Analysis of Link Reversal Routing Algorithms for Mobile Ad Hoc Networks

Seminar of Distributed Computing WS 04/05

ETH Zurich, 1.2.2005

Nicolas Born

nborn@student.ethz.ch

Paper

- Analysis of Link Reversal Routing Algorithms for Mobile Ad Hoc Networks
  Costas Busch, Srikanth Surapaneni, Srikanta Tirthapura; SPAA 2003

Overview

- Link Reversal Routing Algorithms
  - Full Reversal
  - Partial Reversal
- Equivalence of Executions
- Performance Analysis
- Results
- Conclusion

Link Reversal Routing Algorithms

- Introduced by Gafni and Bertsekas (1981)
- Routing in mobile ad hoc networks
- Adaptive, self-stabilizing

- Contribution of the paper: first performance analysis

Model

Link Reversal Routing Algorithms

- Ad-Hoc Network
- Network connectivity is assumed
- Each node has an unique id

- Suited for networks with “average mobility”

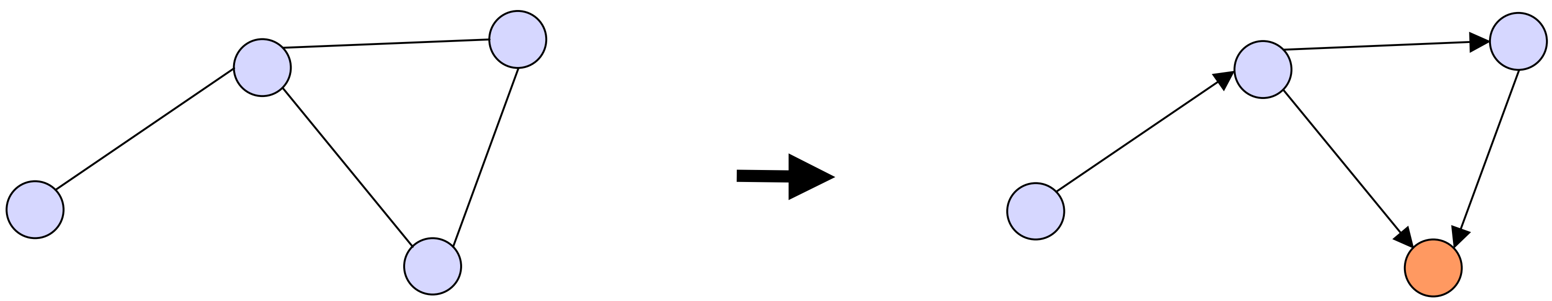
Shortest Path — Link Reversal — Flooding

Fixed Network ⟷ High Mobility

Underlying Communication Graph

Link Reversal Routing Algorithms

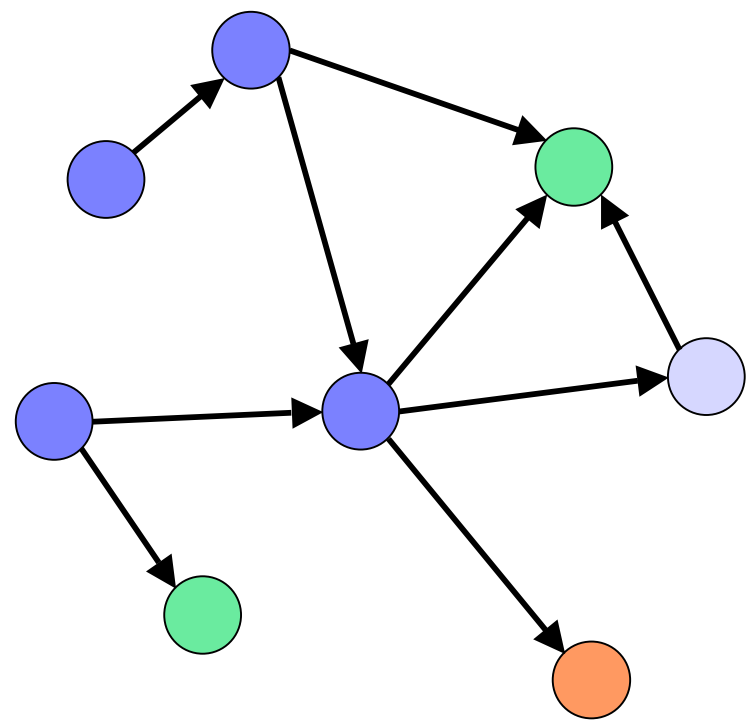
- Convert the ad-hoc network to a destination oriented graph



Notation

Link Reversal Routing Algorithms

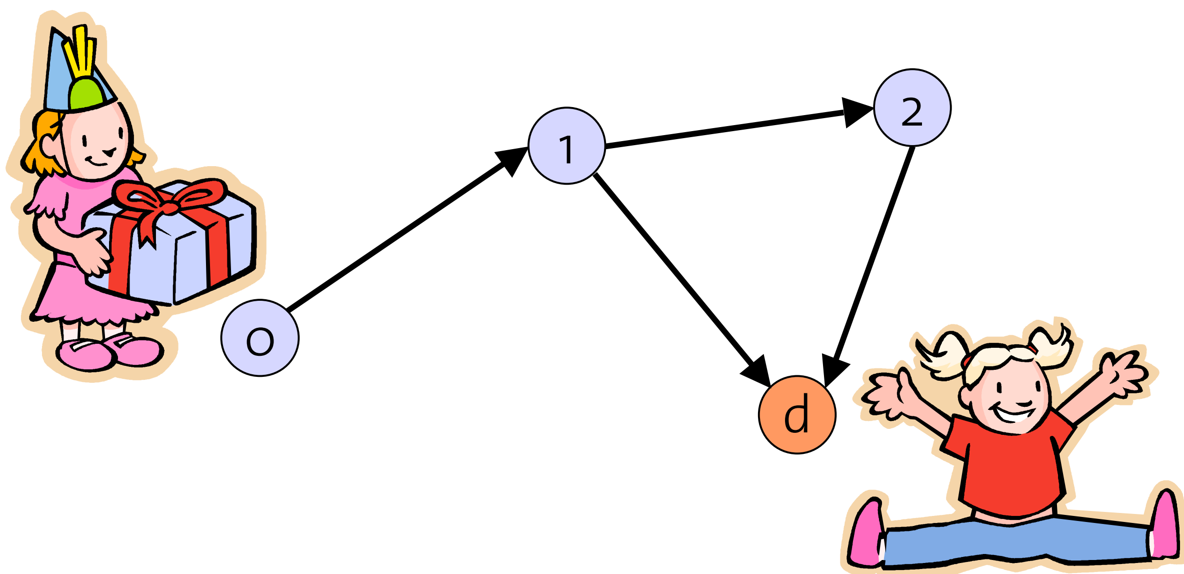
- Destination
- **Good nodes:** nodes with at least one directed path to the destination
- **Bad nodes:** nodes with no directed path to the destination
- **Sinks:** nodes with only incoming links



Routing

Link Reversal Routing Algorithms

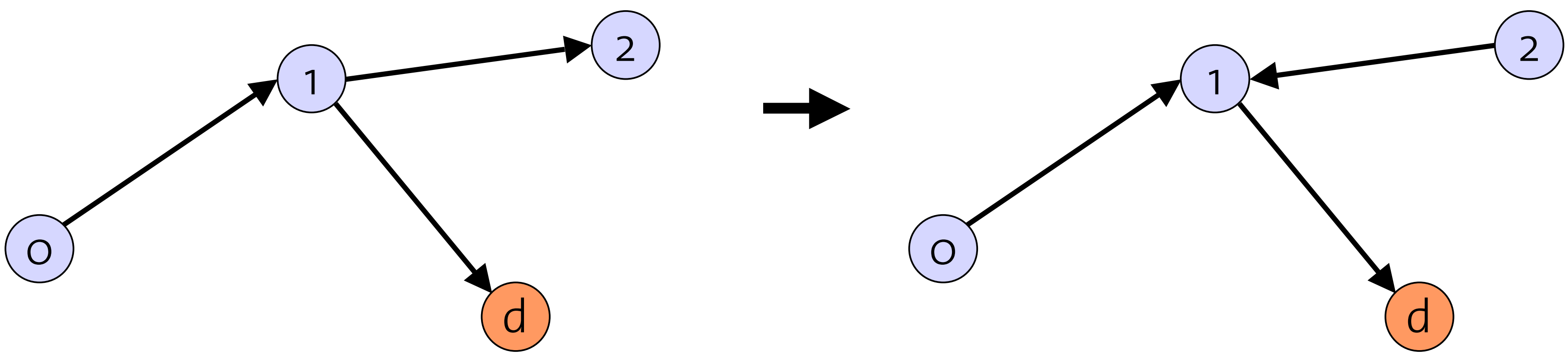
- When a node receives a packet, it forwards the packet on any outgoing link. The packet will eventually reach the destination.



Route Maintenance

Link Reversal Routing Algorithms

- If a node loses its route to the destination, the algorithm reacts by performing link reversals.
- Node finds out that it has become a sink -> it reverses the directions of some or all incoming links.



Work and Time

Link Reversal Routing Algorithms

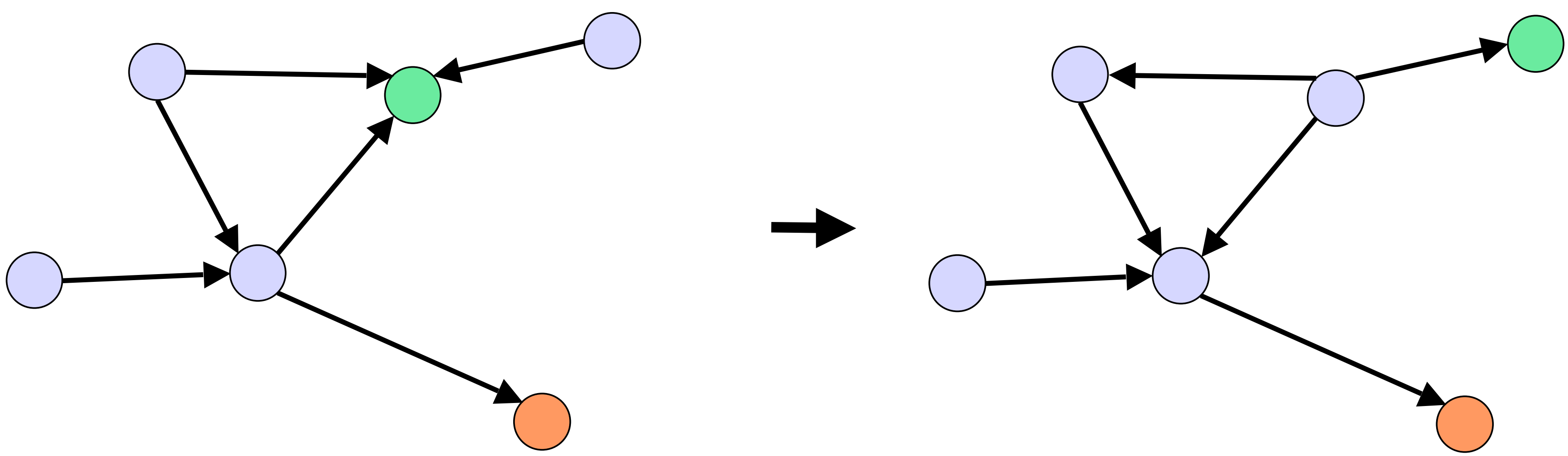
- Work: number of reversals until stabilization.
- Time: number of parallel time steps until stabilization.

Overview

- Link Reversal Routing Algorithms
  - Full Reversal
  - Partial Reversal
- Equivalence of Executions
- Performance Analysis
- Results
- Conclusion

Full Reversal Algorithm

- When a node becomes a sink, it reverses the directions of all its links.

