[
  \text{linearize right}(v, w):
  \quad (v, w \in u.R) \land u < v < w \land \exists x \in u.R \setminus \{w\} : x > v \rightarrow e(u, w) := 0, e(v, w) := 1
  \]
\textbf{LIN}_{all} and \textbf{LIN}_{max}

- \textbf{LIN}_{all} proposes \textit{all possible triples} to the scheduler (for node $u$)

- \textbf{LIN}_{max} proposes the \textit{furthest triple} on each side (for node $u$)
**LIN\text{all} and LIN\text{max}**

- **LIN\text{all}** proposes *all possible triples* to the scheduler (for node $u$)

- **LIN\text{max}** proposes the *furthest triple* on each side (for node $u$)
LIN\textsubscript{all} and LIN\textsubscript{max}

- **LIN\textsubscript{all}** proposes *all possible triples* to the scheduler (for node $u$)

  ![Diagram showing all possible triples]

- **LIN\textsubscript{max}** proposes the *furthest triple* on each side (for node $u$)

  ![Diagram showing furthest triples]
LIN\textsubscript{all} and LIN\textsubscript{max}

- LIN\textsubscript{all} proposes \textit{all possible triples} to the scheduler (for node $u$)

- LIN\textsubscript{max} proposes the \textit{furthest triple} on each side (for node $u$)
LIN_{all} and LIN_{max}

- LIN_{all} proposes \textit{all possible triples} to the scheduler (for node \( u \))

- LIN_{max} proposes the \textit{furthest triple} on each side (for node \( u \))
Time Complexity Model
A Naïv Model

• There are different models for what can happen in one round!

• For example: Every node can fire one action per round

• Problem: Nodes can be involved in many changes
  - Therefore, this solution does not scale!
We propose the following, scalable model:
- Let $V(A)$ be the nodes involved in an action $A$
- Two actions $A$ and $B$ are independent if $V(A) \cap V(B) = \emptyset$
- Only an independent set of actions is fired per round
Schedulers

- Nodes propose different enabled actions to the scheduler...

- ... – which one to choose?

**Worst-case scheduler**: chooses independent set of enabled actions which maximizes the runtime

**Best-case scheduler**: chooses independent set of enabled actions which minimizes the runtime

**Randomized scheduler**: chooses independent sets at random

**Greedy scheduler**: scheduler gives priority to nodes having a large degree
Analysis
Analysis

- It turns out that already these simple algorithms are **challenging**!

- **Overview** of results:

  **Worst-case scheduler:**
  - \( \text{LIN}_{\text{max}} \) requires \( \Theta(n^2) \) rounds
  - \( \text{LIN}_{\text{all}} \) requires \( O(n^2 \log n) \) rounds

  **Greedy scheduler:**
  - \( \text{LIN}_{\text{all}} \) requires \( O(n \log n) \) rounds

  **Best-case scheduler:**
  - \( \text{LIN}_{\text{max}} \) and \( \text{LIN}_{\text{all}} \) require \( \Theta(n) \) rounds

  **With degree cap (worst-case scheduler):**
  - \( \text{LIN}_{\text{max}} \) requires at most \( O(n^2) \) and \( \text{LIN}_{\text{all}} \) at most \( O(n^3) \) rounds
In Silico Experiments

- In reality, the runtimes are often close to linear (or even constant in „local graphs“ where node i connects to nodes [i-k,i-k+1,...,i-1,i+1,i+2,...i+k])!
- $\text{LIN}_{\text{all}}$ and $\text{LIN}_{\max}$ yield a similar performance

Figure 3: Left: Parallel runtime of $\text{LIN}_{\text{all}}$ for different graphs under $S_{\text{rand}}$: two $k$-local graphs with $k = 5$, $k = 10$ and $k = 20$, two random graphs with $p = .1$ and $p = .2$, a spiral graph and a $n/3$-BBG. Right: Same experiments with $\text{LIN}_{\max}$. 

