An Evaluation of the Rotor-Router Mechanism

Julian M. Rice | UCLA
FrontierLab@OsakaU: Graduate School of Information Science & Technology
Masuzawa Laboratory (増澤室)
Total Slide Count: 14 | August 2018

Research Background & Motivation

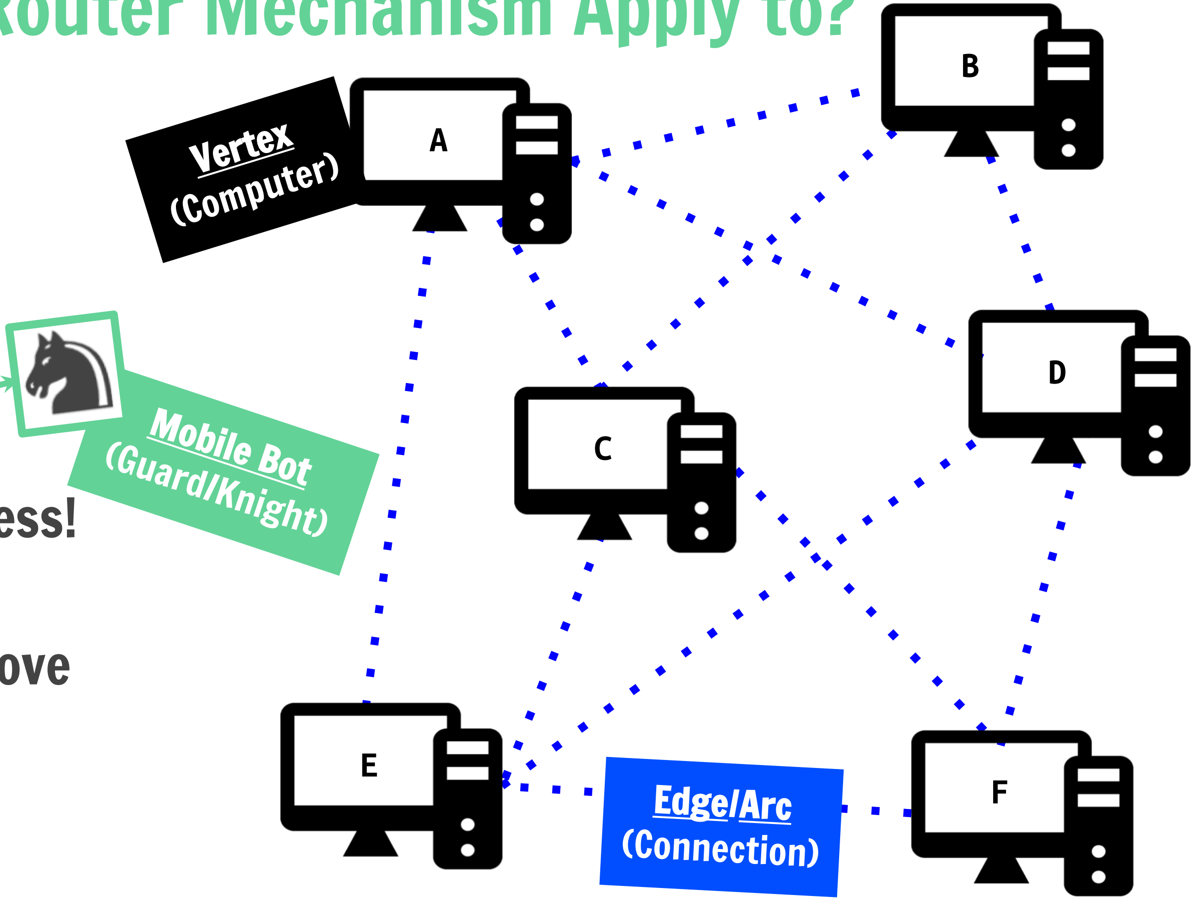
Looking Behind the Rotor-Router Mechanism

[Research Background -> Simulation & Programming -> Results & Conclusion -> Questions]

What does the Rotor-Router Mechanism Apply to?

