Making Randomized Algorithms Self-Stabilizing

Volker Turau

26th Int. Colloquium on Structural Information and Communication Complexity

July 2nd, 2019

Institute of Telematics
Hamburg University of Technology

TUHH

1

Introduction

- For many classical problems known distributed algorithms are faster by orders of magnitude than self-stabilizing algorithms
  - Majority of self-stabilizing algorithms has stabilization time of $\Theta(n)$
  - $O(\log n)$ or even $O(\log^* n)$ are common for distributed algorithms

- Question:
  Is it possible to close the performance gap between general distributed algorithms and self-stabilizing algorithms or does there exist an inherent barrier?

State of the Art

- Self-stabilizing algorithms with sublinear run-time
  - Barenboim et al. (2018): $\Delta + 1$ coloring, $2\Delta + 1$ edge-coloring, maximal independent set, maximal matching in $O(\Delta + \log^* n)$ rounds
  - T. (2018): $\Delta + 1$ coloring in $O(\log n)$ rounds w.h.p.

Making Distributed Algorithms Self-Stabilizing

- Program transformation techniques can make local algorithms self-stabilizing (Afek (1997), Awerbuch(1994), Lenzen (2009))
  - Proof labeling schemes, self-stabilizing reset algorithms
- Disadvantage: Overhead in run-time or memory consumption
- Many techniques cannot be applied to randomized algorithms

- Topic of this work:
  How to transform phase-oriented distributed algorithms into self-stabilizing algorithms without overhead?

Contribution

- Randomized self-stabilizing algorithms for maximal independent set and maximal matching stabilizing w.h.p. in $O(\log n)$ rounds in the synchronous model

2

Phase-Oriented Distributed Algorithms

Phase-Oriented Algorithms

- Phase-Oriented algorithms in synchronous model
  - A phase consists of a fixed number of rounds
  - Phases are executed periodically
  - Nodes perform a dedicated task in each round of a phase
- Faults can have devastating consequences
  - Some nodes may be still in the first round of a phase, others already in the second round, etc.
  - In such a scenario, phase-oriented algorithms will produce incorrect results

Self-stabilizing Synchronous Unison

- Implementation of phases in a synchronous system is based on a synchronized counter variable
  - Counter makes nodes round- and phase-aware
  - Self-stabilizing algorithm must handle faults hitting counter
- Thus, phase-oriented self-stabilizing algorithms require a self-stabilizing counter
  - Self-stabilizing synchronous unison
  - Existing algorithms require $\Omega(\mathrm{Diam}(G))$ rounds to stabilize
- To achieve $O(\log n)$ run-time an approach that relinquishes the phase concept is required

Approach

- Each node continuously and independently performs its original actions but not necessarily in the original order
- Thus, nodes execute their actions no longer synchronized but interleaved
- To still converge to a legitimate state, phase-dependent behavior is mapped to a *phase variable*
    - A node can determine from the phase variables of its neighbors its position within a phase and act accordingly
- This way transient errors can be tolerated

Notations & Model

- Synchronous model, locally shared memory
- A distributed algorithm is called *self-stabilizing* if it satisfies
  - *closure property* and
  - *convergence property*
- A randomized algorithm terminates w.h.p. within $O(f(n))$ time if it does so with probability at least $1 - 1/n^c$ for some $c > 1$
- A randomized distributed algorithm is called *self-stabilizing* if it satisfies closure property and w.h.p. the convergence property

3

Algorithm $\mathcal{A}_{\text{MAT}}$

Maximal Matching

- Many self-stabilizing algorithms for maximal matching exist
- The only self-stabilizing algorithm with sublinear time is by Barenboim et al. (2018) $O(\Delta + \log^* n)$ rounds
- Much stronger results for general distributed algorithms: Fischer (2017) proposed an algorithm requiring $O(\log^2(\Delta)\ \log n)$ rounds

- This work:
  We transform a randomized max. mat. algorithm of Israeli & Itai into a self-stabilizing algorithm stabilizing w.h.p. in $O(\log n)$ rounds

Algorithm of Israeli and Itai

- Algorithm uses phases of four rounds

Invite: Each node invites a random neighbor

Accept: Invited nodes randomly accept one invitation

  - Nodes that accepted an invitation or whose invitation was accepted form a subgraph U
  - Connected components of U are paths and cycles

Peer: Each node of U selects either edge towards the accepted or to the accepting neighbor

  - Corresponding edge is called a peer

Match: Edges that were selected by both end-nodes as peers join matching

End nodes of matched edges become passive

Variables used by $\mathcal{A}_{\text{MAT}}$

- *match*$($*false*$)$: Indicates whether node is already matched
- *partner*$($*null*$)$: Either a neighbor or *null*. If *match* $=$ *true* then edge connecting node with *partner* belongs to matching. Otherwise it indicates invitation, acceptance, or peer
- *phase*$($*none*$)$: Semantics of *partner*
    - *invit*: *partner* is invited
    - *accept*: *partner*'s invitation is accepted
    - *peer*: edge connecting node and *partner* is proposed for matching
    - *none*: No *partner* selected, i.e., *partner* $=$ *null*