Stefan Schmid @ Wroclaw, 2008
Degree Evolution

- Maximum and average degree do not increase
- Rather, degrees are reduced quickly

Figure 4: Left: Maximum and average degree during a run of LIN_{all} and LIN_{max} on a random graph with edge probability $p = .1$. Right: The same experiment on a random graph with $p = .2$. 
Degree Cap Phenomenon

- It appears as a **degree cap constraint** can sometimes improve the runtime!
  - too small degree: blocks many options
  - however, small degree also forces execution on „good paths“
A Sample Analysis (1)

Theorem: Under a worst-case scheduler, LIN_{\text{max}} terminates after at most O(n^2) single linearization steps.

Unfortunately, executions can be highly serial and hence the number of linearization steps is asymptotically equivalent to the number of rounds!

Proof.

Consider the potential function

$$\Phi = \sum_{v \in V} [(2\zeta_l(v) - 1) + (2\zeta_r(v) - 1)] = \sum_{v \in V} 2(\zeta_l(v) + \zeta_r(v) - 1)$$

where $\zeta(v)$ is the length of the longest edge out of $v$ to the left and right.
A Sample Analysis (2)

\[
\Phi = \sum_{v \in V} [(2\zeta_l(v) - 1) + (2\zeta_r(v) - 1)] = \sum_{v \in V} 2(\zeta_l(v) + \zeta_r(v) - 1)
\]

Initially \(\Phi_0 < 2 \, n^2\), as \(\zeta_l(v) + \zeta_r(v) < n\) for each node \(v\).

After round \(i\), the potential is at most \(\Phi_i < 2 \, n^2 - i\).

It most hold for any \(j\) that \(\Phi_j > 0\), otherwise a node would be isolated.

Thus, the claim follows.
A Sample Analysis (3)

$$\Phi = \sum_{v \in V} [(2\zeta_l(v) - 1) + (2\zeta_r(v) - 1)] = \sum_{v \in V} 2(\zeta_l(v) + \zeta_r(v) - 1)$$

Why is $\Phi_i < 2n^2 - i$ true?

Consider a right linearization step:

![Diagram](u(v)w)

**Case 1:** If $\{u,w\}$ is also longest edge of $w$ to the left.

We remove two longest edges of length $|\{u,w\}|$ from $\Phi$.

On the other hand, $u$ may have a new longest edge $\{u,v\}$ to the right, $v$ may have a new longest edge $\{v,w\}$ to the right, and $w$ a new edge of length at most $|\{u,w\}| - 1$ to the left. Since $|\{u,w\}| = |\{u,v\}| + |\{v,w\}|$, it follows that

$$\Delta \Phi \leq (2 \cdot \text{len}(\{u,v\}) - 1) + (2 \cdot \text{len}(\{v,w\}) - 1) + (2(\text{len}(\{u,w\}) - 1) - 1) - (4 \cdot \text{len}(\{u,w\}) - 2) \leq -3$$
A Sample Analysis (4)

\[ \Phi = \sum_{v \in V} [(2\zeta_l(v) - 1) + (2\zeta_r(v) - 1)] = \sum_{v \in V} 2(\zeta_l(v) + \zeta_r(v) - 1) \]

Why is \( \Phi_i < 2n^2 - i \) true?

Consider a right linearization step:

**Case 2:** If \( \{u,w\} \) is not longest edge of \( w \) to the left.

We remove longest edge of length \( |\{u,w\}| \) from \( \Phi \).
On the other hand, \( u \) may have a new longest edge \( \{u,v\} \) to the right, \( v \) may have a new longest edge \( \{v,w\} \) to the right. In this case

\[ \Delta \Phi \leq (2 \cdot \text{len}(\{u,v\}) - 1) + (2 \cdot \text{len}(\{v,w\}) - 1) - (2 \cdot \text{len}(\{u,w\}) - 1) \leq -1 \]

QED
Another Sample Analysis (1)

Theorem: Under a greedy scheduler, LIN\textsubscript{all} terminates after at most $O(n \log n)$ rounds.

Greedy scheduler: In each round, nodes are sorted w.r.t. remaining degree (remove fired triples with incident edges). Scheduler picks node $v$ with largest degree, and schedules triple of $v$ (to the larger degree side) with most distant neighbors.

Proof.