I. Introduction

- Distributed Systems
- Patrolling Networks
- Mobile Bots
- Upgrade from Randomness!



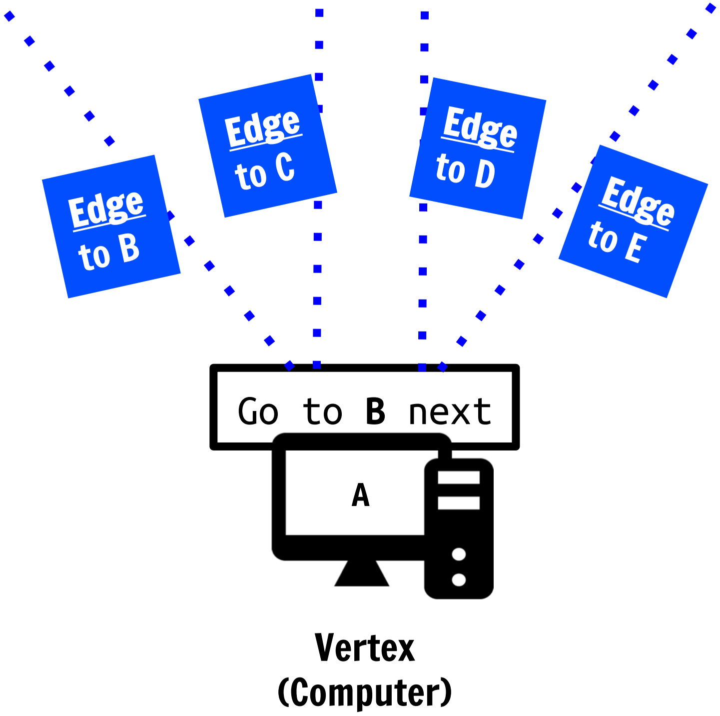
Migration: Allows a bot to move from one place to another.

Behind the Rotor-Router Mechanism

II. Rotor-Router Mechanism

- Alternative to Random Walk
- Each Vertex Points to an Edge
  - ✓ A (go to B, C, D, E, then B..)
- Deterministic Algorithm
- Euler Cycle (Lock-In)

Visiting: When a bot arrives at a vertex **or** crosses an edge.



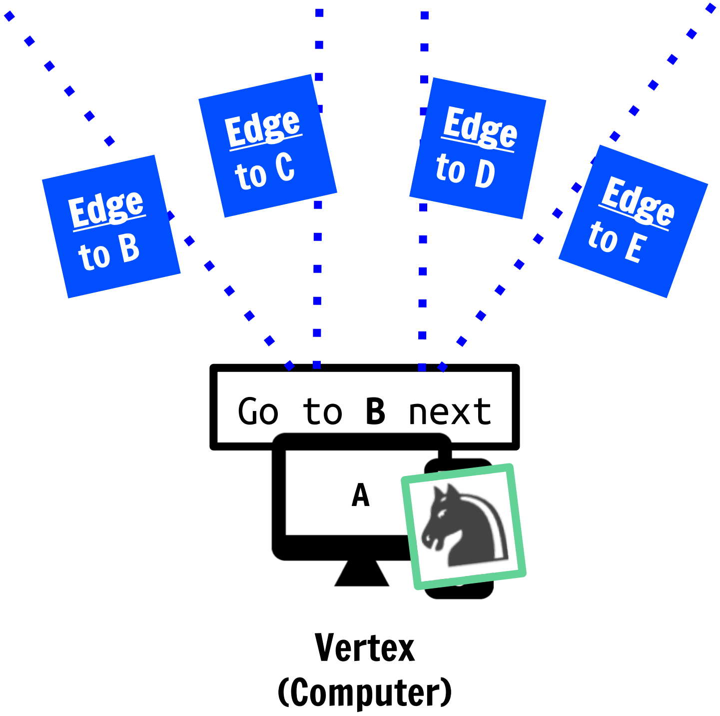
Vertex A's **fixed cyclic order** (↻): BCDE