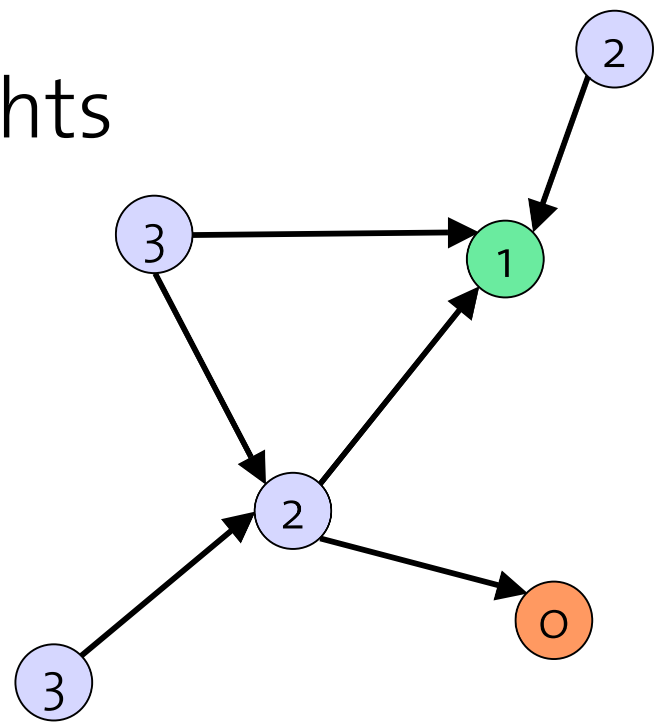


Implementation

Full Reversal Algorithm

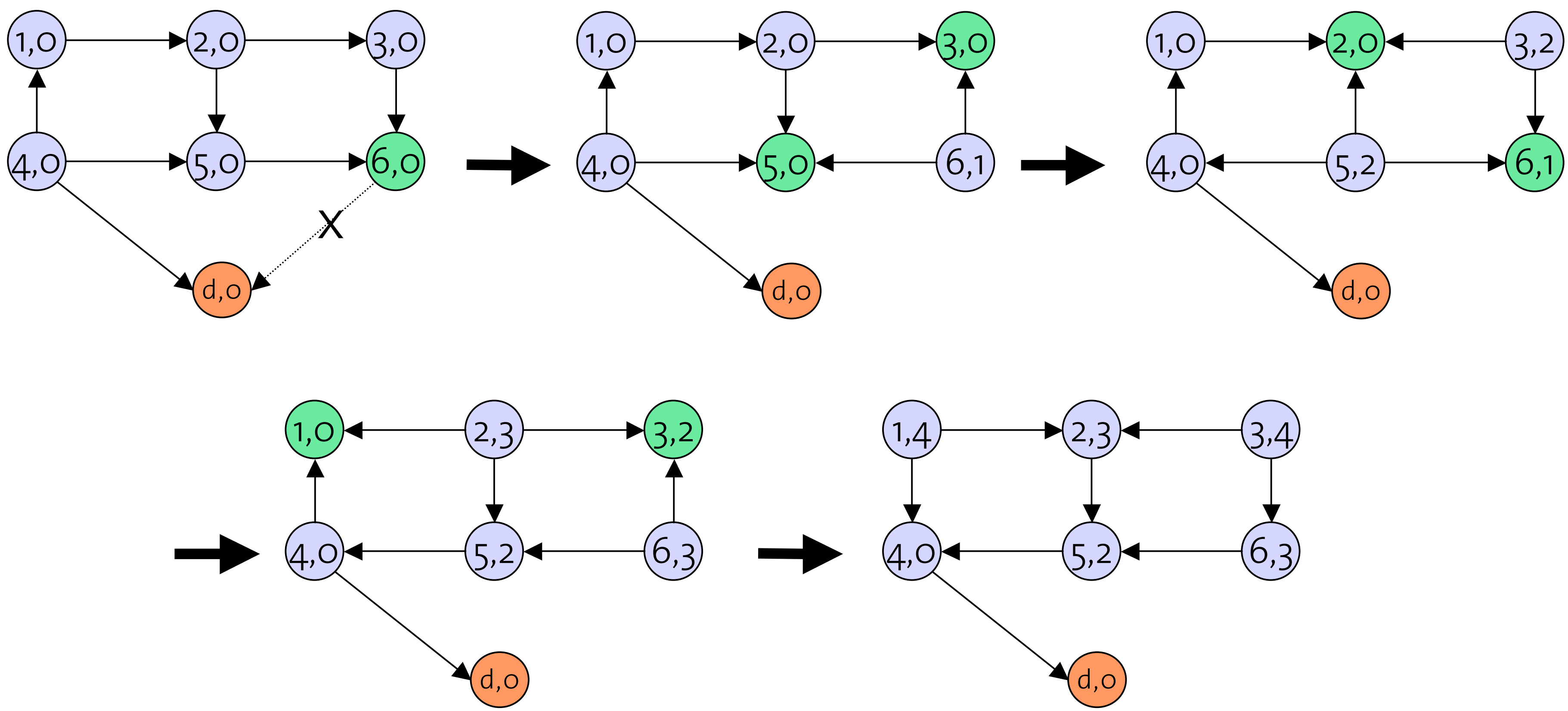
- Idea: analogy to a river. Water flows from bigger height to lower height.

- => Implemented with heights
  - Height of node v_i: h_i
  - $h_d = 0$
  - N_i: neighborhood of v_i
  - Height of v_i after reversal: $\max\{ h_j \mid v_j \in N_i \} + 1$



Example

Full Reversal Algorithm



Node that reverses

Reversals: 7
Time: 4

Overview

- Link Reversal Routing Algorithms
  - Full Reversal
  - Partial Reversal
- Equivalence of Executions
- Performance Analysis
- Results
- Conclusion

Partial Reversal Algorithm

- If a node *v* becomes a sink, it reverses the links to those neighbors that have not reversed their links into *v*.
- If every neighbor node has a reversed link to *v*, it reverses every link.

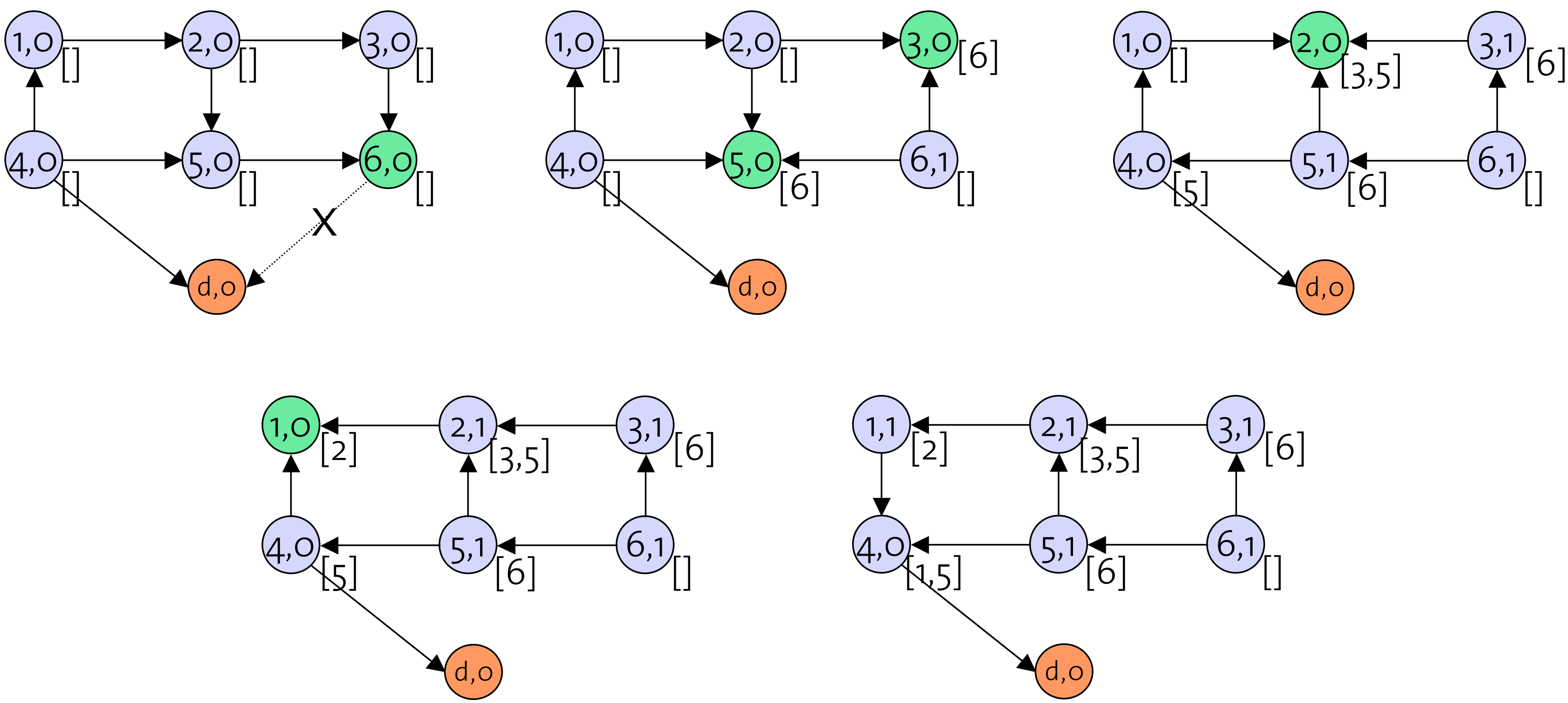
Implementation

Partial Reversal Algorithm

- Also implemented using heights
  - Height of node v_i: h_i
  - $h_d = 0$
  - Height of v_i after reversal: $\min\{ h_j \mid v_j \in N_i \} + 1$
- Every node *v* keeps a list of its neighboring nodes that have reversed their links into *v*.

Example

Partial Reversal Algorithm



Node that reverses

Reversals: 5
Time: 4

Overview

- Link Reversal Routing Algorithms
  - Full Reversal
  - Partial Reversal
- Equivalence of Executions
- Performance Analysis
- Results
- Conclusion

Equivalence of Executions

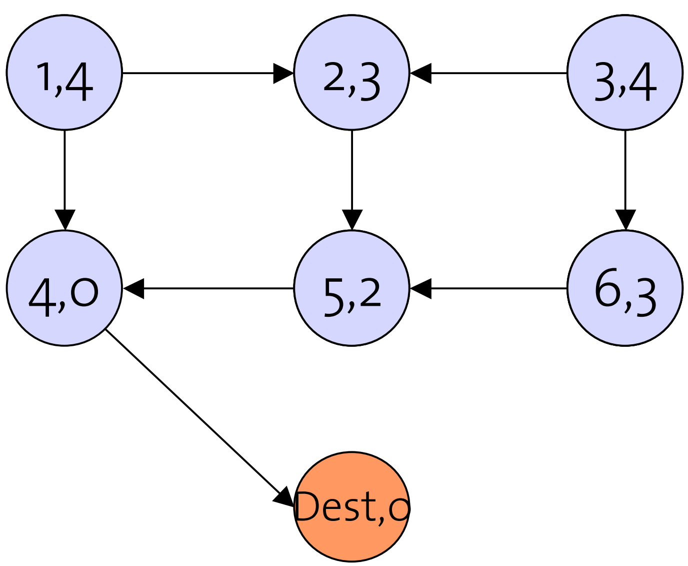
- There are many different reversal schedules.

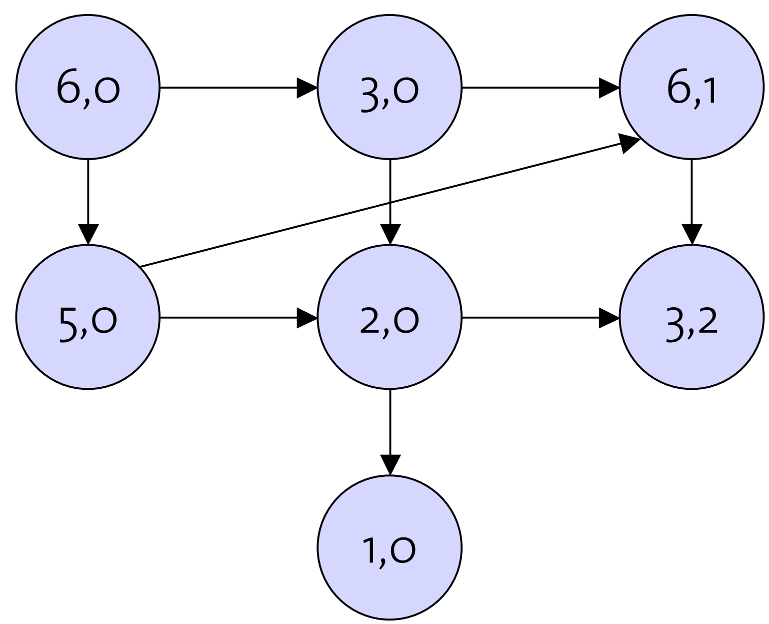
- **Goal:** show that any two executions of a deterministic reversal algorithm starting from the same initial state are equivalent.

Dependency Graph

Equivalence of Executions

- Execution $R=r_1,\ldots,r_k$
- Directed edge from r_i to r_j, iff
  - v_i is neighbor of v_j
  - r_j is first reversal of v_j after r_i in execution R





Dependency Graph

Main Theorem
Equivalence of Executions

- Two executions are equivalent, if they have the same dependency graph.

- **Theorem:** Any two executions of a deterministic reversal algorithm starting from the same initial state are equivalent.

Conclusions
Equivalence of Executions

- For all executions of a deterministic reversal algorithm starting from the same initial state:
  - Final state is the same
  - Number of reversals of each node is the same
- The depth of the dependency graph is a lower bound for the time complexity of execution of a deterministic reversal algorithm.