Algorithm $\mathcal{A}_{\text{MAT}}$

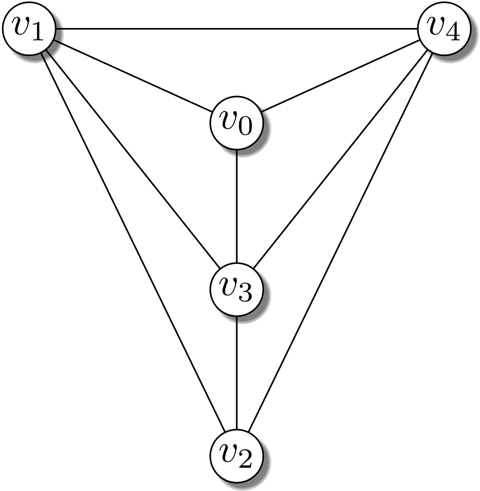
- Locally detected inconsistencies cause a local reset
- Nodes execute **Match, Peer, Accept, Invite** according to their own phase variable and that of their neighbors

- An active node *v* matches its partner if edge to partner is peer for both nodes (**Match**)
- An unmatched active node *v* **randomly** selects an active neighbor *w* satisfying the highest option of
  1. *w* accepted *v*'s invitation or vice versa (**Peer**)
  2. *w* invites *v* (**Accept**)
  3. *w* is not a peer or accepting an invitation (**Invite**)
  4. *null*

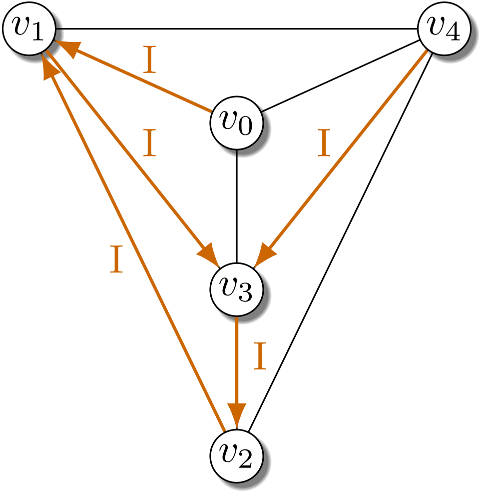
The Three Rules of $\mathcal{A}_{\mathsf{MAT}}$

- RESET: Corrects inconsistent states, assigns fallback values to variables
- MATCH: Promotes nodes with $match = false$ to $match = true$ if conditions are met
- RANDOM: If $match = false$ then update variables *partner* and *phase* as described above

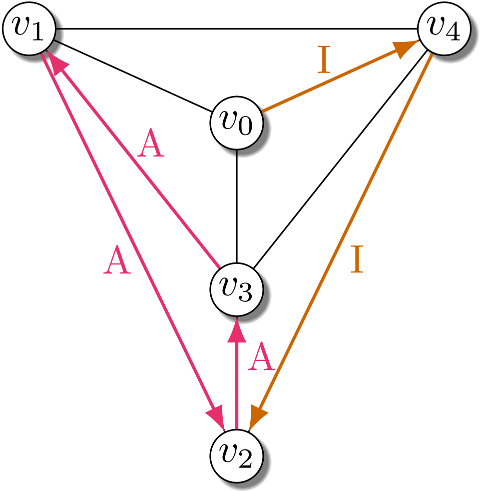
Execution of Algorithm $\mathcal{A}_{\text{MAT}}$



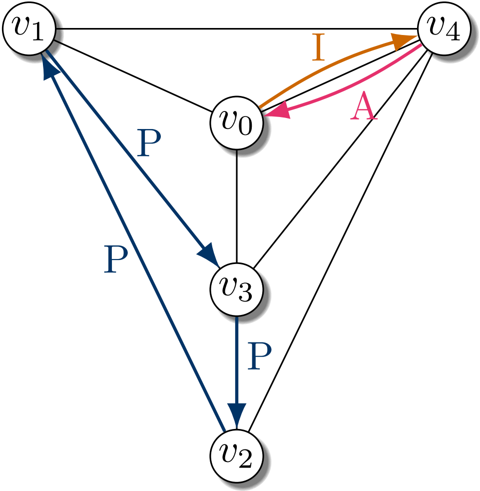
Execution of Algorithm $\mathcal{A}_{\text{MAT}}$