Behind the Rotor-Router Mechanism

II. Rotor-Router Mechanism

- Alternative to Random Walk
- Each Vertex Points to an Edge
  - ✓ A (go to B, C, D, E, then B..)
- Deterministic Algorithm
- Euler Cycle (Lock-In)

Visiting: When a bot arrives at a vertex **or** crosses an edge.



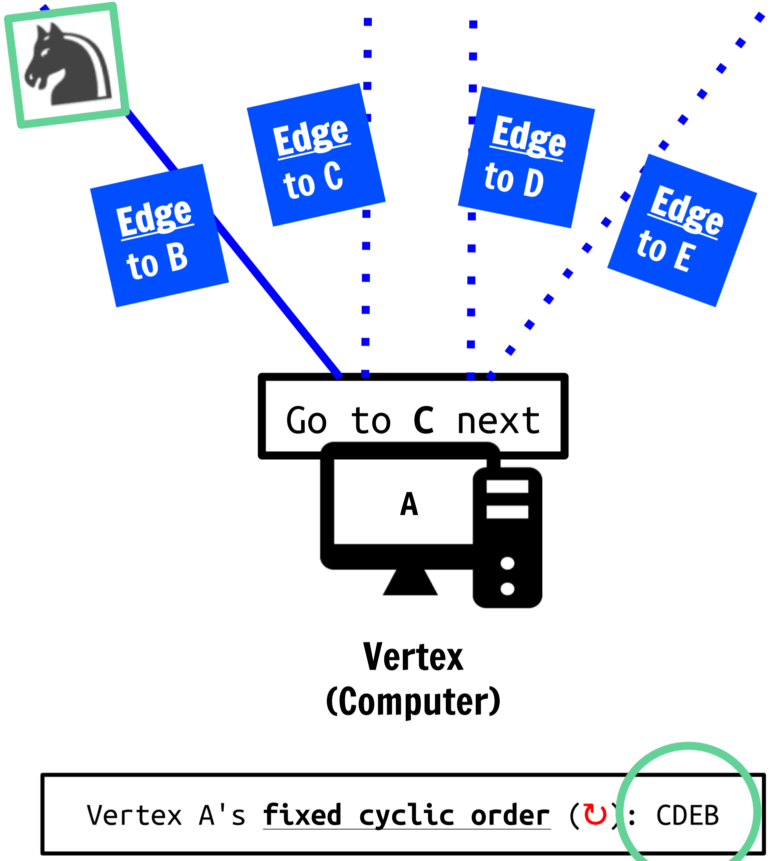
Vertex A's **fixed cyclic order** (↻): BCDE

Behind the Rotor-Router Mechanism

II. Rotor-Router Mechanism

- Alternative to Random Walk
- Each Vertex Points to an Edge
  - ✓ A (go to B, C, D, E, then B..)
- Deterministic Algorithm
- Euler Cycle (Lock-In)

Visiting: When a bot arrives at a vertex **or** crosses an edge.