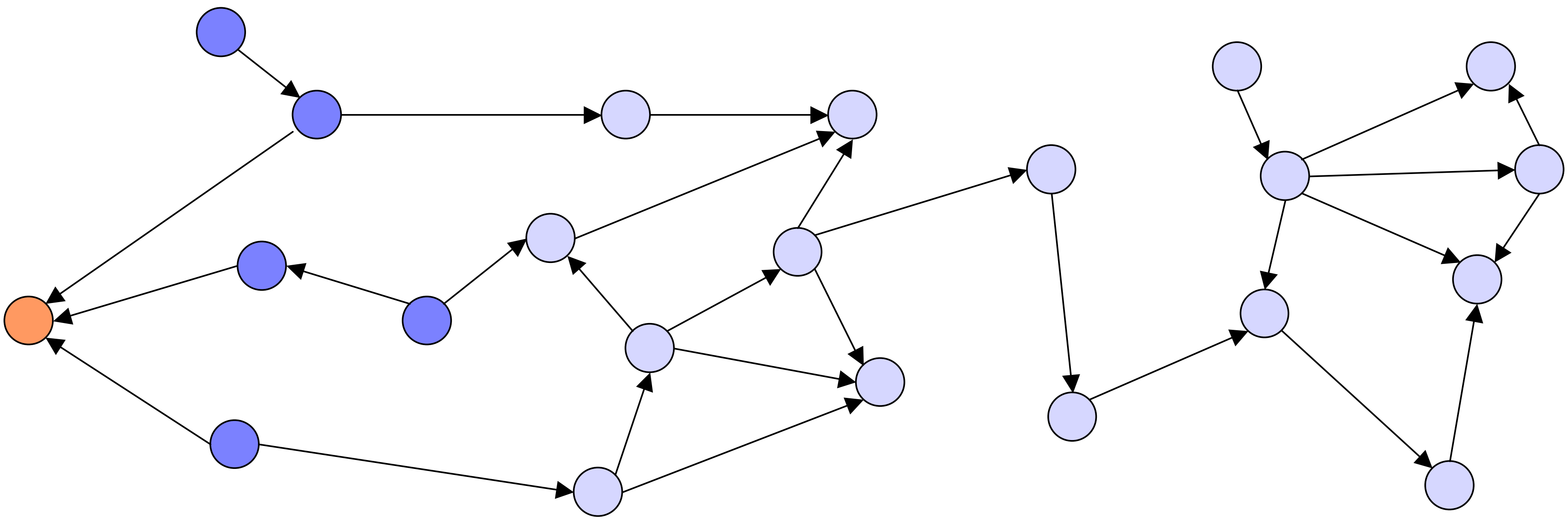
Overview

- Link Reversal Routing Algorithms
  - Full Reversal
  - Partial Reversal
- Equivalence of Executions
- Performance Analysis
- Results
- Conclusion

Full Reversal Algorithm

Performance Analysis

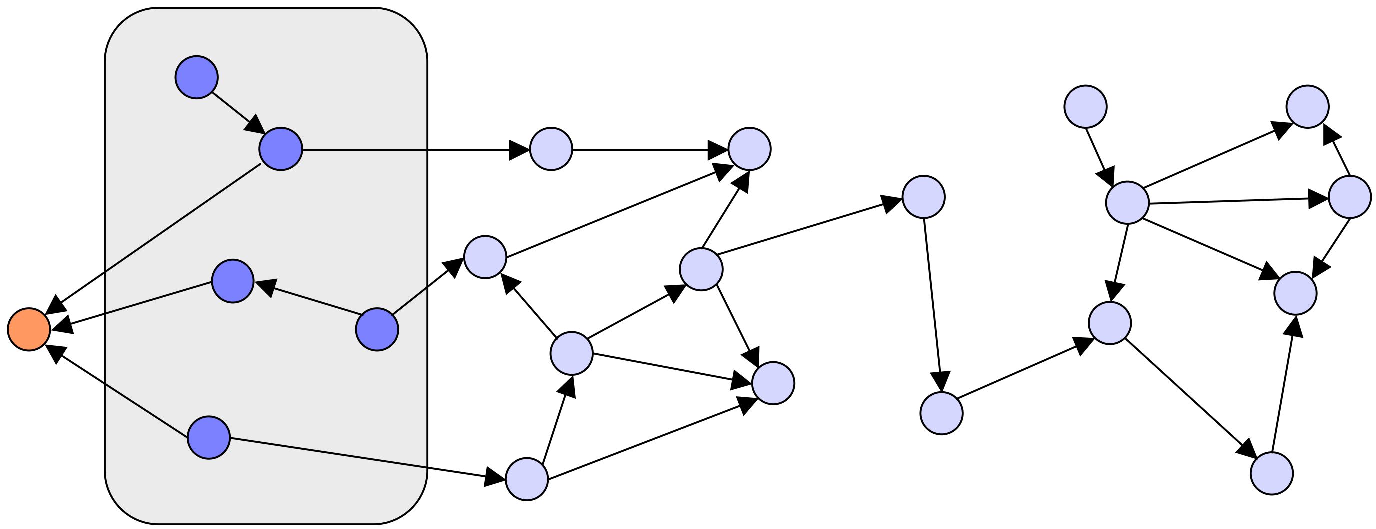
- **Goal:** lower and upper bound on the performance of the full reversal algorithm



Question

Full Reversal Algorithm

- For any reversal algorithm starting from any initial state, a good node never reverses till stabilization.

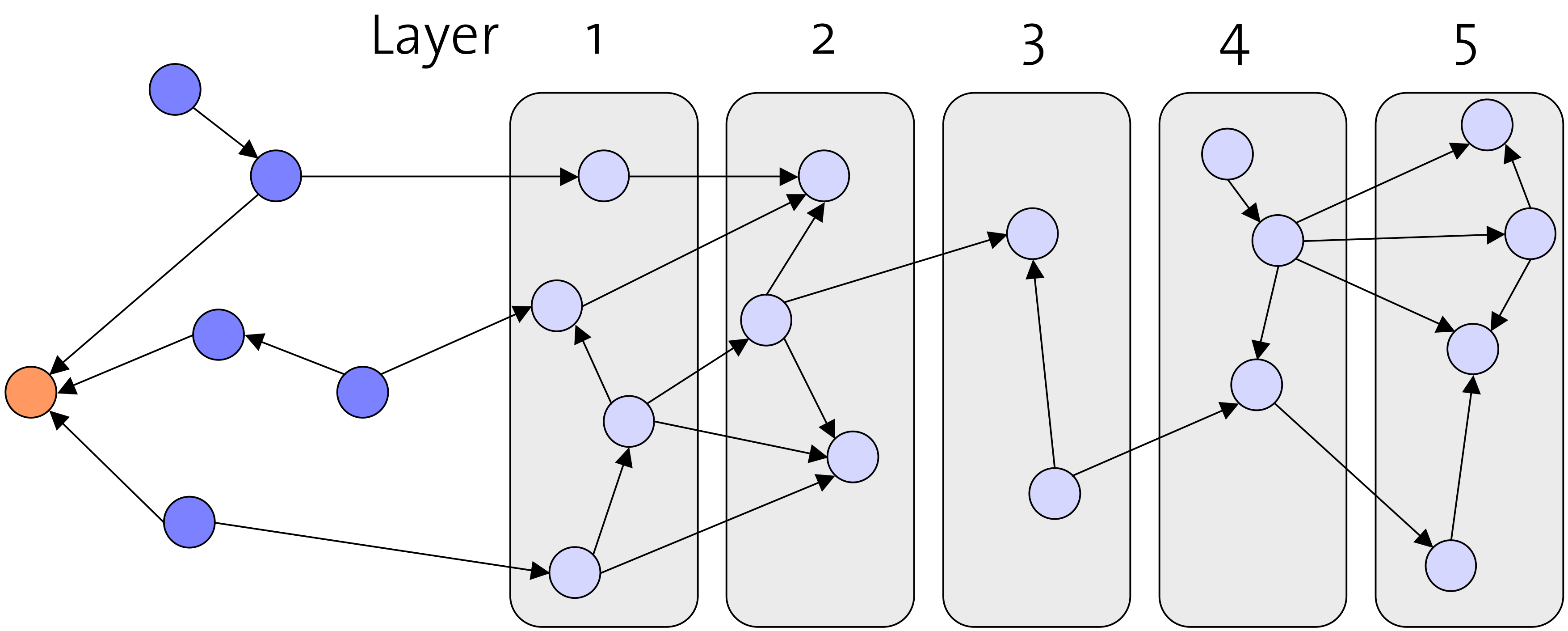


- But how many times do the bad nodes reverse?
- Idea: Group the bad nodes in layers!

Layers

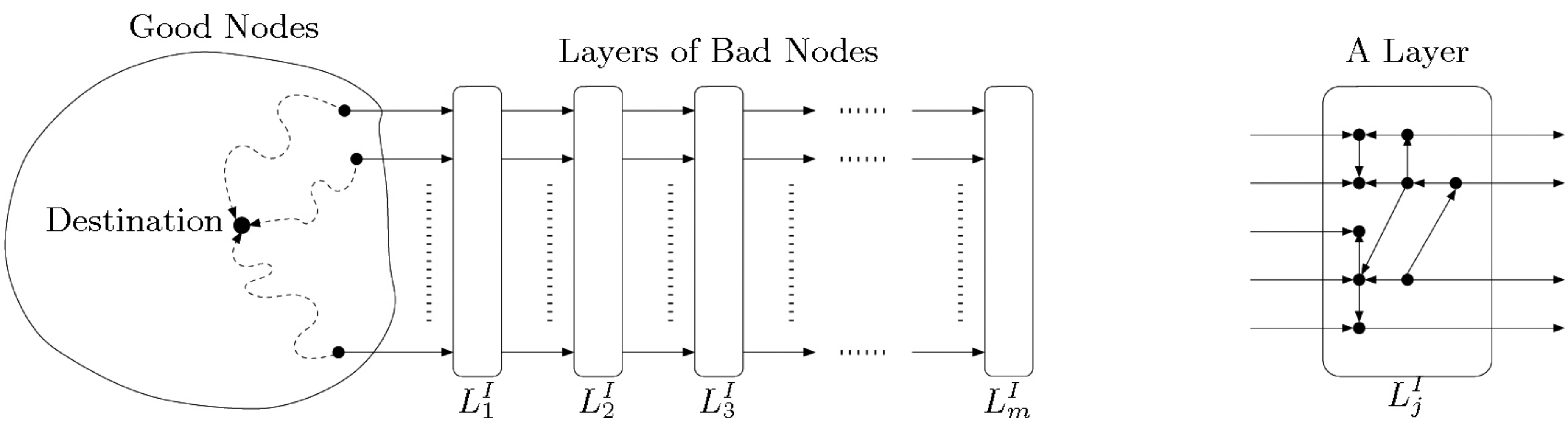
Full Reversal Algorithm

- Bad node *v* is in layer *i*, iff
  - there is an incoming link to *v* from a node in layer *i-1*, or
  - there is an outgoing link from *v* to a node in layer *i*.



Schematic View

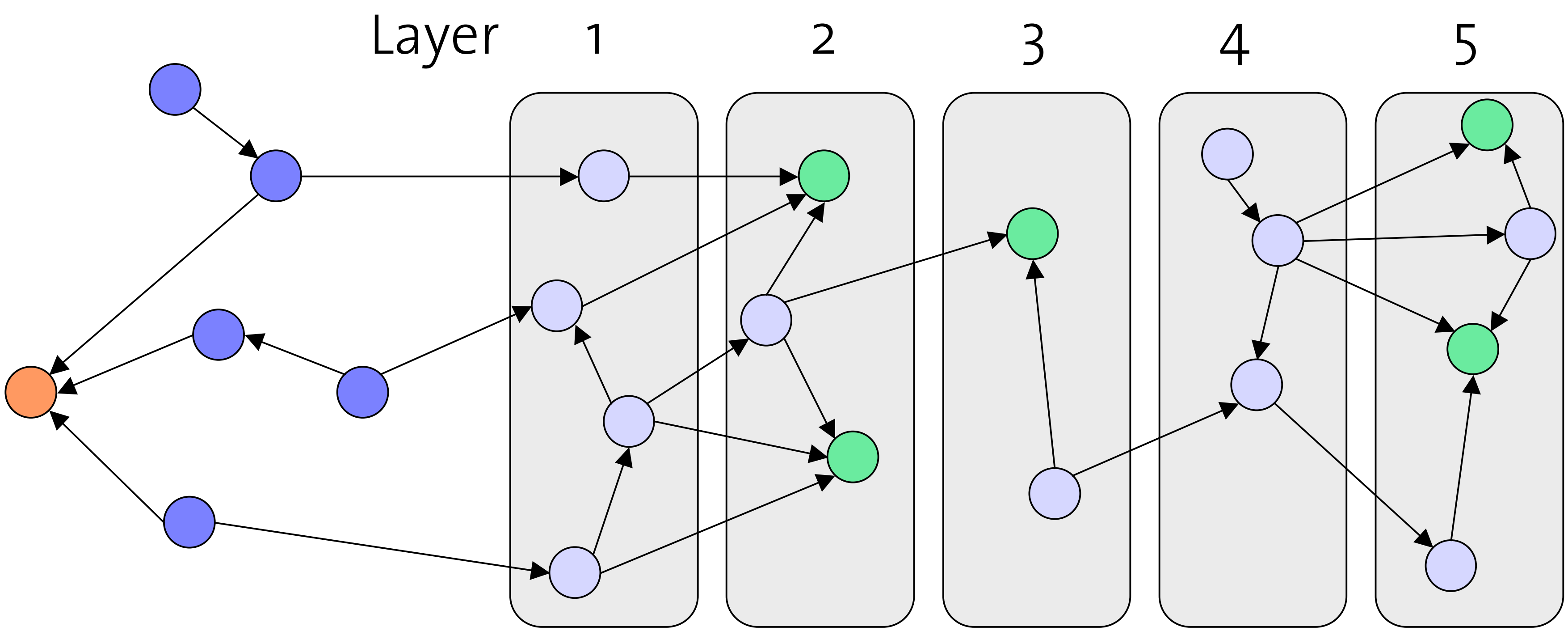
Full Reversal Algorithm



Execution E_1 (Step 1)