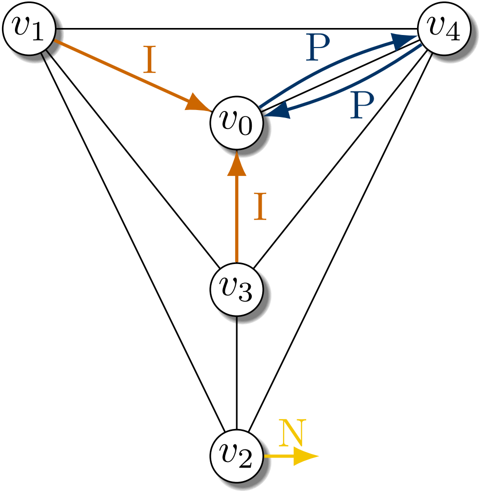
Execution of Algorithm $\mathcal{A}_{\text{MAT}}$



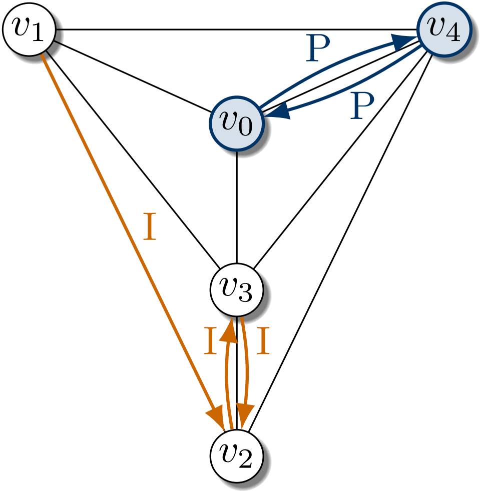
Execution of Algorithm $\mathcal{A}_{\text{MAT}}$



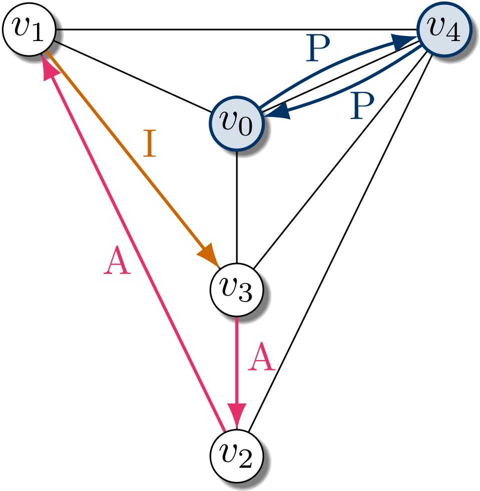
Execution of Algorithm $\mathcal{A}_{\mathsf{MAT}}$



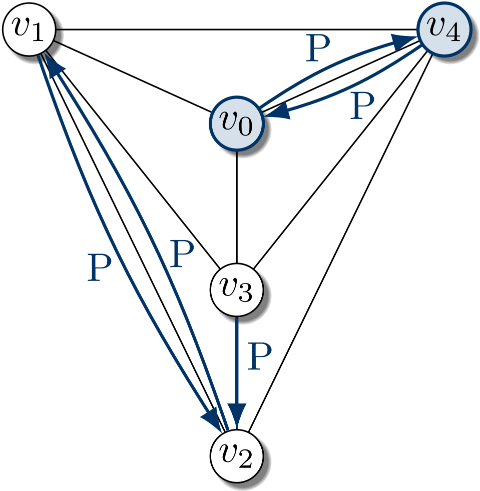
Execution of Algorithm $\mathcal{A}_{\text{MAT}}$



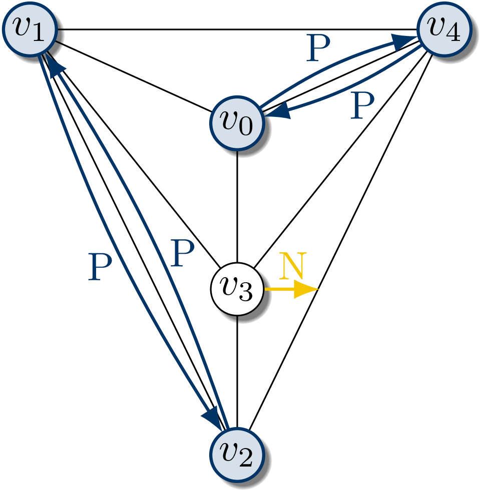
Execution of Algorithm $\mathcal{A}_{\text{MAT}}$