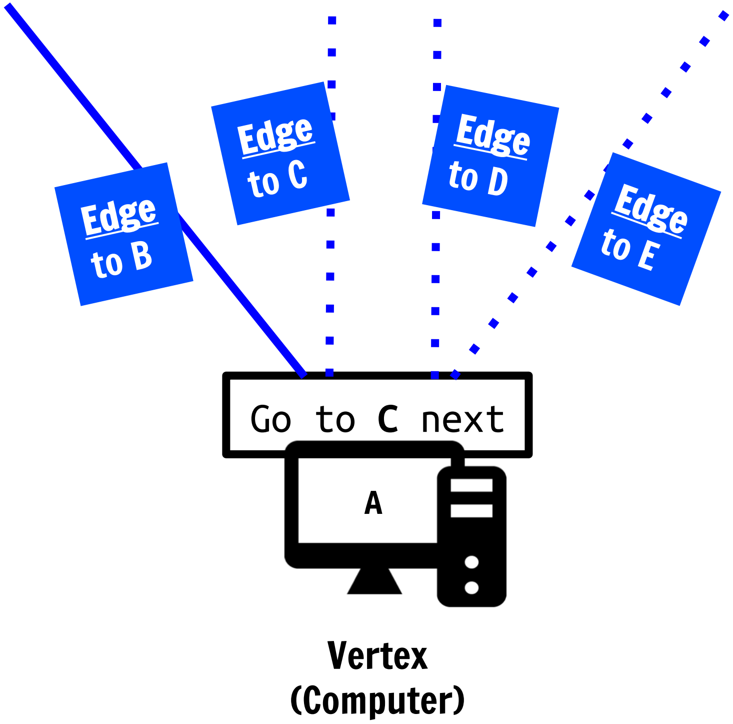


Behind the Rotor-Router Mechanism

II. Rotor-Router Mechanism

- Alternative to Random Walk
- Each Vertex Points to an Edge
  - ✓ A (go to B, C, D, E, then B..)
- Deterministic Algorithm
- Euler Cycle (Lock-In)

Visiting: When a bot arrives at a vertex **or** crosses an edge.



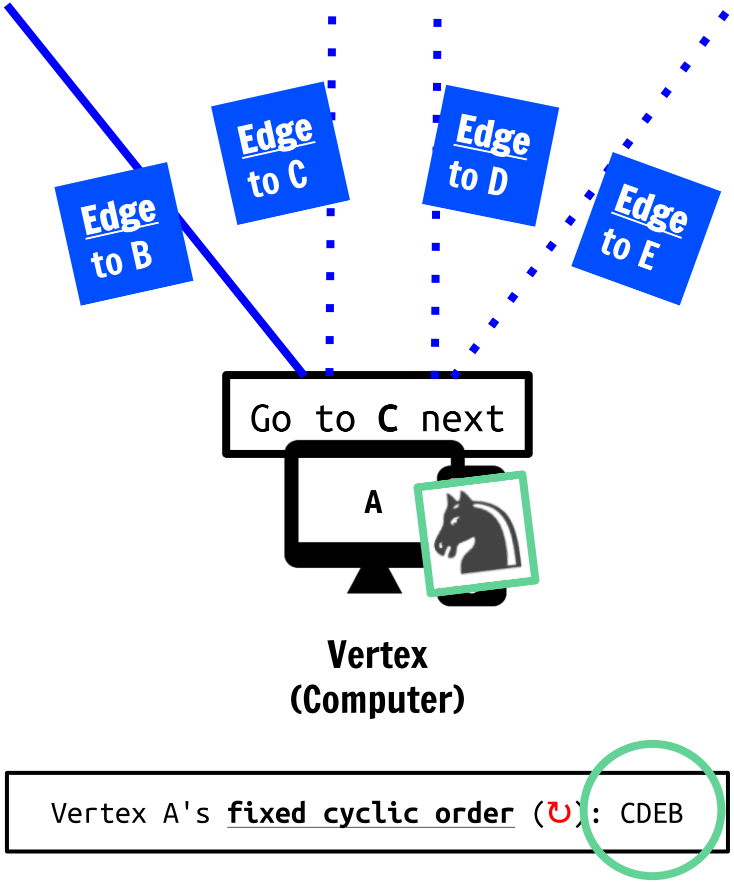
Vertex A's **fixed cyclic order** (↻): CDEB

Behind the Rotor-Router Mechanism

II. Rotor-Router Mechanism

- Alternative to Random Walk
- Each Vertex Points to an Edge
  - ✓ A (go to B, C, D, E, then B..)
- Deterministic Algorithm
- Euler Cycle (Lock-In)

Visiting: When a bot arrives at a vertex **or** crosses an edge.