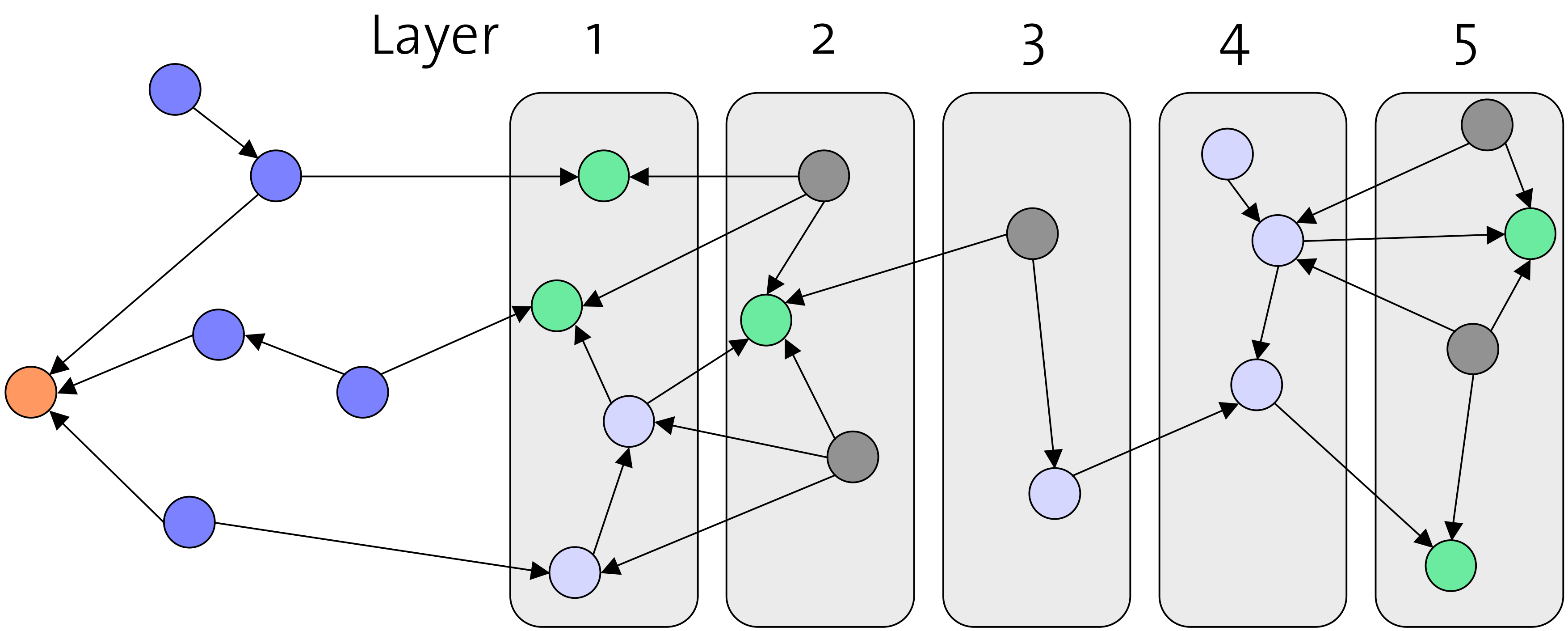
Full Reversal Algorithm

- There exists an execution E_1 which brings the system from state I to state I', such that every bad node reverses exactly one time.

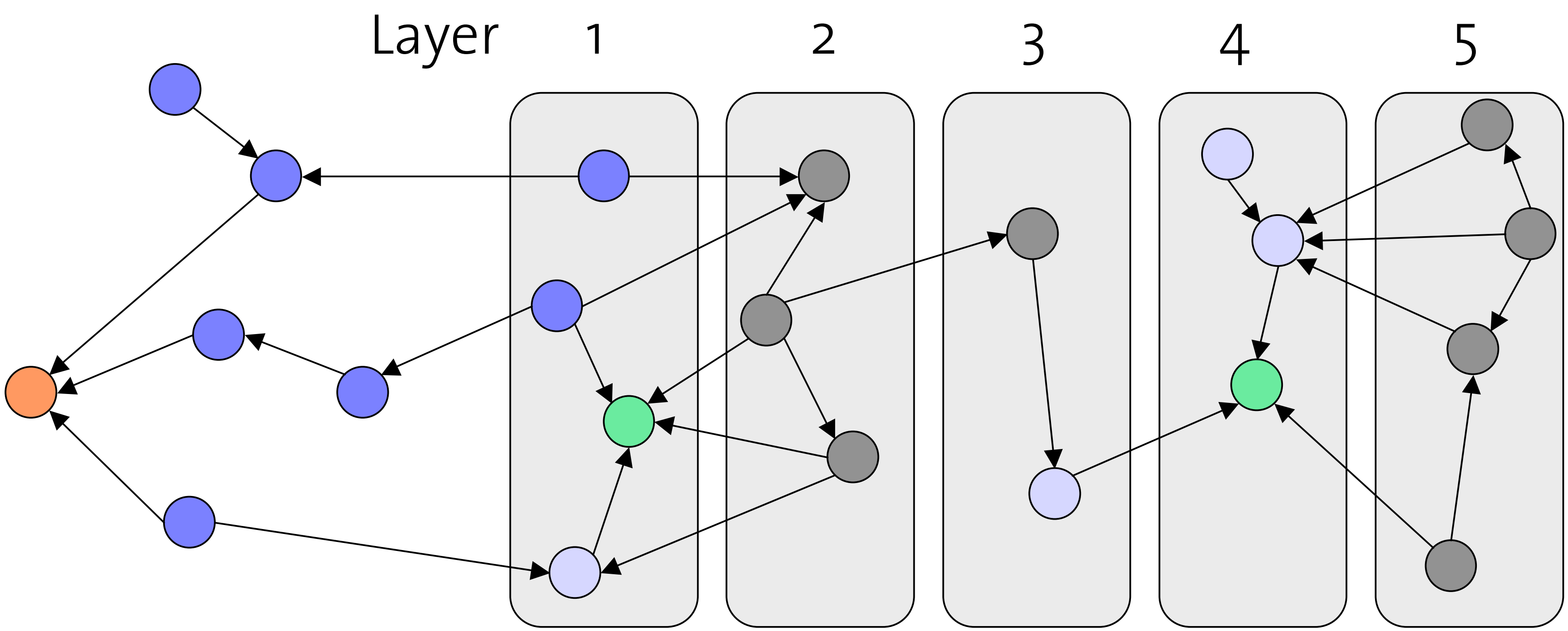


Execution E_1 (Step 2)

Full Reversal Algorithm

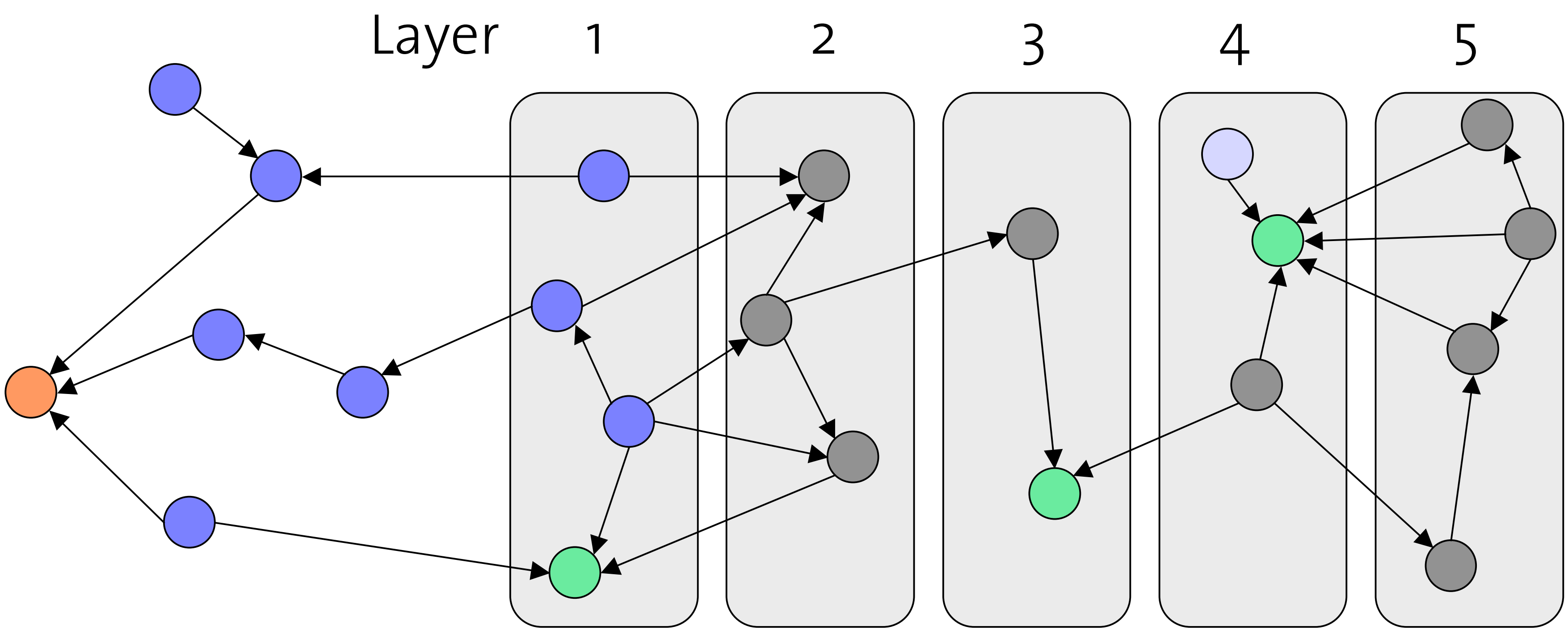


Execution E_1 (Step 3)
Full Reversal Algorithm



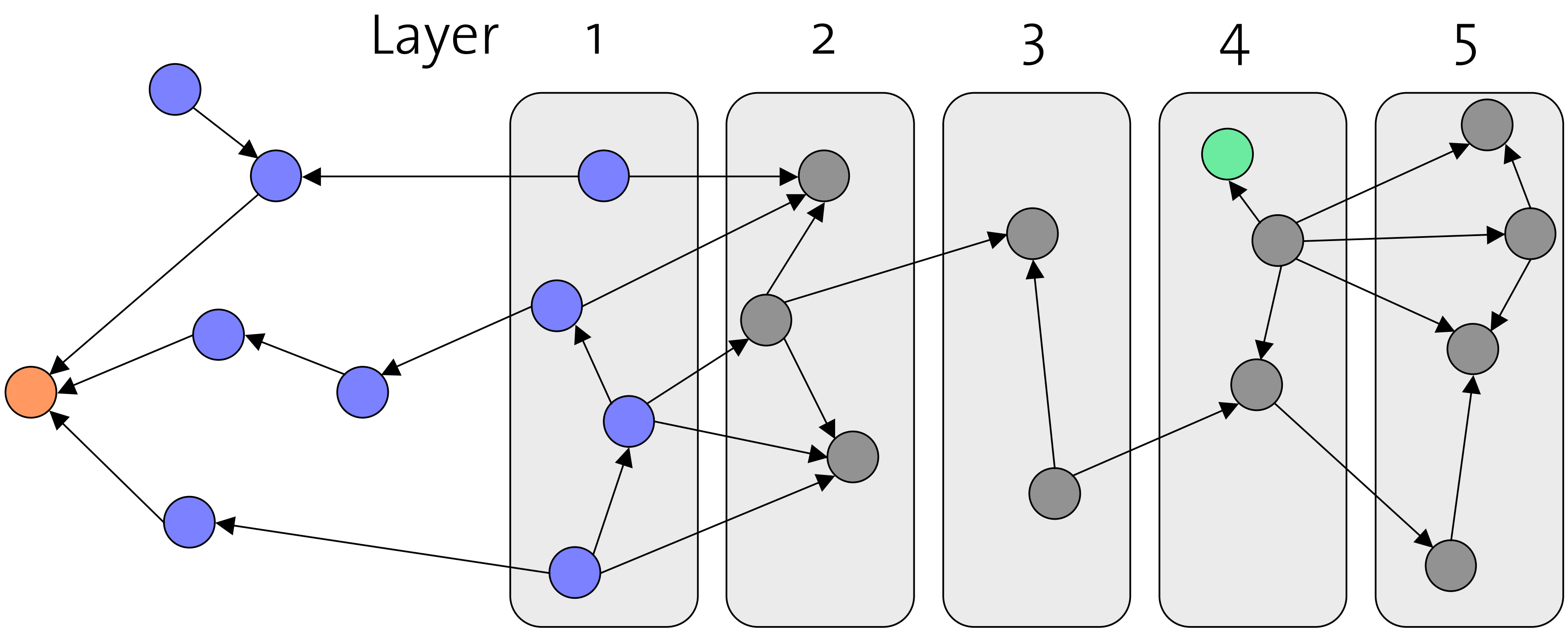
Execution E_1 (Step 4)