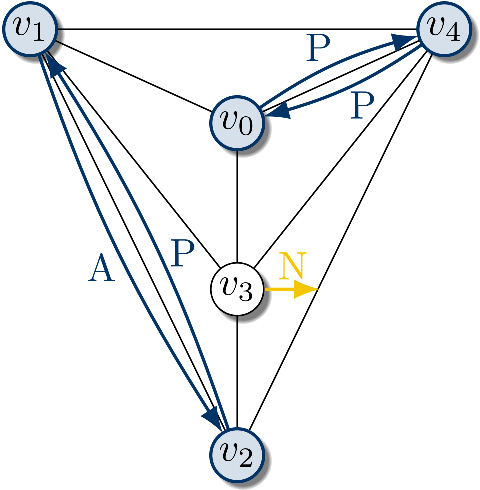
Execution of Algorithm $\mathcal{A}_{\mathsf{MAT}}$



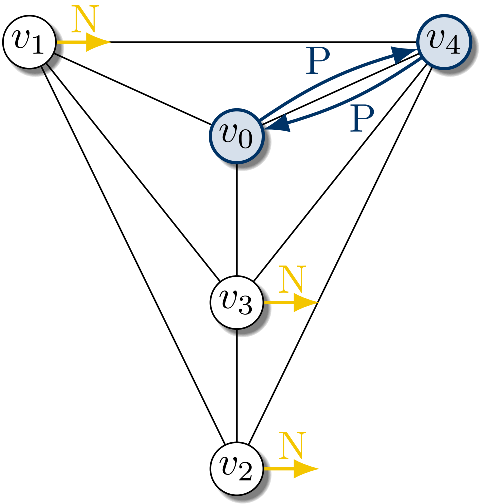
Execution of Algorithm $\mathcal{A}_{\text{MAT}}$



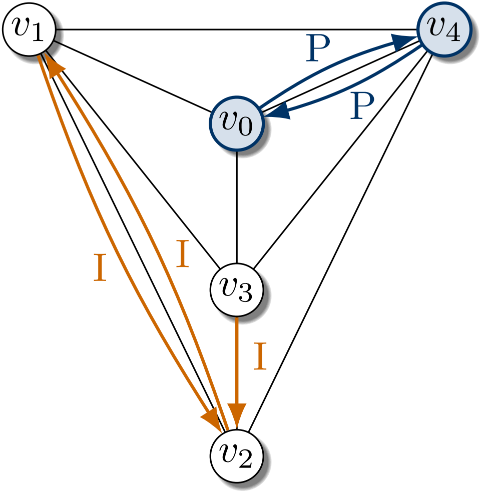
Execution of Algorithm $\mathcal{A}_{\text{MAT}}$



Execution of Algorithm $\mathcal{A}_{\mathsf{MAT}}$



Execution of Algorithm $\mathcal{A}_{\mathsf{MAT}}$



Stabilization Time of $\mathcal{A}_{\text{MAT}}$

- Let G_i be the subgraph of G induced by the unmatched nodes in round i
- $G_i \subseteq G_{i-1}$ for $i > 1$
- A node v of a graph is called *good* if it has many neighbors with smaller degree than itself
  - Idea: Good nodes have a high chance of getting invited

Lemma

Let v be a good node of G_i. The expected number of edges incident to v in G_i not contained in G_{i+4} is at least $(1 - e^{-1/6})d_{G_i}(v)/12$ if $i > 1$.

Stabilization Time of $\mathcal{A}_{\text{MAT}}$

Lemma (Alon et al.)

At least half of the edges of any graph are adjacent to a good node.

This proves that after expected $O(\log n)$ rounds graph G_i consists of isolated nodes only

Apply probabilistic arguments to prove the result

Stabilization Time of $\mathcal{A}_{\text{MAT}}$

Theorem

Algorithm $\mathcal{A}_{\text{MAT}}$ is self-stabilizing and computes w.h.p. in $O(\log n)$ rounds using $O(\log n)$ memory a maximal matching.

- $\mathcal{A}_{\text{MAT}}$ exhibits more concurrency than original algorithm

Algorithm $\mathcal{A}_{\text{MIS}}$

Theorem

Algorithm $\mathcal{A}_{\text{MIS}}$ is self-stabilizing and computes w.h.p. in $O(\log n)$ rounds using $O(\log n)$ memory a maximal independent set.

4

Conclusion

Conclusion & Outlook

- We demonstrated that phase-oriented randomized distributed algorithms can be made self-stabilizing in the synchronous model while retaining their time complexity with almost no overhead
- We transformed two classical distributed randomized graph algorithms into self-stabilizing algorithms
- They outperform existing self-stabilizing algorithms
- Ultimate goal of this work:
  Operationalize this transformation to have a tool that automatically performs transformation for a rich class of randomized algorithms even in asynchronous model

Making Randomized Algorithms Self-Stabilizing

Volker Turau

26th Int. Colloquium on Struc ... plexity

Volker Turau

Professor

Phone +49 / (0)40 428 78 3530

e-Mail turau@tuhh.de

http://www.ti5.tu-harburg.de/staff/turau