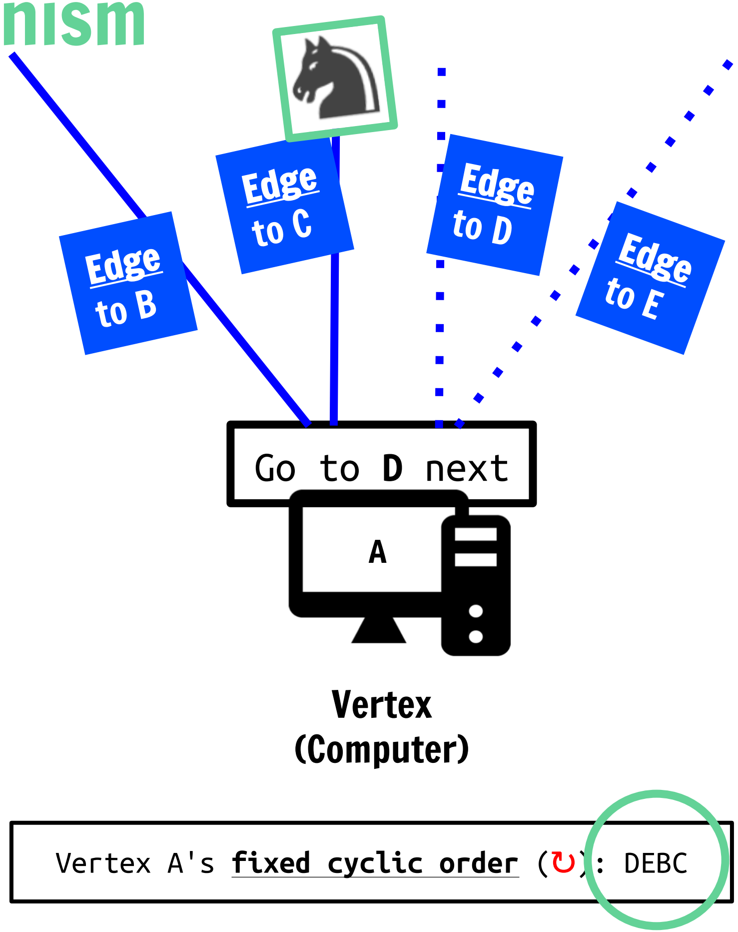


Behind the Rotor-Router Mechanism

II. Rotor-Router Mechanism

- Alternative to Random Walk
- Each Vertex Points to an Edge
  - ✓ A (go to B, C, D, E, then B..)
- Deterministic Algorithm
- Euler Cycle (Lock-In)

Visiting: When a bot arrives at a vertex **or** crosses an edge.