Full Reversal Algorithm



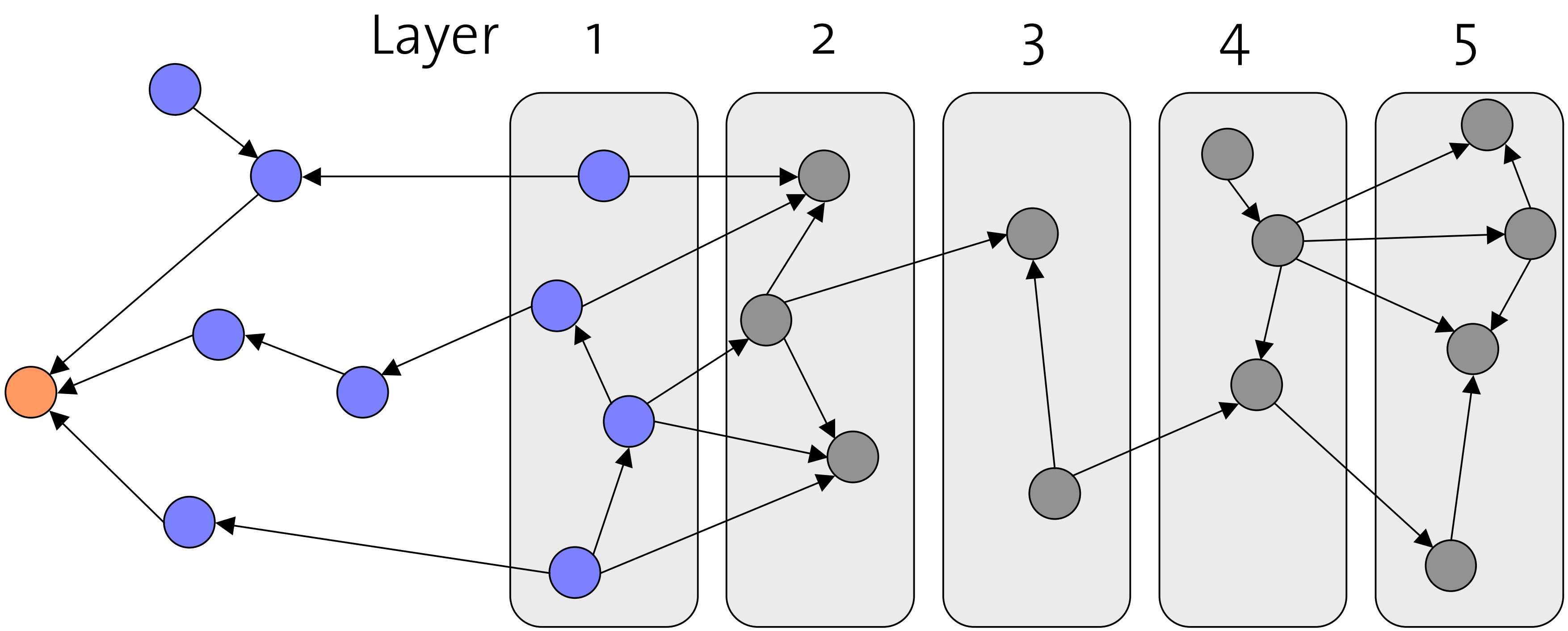
Execution E_1 (Step 5)

Full Reversal Algorithm



End of Execution E_1

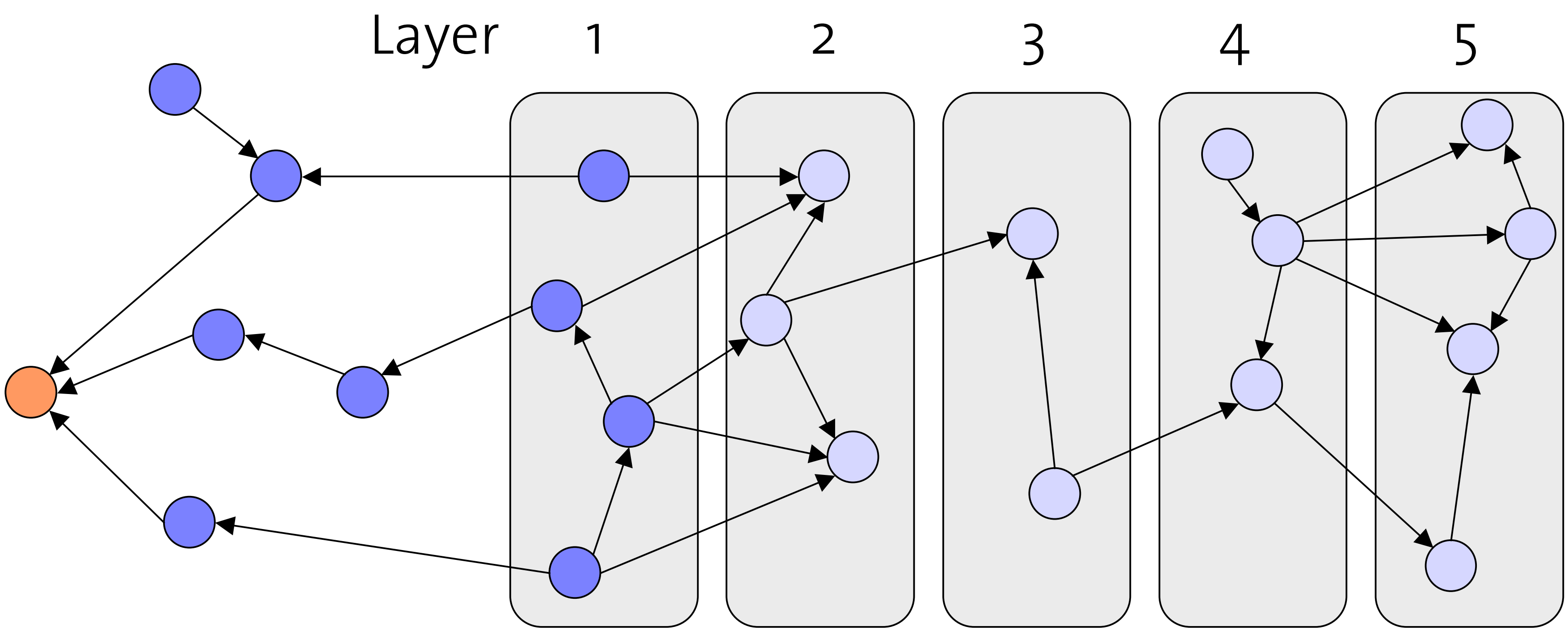
Full Reversal Algorithm



After Execution E_1

Full Reversal Algorithm

- At the end of this execution, all the bad nodes of layer 1 have become good, while all the bad nodes in the other layers stay bad.



Lemma

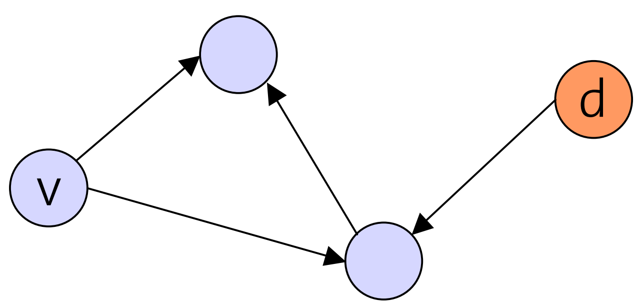
Full Reversal Algorithm

- **Lemma:** At the end of an execution E_i, all the bad nodes of layer i become good, while all the bad nodes in layers $j>i$, remain bad.

Proof

Full Reversal Algorithm

- Any bad node not adjacent to a good node will remain in the same (bad) node-state after execution E_i.
  - Node-state: directions of its incident links



Each neighbor node is bad in state I

$\Rightarrow$ Each of them reverses in E_i

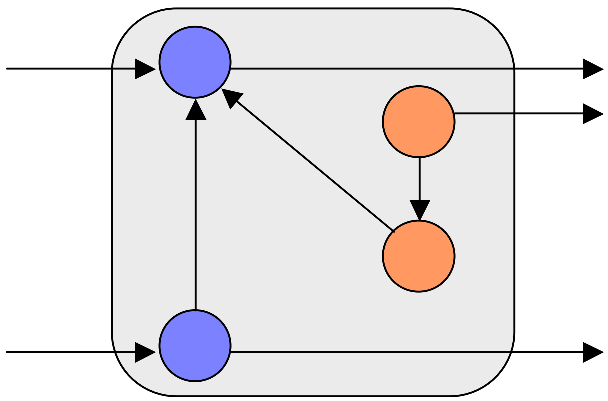
$\Rightarrow$ v also reverses in E_i

$\Rightarrow$ Reversals leave the directions the same

Proof

Full Reversal Algorithm

- Proof:
  - Bad nodes of layer *i* become good:

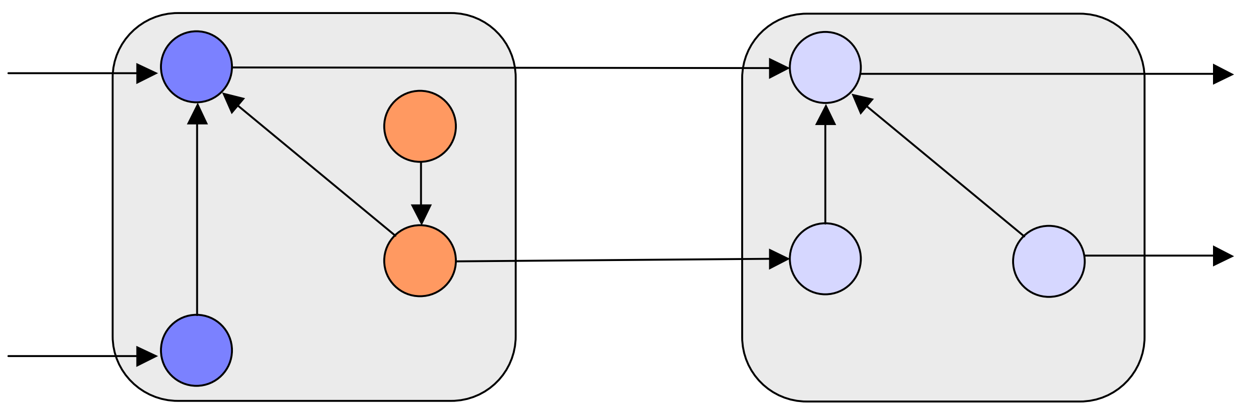


Layer i

- (blue) Nodes connected with an incoming link to a good node
- (orange) Nodes connected with an outgoing link to another node in layer *i*

Proof
Full Reversal Algorithm

- Bad nodes in layers $j > i$ remain bad.



□

Lemma

Full Reversal Algorithm

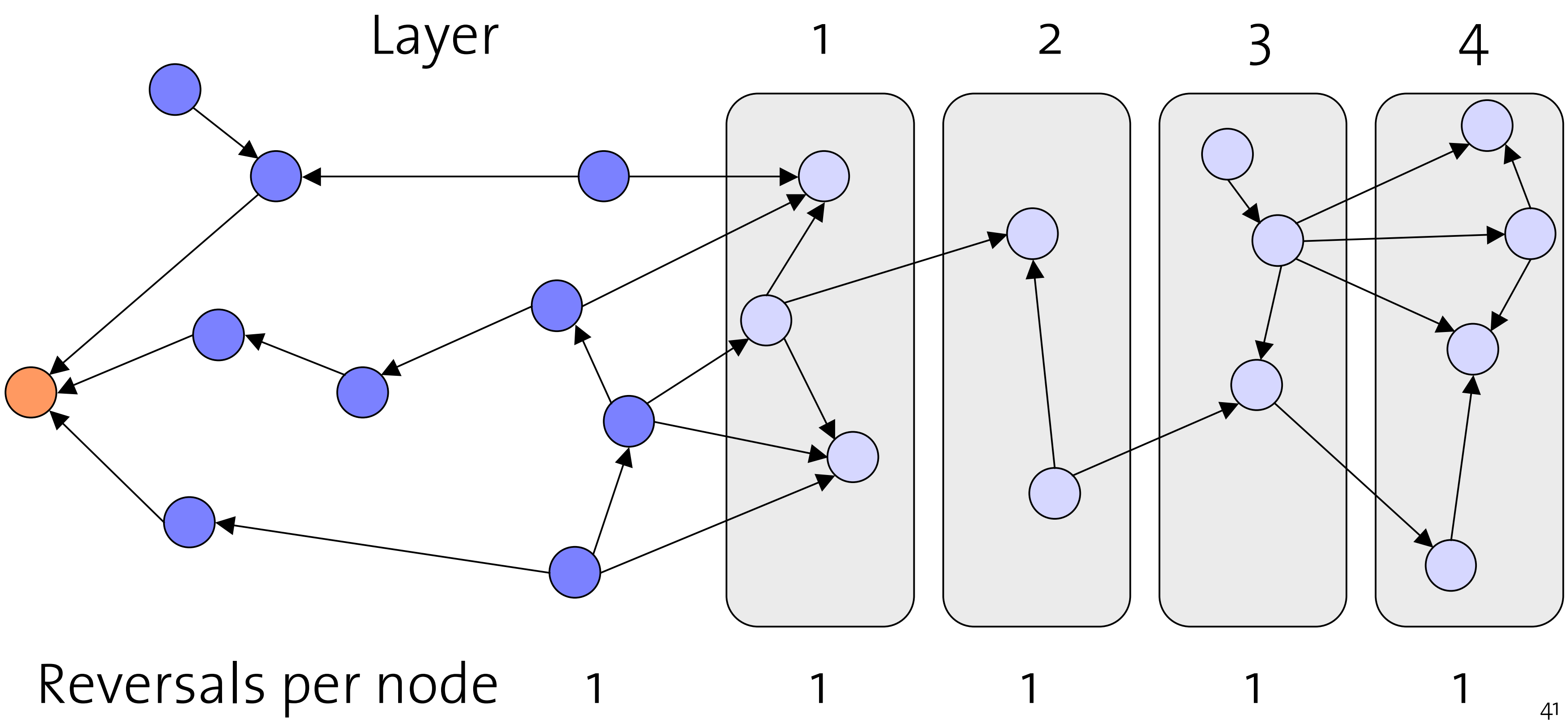
- **Lemma:** Layer $j+1$ becomes layer j after execution E_i (in the new state).
- **Proof:**
  - All bad nodes of layer i become good and bad nodes in other layers remain bad.
  - All bad nodes in layers $j>i$ remain in the same node-state.