Consider the potential function $\Psi = \sum_{e \in E} \text{len}(e)$

Initially: $\psi_0 < n^3$
In the end: $\psi = n-1$

We will show that in each round, potential $\psi$ is multiplied by a factor of at most $1-1/(24n)$. This implies the claim.
Another Sample Analysis (2)

This implies the claim?

**Lemma 3.4.** Let $\Xi$ be any positive potential function, where $\Xi_0$ is the initial potential value and $\Xi_i$ is the potential after the $i^{\text{th}}$ round of a given algorithm $ALG$. Assume that $\Xi_i \leq \Xi_{i-1} \cdot (1 - 1/f)$ and that $ALG$ terminates if $\Xi_j \leq \Xi_{\text{stop}}$ for some $j \in \mathbb{N}$. Then, the runtime of $ALG$ is at most $O(f \cdot \log (\Xi_0 / \Xi_{\text{stop}}))$ rounds.

**Proof.** From $\Xi_i \leq \Xi_{i-1} \cdot (1 - 1/f)$, it follows that $\Xi_j \leq \Xi_0 \cdot (1 - 1/f)^j$. Now consider $j = f \cdot \ln \frac{\Xi_0}{\Xi_{\text{stop}}}$, which leads to (using $\ln(1 + x) \leq x$ for all $x > -1$)

$$
\Xi_j \leq \Xi_0 \cdot (1 - 1/f)^{f \cdot \ln \frac{\Xi_0}{\Xi_{\text{stop}}}} = \Xi_0 e^{f \cdot \ln \frac{\Xi_0}{\Xi_{\text{stop}}} \cdot (1 - 1/f)} \leq \Xi_0 e^{f \cdot \ln \frac{\Xi_0}{\Xi_{\text{stop}}} \cdot (-1/f)} = \Xi_0 e^{-\ln \frac{\Xi_0}{\Xi_{\text{stop}}}} = \Xi_{\text{stop}}.
$$

$\square$
Another Sample Analysis (3)

Greedy scheduler: In each round, nodes are sorted w.r.t. remaining degree (remove fired triples with incident edges). Scheduler picks node $v$ with largest degree, and schedules triple of $v$ with most distant neighbors (to larger degree side).

Consider the potential function $\Psi = \sum_{e \in E} \text{len}(e)$

We will show that in each round, potential $\psi$ is multiplied by a factor of at most $1 - 1/(24 \cdot n)$. This implies the claim.

- Observe: firing a triple reduces potential $\psi$...
- ... but other nodes will be blocked in this round.

• Idea: we want to estimate the amount of blocked potential.
Another Sample Analysis (4)

• Consider the following right-linearization step

• Removing \{u,w\} and adding \{v,w\} reduces the potential by at least

\[\text{dist}(u,w) - \text{dist}(v,w) = \text{dist}(u,v)\]

• Since the greedy scheduler takes larger degree side:

\[\text{dist}(u,v) \geq \frac{\text{deg}(u)}{2} - 1 \geq \frac{\text{deg}(v)}{4}\]
Another Sample Analysis (5)

- Thus, potential reduced in one step by at least $\text{deg}(u)/4$

- How much potential is blocked?

- Consider remaining components after removing triple
- Consider neighbor $x$ of $u$, $v$ or $w$
  - if $x$ is in ordered line component => blocked potential at most $n+n$
  - if $x$ is in different component => can still be linearized further (account for blocked component's potential later, only count link length potential: $n$)
Another Sample Analysis (6)

- The amount of blocked potential is at most $6 \cdot \text{deg}(u) \cdot n$
  - since $u$ has larger degree than $v$ and $w$,
  - and since we have at most blocked potential $2 \cdot n$ per neighbor ($n$ for component plus $n$ for link to this neighbor)

- Thus, potential reduced by a factor at least $1-\Theta(1/n)$ per round.

QED.
Conclusion
Self-stabilizing Graph Linearization

- Most simple algorithms already have many interesting properties
- The quest for faster algorithms has already started!
- Besides linearization, it will be useful to construct alternative graphs in a self-stabilizing manner

Dziekuje!

Slides and papers at http://www14.informatik.tu-muenchen.de/personen/schmiste/