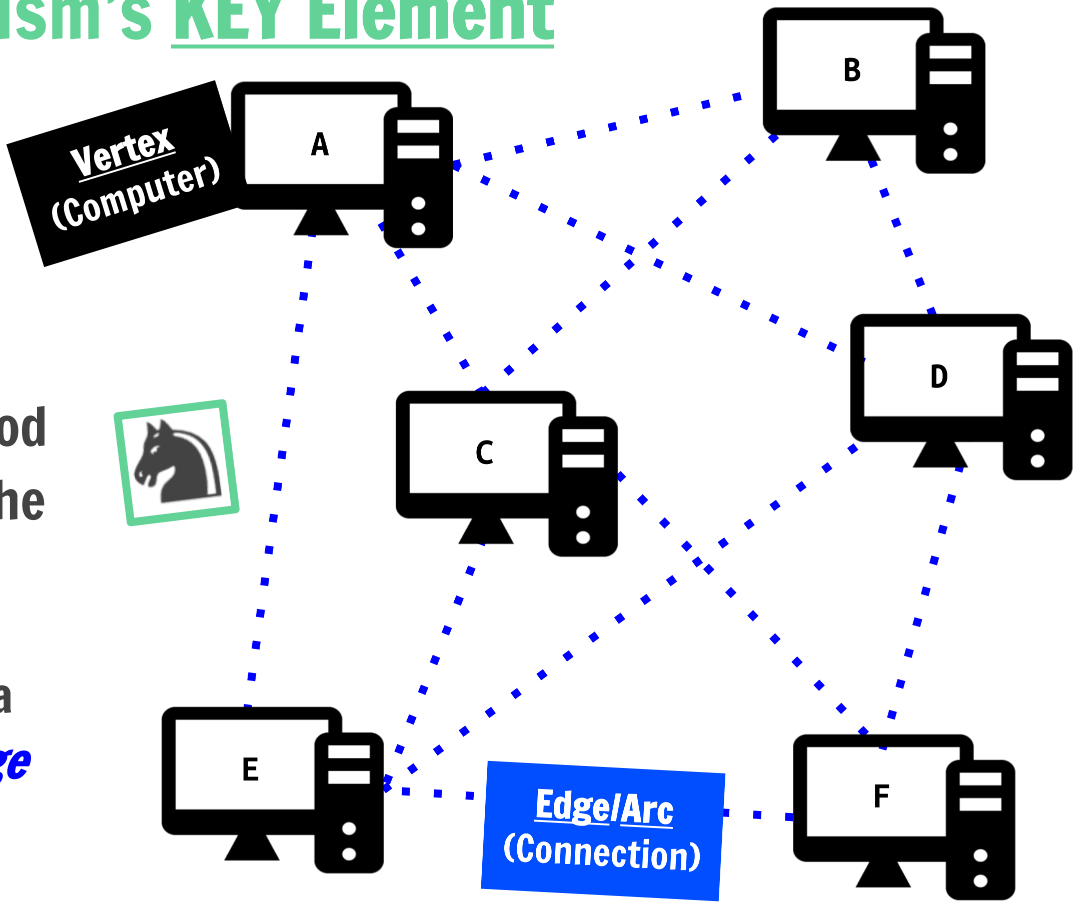


The Rotor-Router Mechanism's KEY Element

Euler Cycle: A path where every edge is visited ***exactly once***.

The bot will eventually end up entering a **Euler cycle** after a period of time (*exploring the network*), the **stabilization period**, passes.

GOAL: The overall goal is to lock into a **Euler cycle** and visit ***each & every edge*** as efficiently as possible.