□

Back to our example

Full Reversal Algorithm

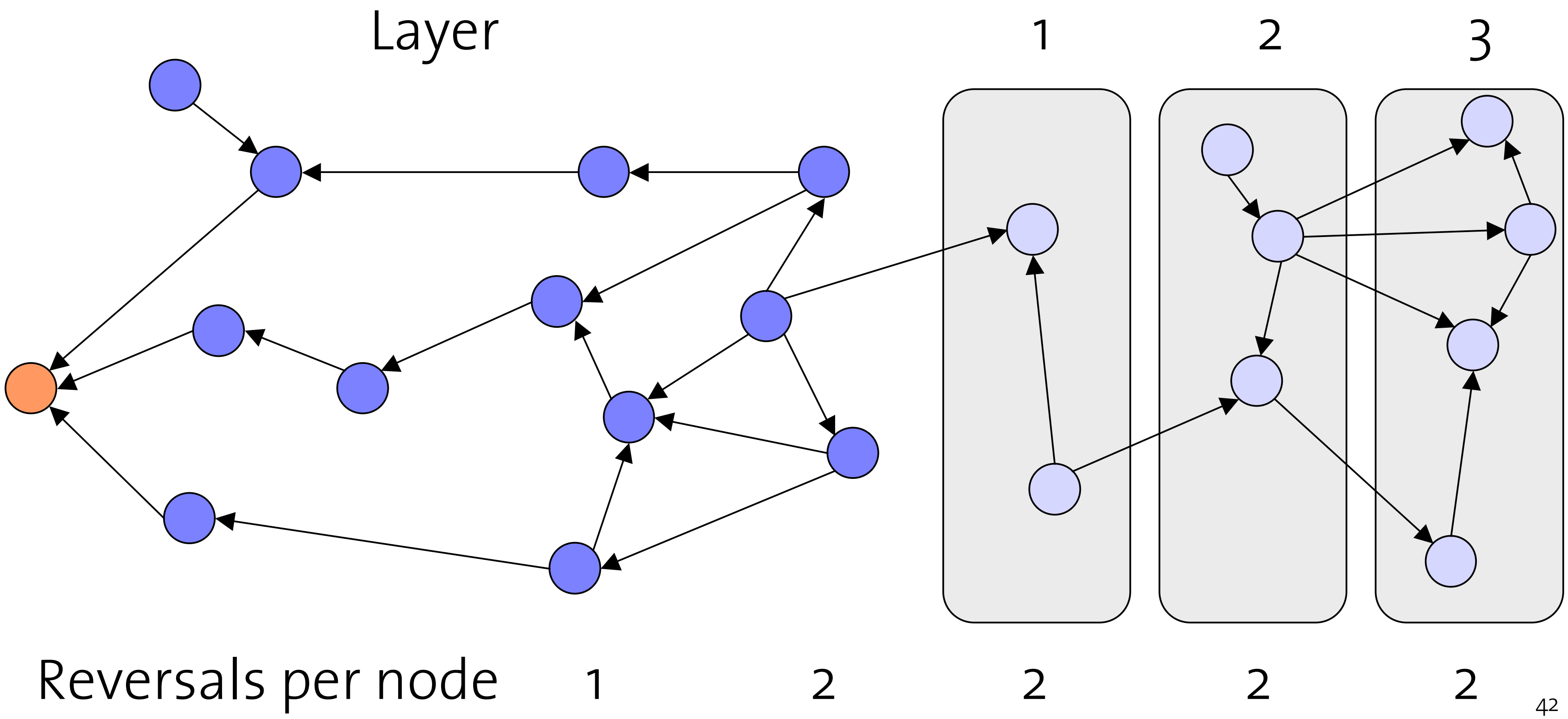
- After execution E_1



Back to our example

Full Reversal Algorithm

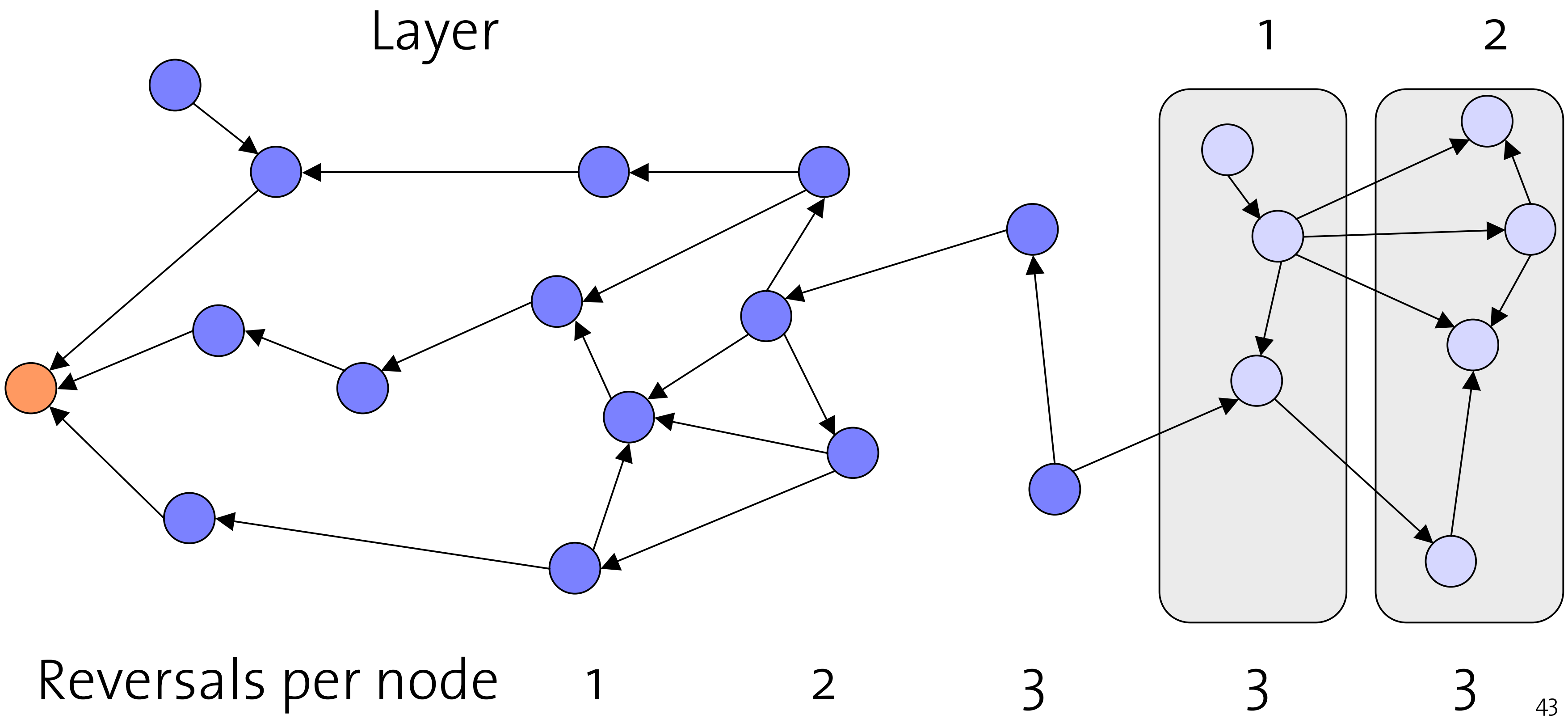
- After execution E_2



Back to our example

Full Reversal Algorithm

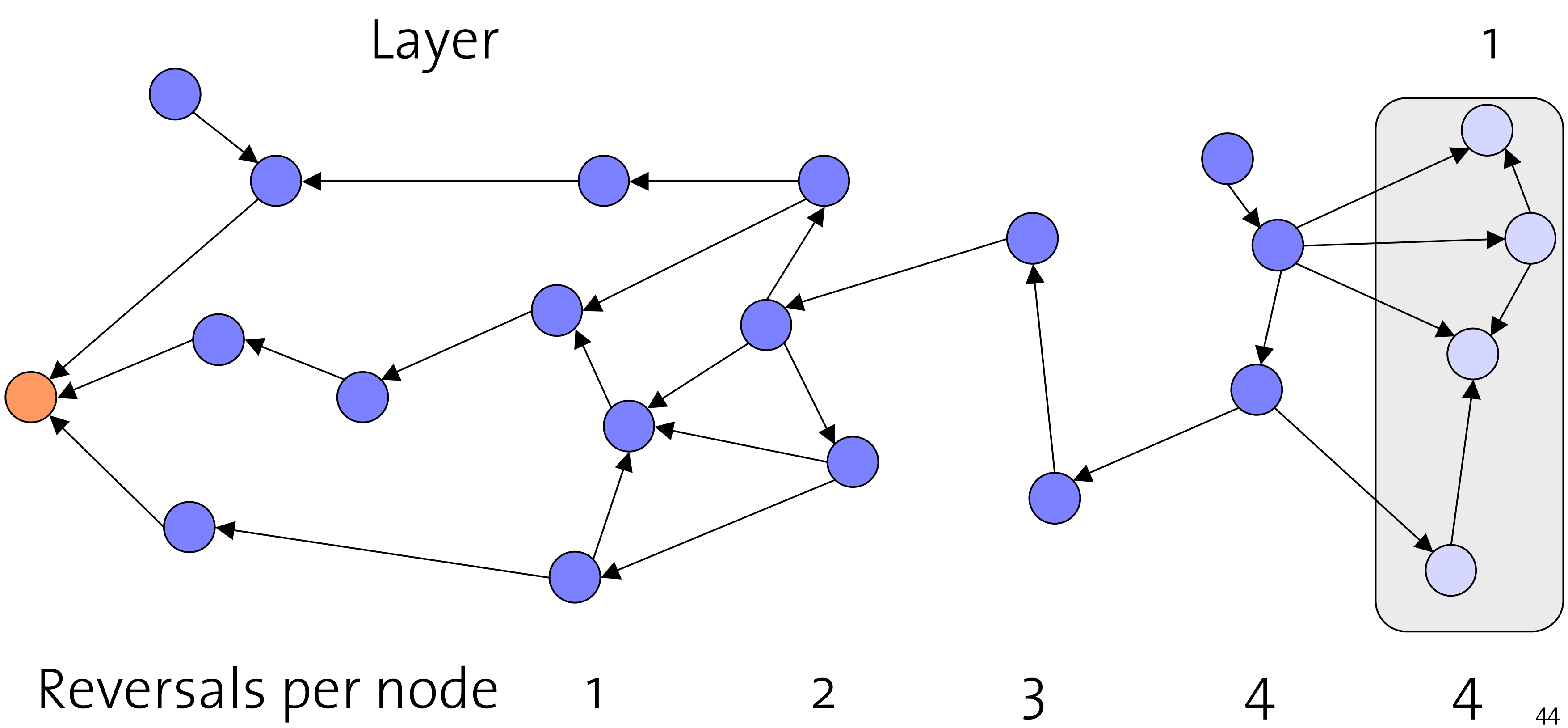
- After execution E_3



Back to our example

Full Reversal Algorithm

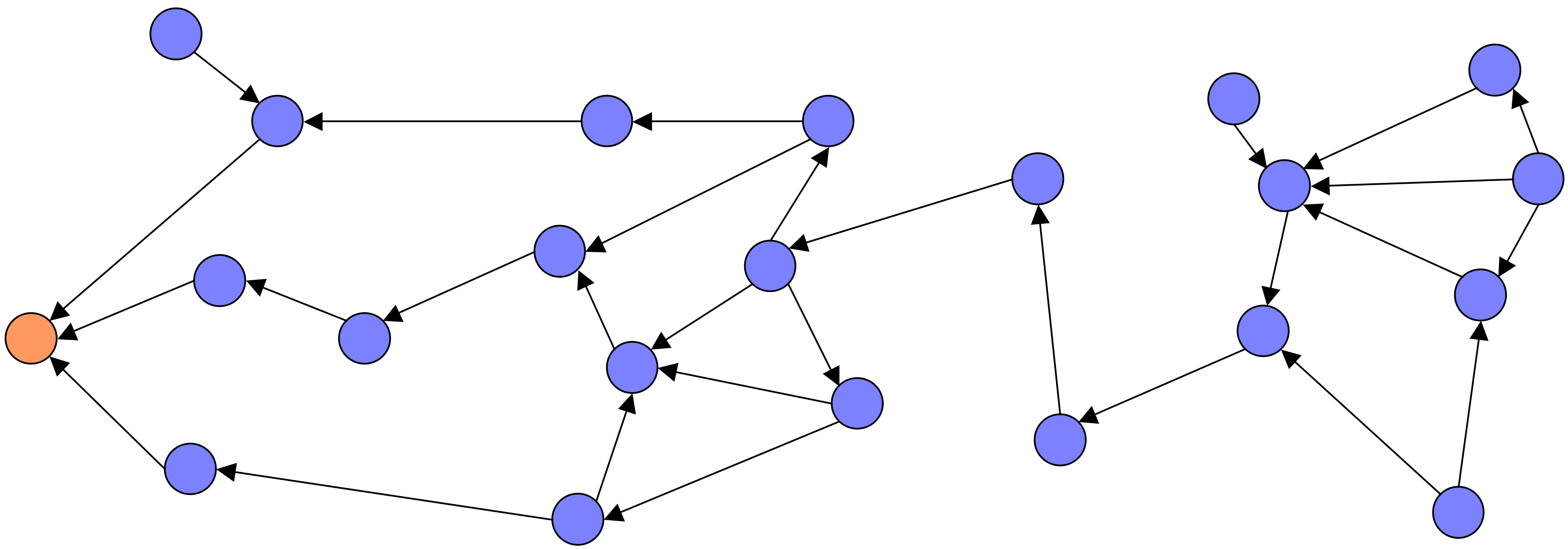
- After execution E_4



Back to our example

Full Reversal Algorithm

- After execution E_5

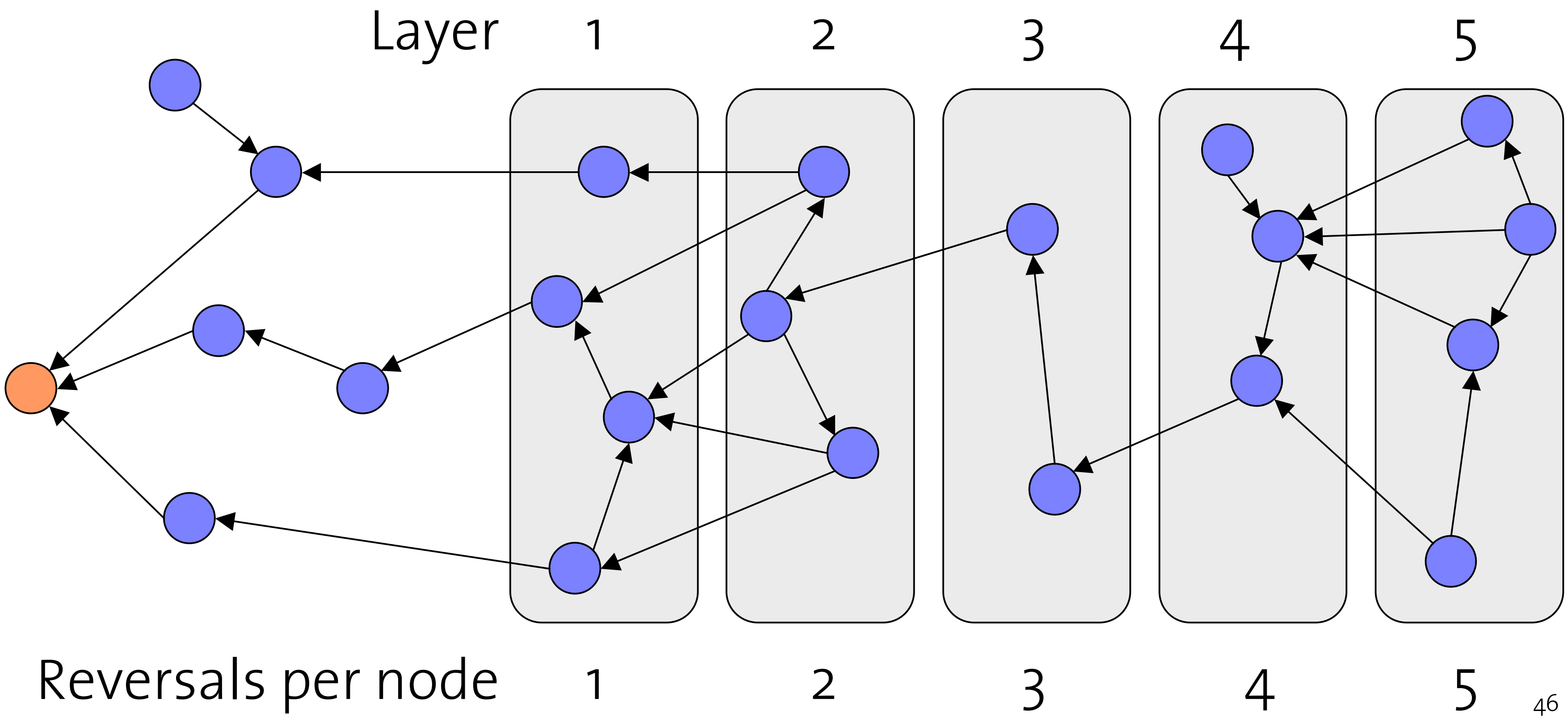


Reversals per node 1 2 3 4 5

Number of Reversals

Full Reversal Algorithm

- Back to our question: how many times do the bad nodes reverse?



Number of Reversals
Full Reversal Algorithm

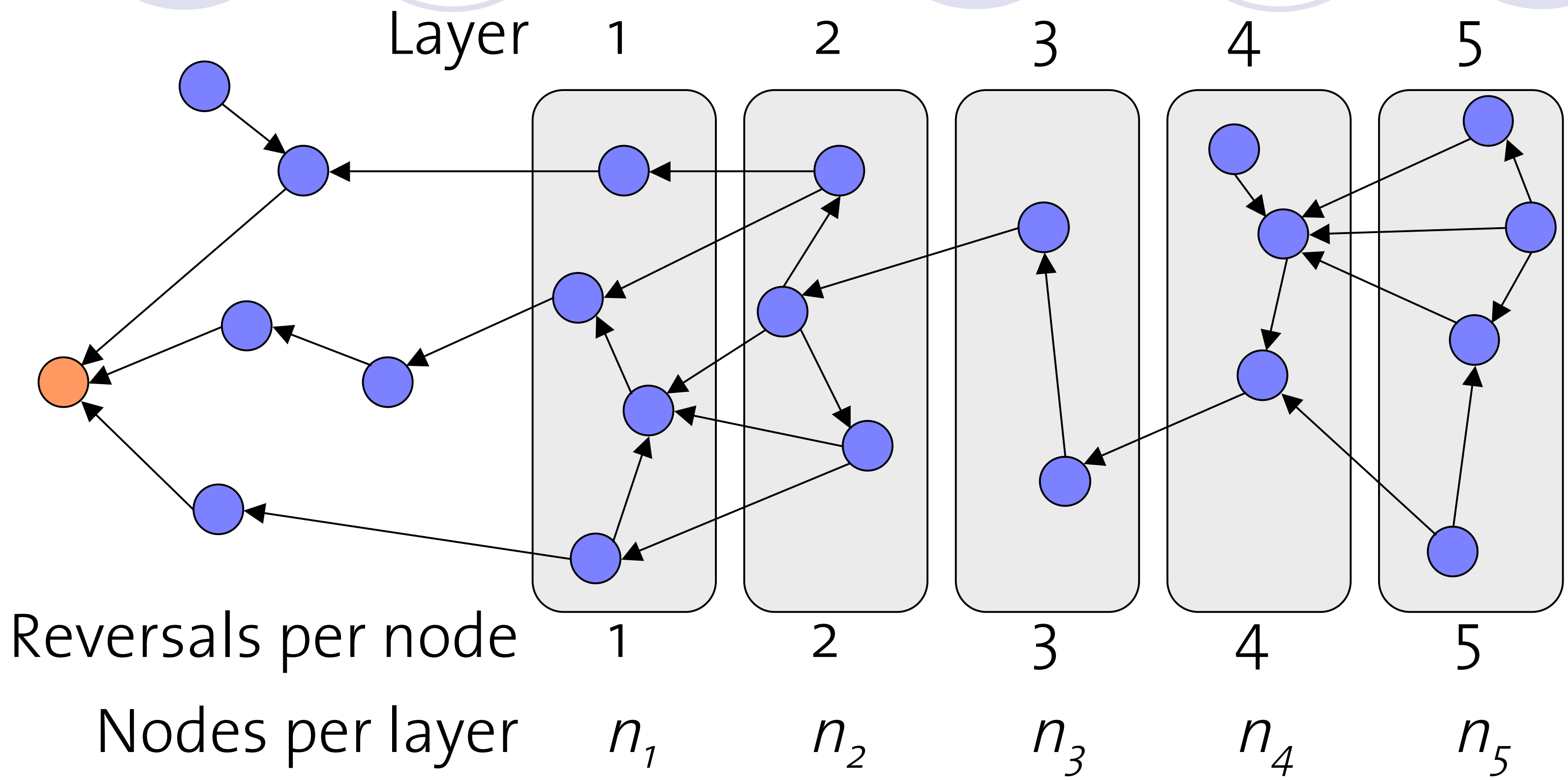
- Every bad node reverses in each execution exactly one time.
- Each node in layer 1 became good after 1 reversal. Each node in layer 2 needed 2 reversals.

  => Each node in layer *i* needs *i* reversals before it becomes a good node.

- Graph has *n* bad nodes
- Layer *i* has n_i nodes