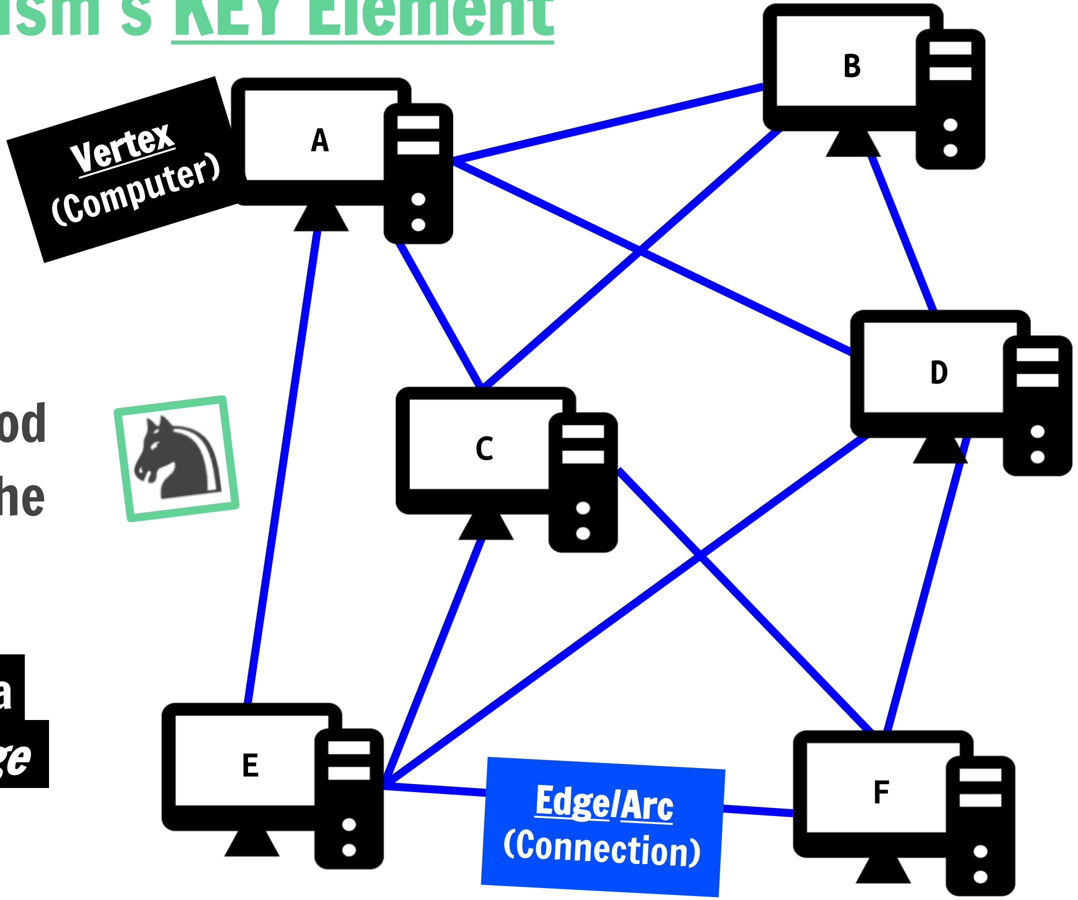


The Rotor-Router Mechanism's KEY Element

Euler Cycle: A path where every edge is visited *exactly* ***once***.

The bot will eventually end up entering a **Euler cycle** after a period of time (*exploring the network*), the **stabilization period**, passes.

GOAL: The overall goal is to lock into a Euler cycle and visit ***each & every edge*** as efficiently as possible !

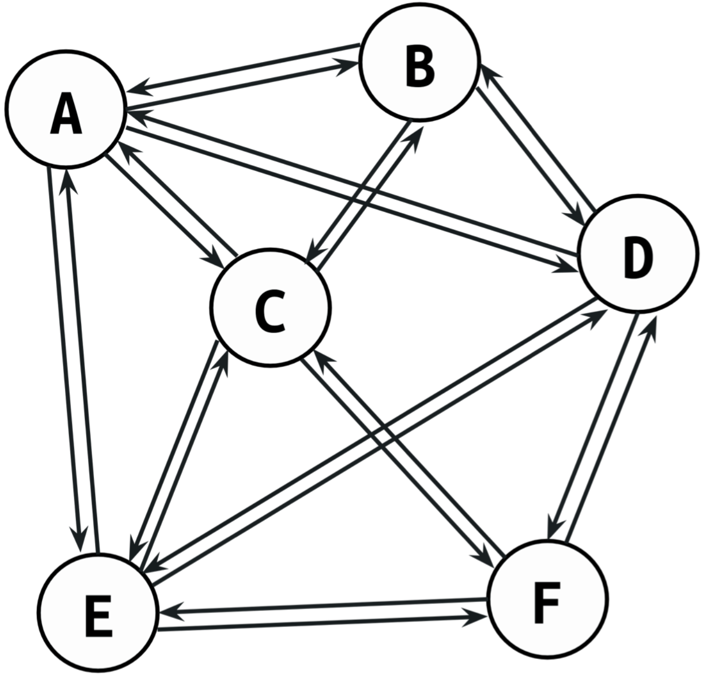


Preparing to Observe the Rotor-Router Algorithm

I created a custom "network" with arrows (arcs, rather than edges) that start at one vertex and end at another.