Number of Reversals

Full Reversal Algorithm



⇒ Number of reversals: $n_1 \cdot 1 + n_2 \cdot 2 + n_3 \cdot 3 + n_4 \cdot 4 + n_5 \cdot 5$

⇒ Trivial upper bound for n bad nodes: $O(n^2)$

Upper Bound

Full Reversal Algorithm

- We get an upper bound for the number of reversals in the full reversal algorithm:

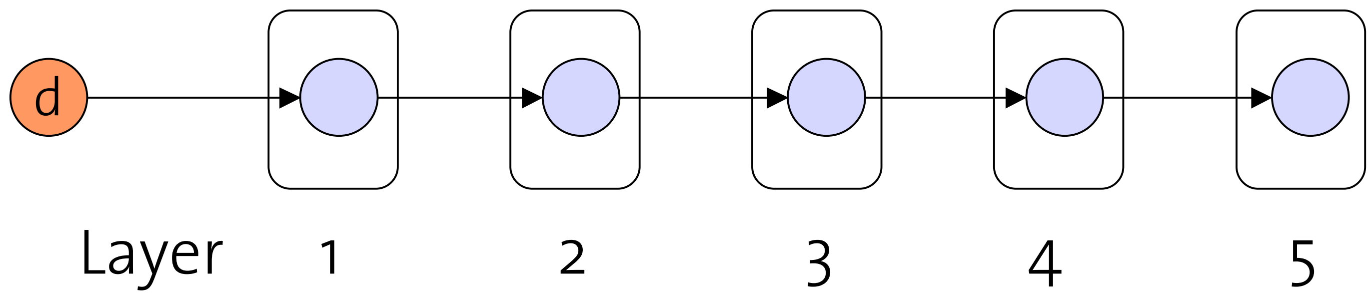
  For any graph with an initial state with n bad nodes, the full reversal algorithm requires at most $O(n^2)$ work and time till stabilization.

- We will now show that these bounds are tight

Lower Bound

Full Reversal Algorithm

- There is a graph with an initial state containing *n* bad nodes such that the full reversal algorithm requires $\Omega(n^2)$ work until stabilization.

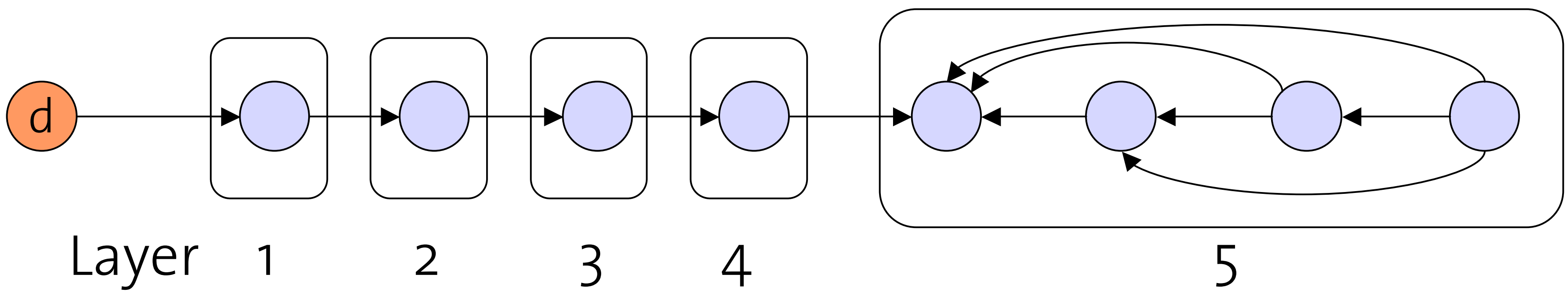


- Each node in layer *i* will reverse *i* times
- sum of all reversals is 1+2+3+…+n = $n(n+1)/2 = \Omega(n^2)$

Lower Bound

Full Reversal Algorithm

- There is a graph with an initial state containing n bad nodes such that the full reversal algorithm requires $\Omega(n^2)$ time until stabilization.



- $\lfloor n/2 \rfloor + 1$ layers
- First $\lfloor n/2 \rfloor$ layers contain 1 node each
  last layer contains $\lceil n/2 \rceil$ nodes
- sum of all reversals is $1+2+\ldots+\lfloor n/2 \rfloor + (\lfloor n/2 \rfloor + 1)\cdot\lceil n/2 \rceil = \Omega(n^2)$

Partial Reversal Algorithm
Performance Analysis

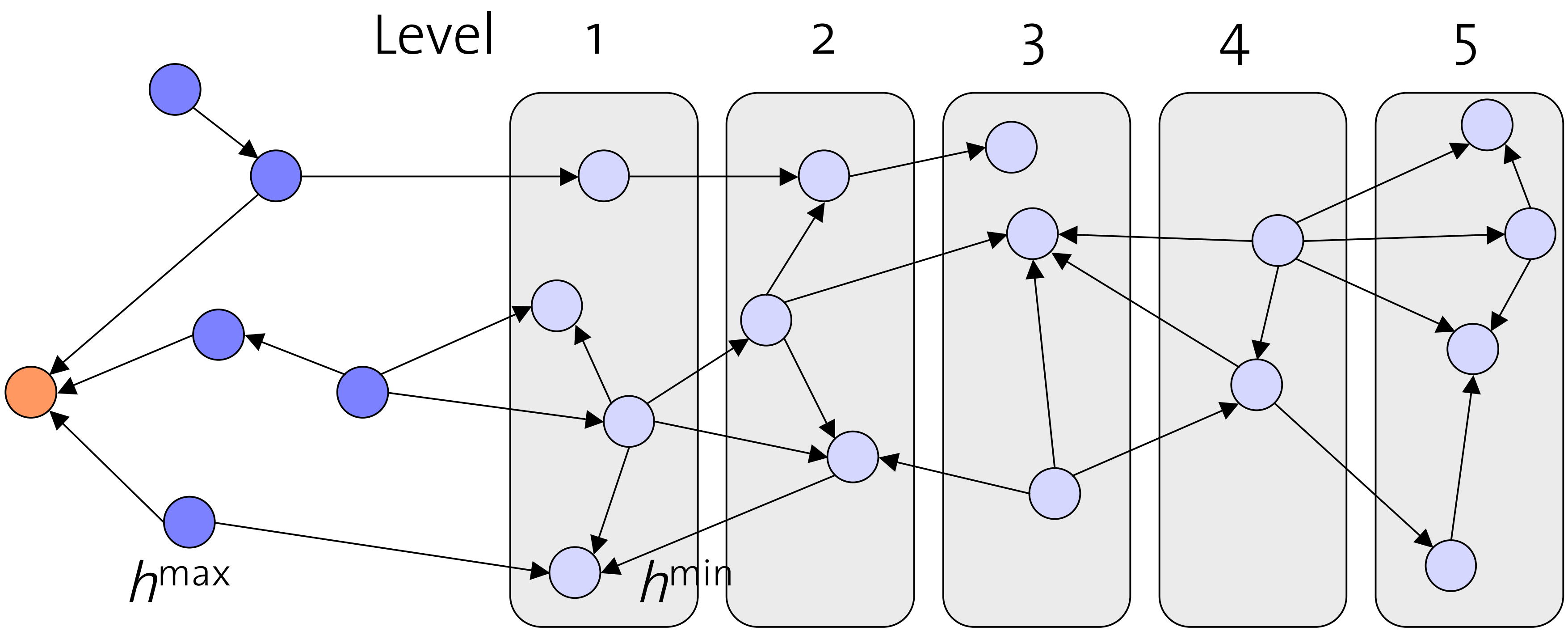
- One might expect that the partial reversal algorithm needs less reversals in the worst case than the full reversal algorithm.
  Is this true?

- Idea: group the bad nodes in levels.

Levels

Partial Reversal Algorithm

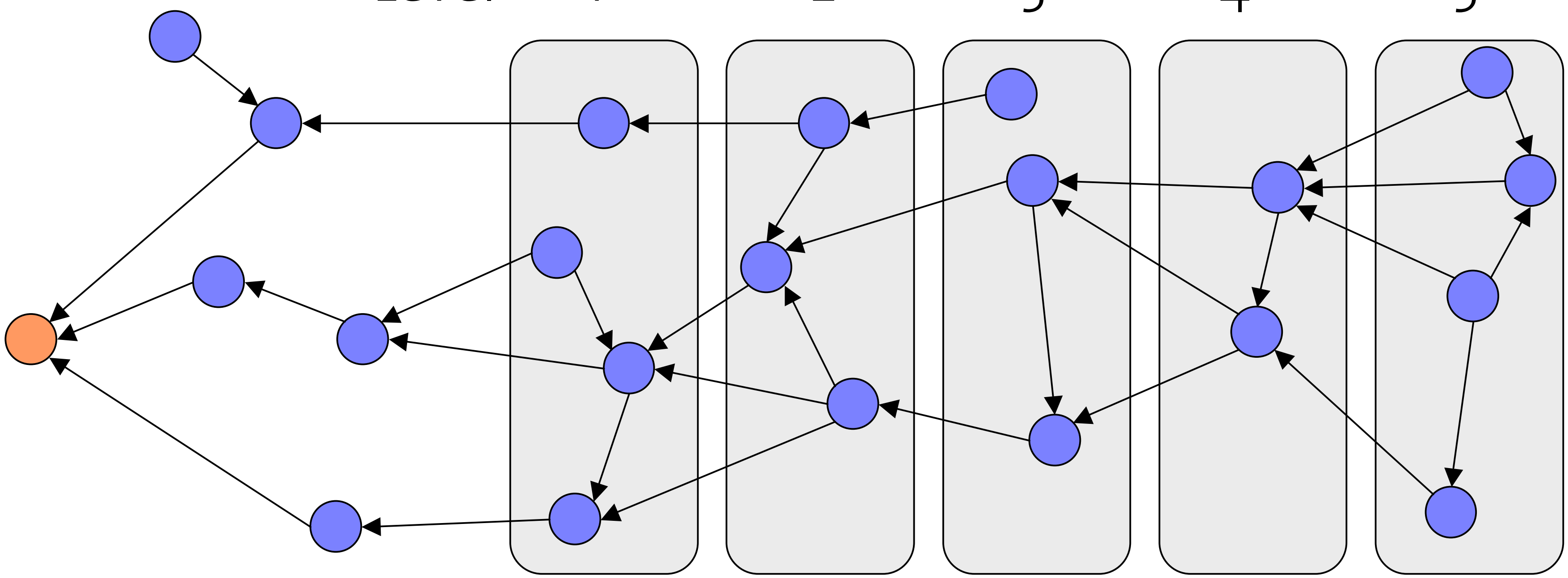
- Bad node *v* is in level *i*, if the shortest undirected path from *v* to a good node has length *i*.



Some Reversals later

Partial Reversal Algorithm

Level 1 2 3 4 5

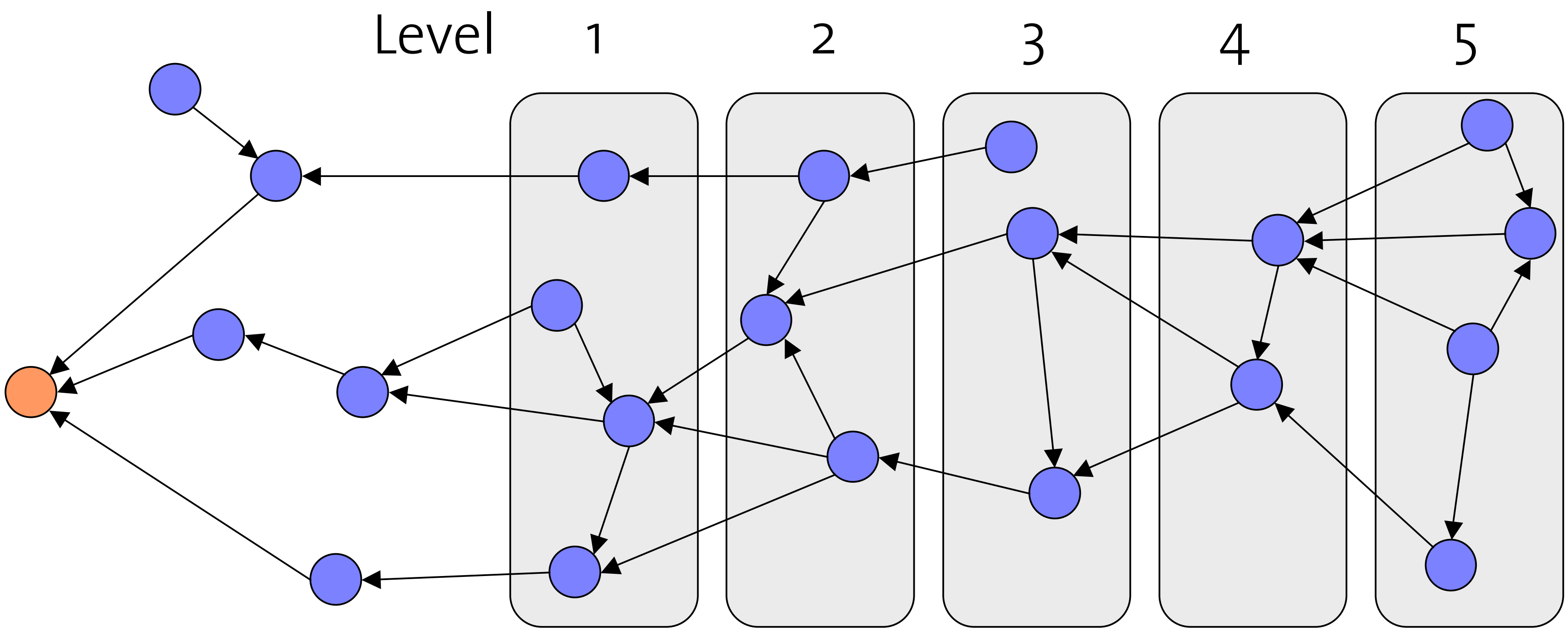


Upper bound on height

h^{max} +1 +2 +3 +4 +5

Number of Reversals

Partial Reversal Algorithm



Upper bound on number of reversals

$h^{max} - h^{min} = h^*$ +1 +2 +3 +4 +5

Each reversal increases the height by at least 1.

Upper Bound
Partial Reversal Algorithm

- A bad node needs in the worst case h^*+n reversals.
- We have n bad nodes:
  => $O(n \cdot h^* + n^2)$

Upper Bound

Partial Reversal Algorithm

- For any initial state with n bad nodes, the partial reversal algorithm requires at most $O(n \cdot h^* + n^2)$ work and time until the network stabilizes.
  - Problem: h* (= h^{max} - h^{min}) may be arbitrarily large

Lower Bound

Partial Reversal Algorithm

- There is a graph with an initial state containing n bad nodes, such that the partial reversal algorithm requires $\Omega(n \cdot h^* + n^2)$ work (time) until stabilization.

Deterministic Reversal Algorithms

Definition

- Defined by a "height increase" function *g*.
- Heights of different nodes are unique
- Node v is sink with height h_v and adjacent nodes $v_1, v_2, \ldots, v_d$ with heights $h_1, h_2, \ldots, h_d$ -> v's height after reversal is $g(h_1, h_2, \ldots, h_v)$
- => Full and partial reversal algorithms are deterministic

Bounds

Deterministic Reversal Algorithms

- There is a graph with an initial state containing *n* bad nodes such that any deterministic reversal algorithm requires $\Omega(n^2)$ work (time) until stabilization.

=> Full reversal algorithm is optimal in the worst case, while the partial reversal algorithm is not!

Overview

- Link Reversal Routing Algorithms
  - Full Reversal
  - Partial Reversal
- Equivalence of Executions
- Performance Analysis
- Results
- Conclusion

Results

- Full reversal algorithm requires $O(n^2)$ work and time (n = nodes which have lost the routes to the destination)
- Partial reversal algorithm requires $O(n \cdot h^* + n^2)$ work and time (h^* = nonnegative integer)
- For every deterministic link reversal algorithm, there are initial states which require $\Omega(n^2)$

Overview

- Link Reversal Routing Algorithms
  - Full Reversal
  - Partial Reversal
- Equivalence of Executions
- Performance Analysis
- Results
- Conclusion

Conclusion

- Full reversal outperforms partial reversal algorithm in the worst case.
- Full reversal is optimal while the partial reversal algorithm is not.
- Number of reversals only depends on the number of bad nodes.
- Is there a variation of the partial reversal algorithm with $O(n^2)$ in the worst case?
- Partial reversal better in the average case?
- Analysis of non-deterministic algorithms (TORA)
- Algorithms only suited for connected graphs
- What about >1 destinations?

Thanks for your attention!

Questions