The supposed limit is **4 • m • D**, or the **number of edges (m = 11)** multiplied by the **diameter** of the graph **(D = 2)** ⇒ **44 steps**

What are the main questions that rise from this algorithm, and why perform a simulation?



Simplified visual of the previous diagram

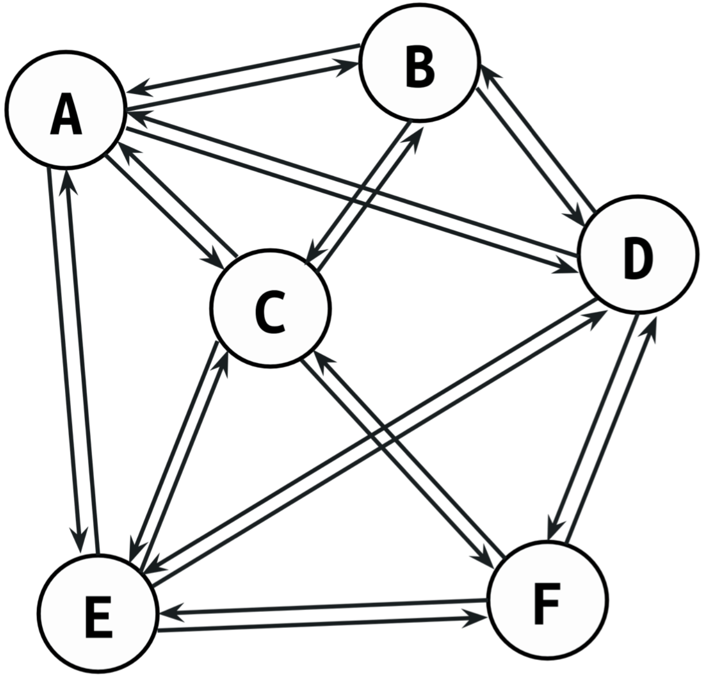
Preparing to Observe the Rotor-Router Algorithm: Mysteries

The supposed limit is 4 • m • D, or the number of edges (m = 22) multiplied by the diameter of the graph (D = 2) ⇒ 44 steps

Mysteries (Why, Why, Why?)

[1] No matter where the bot starts, it will always end up in a forever lasting loop.

[2] The worst case is 44 steps (for this example), but what is the *average* case? Is it close to 44, close to 0, or in the middle?



Simplified visual of the previous diagram

Simulations & Programming

What simulations did I run to get the data I needed?

[~~Research Background~~ -> Simulation & Programming -> Results & Conclusion -> Questions]

Simulation Program (C++)

***Files**: Edge.h, Vertex.h, Graph.h, StatList.h, Functions.h, Edge.cpp, Vertex.cpp, Graph.cpp, StatList.cpp, Functions.cpp, main.cpp*

Input: # of **steps** (for the agent)
of **tests** (for accuracy)

Output: Average visit count (edge)
Average visit count (vertex)
Stabilization period
(average, min, max, distr)
Euler cycle (repeated forever)

Line Count: 673 (11 files)

```
RotorRouter-v1
  Edge.h
  Vertex.h
  Graph.h
  StatList.h
  Functions.h
  Edge.cpp
  Vertex.cpp
  Graph.cpp
  StatList.cpp
  Functions.cpp
  main.cpp
  Products
    RotorRouter-v1
```

```cpp
int mobileAgent(Graph graph, Vertex* start, int iterations, string& temp) {
    Vertex* destination = NULL;
    for (int i = 0; i < iterations; i++) {
        int currentIndex = start->getIndex();
        start->edgeToCross(start, currentIndex)->incrementVi
        Edge* cross = start->edgeToCross(start, currentIndex
        destination = cross->getEndVertex();

        cout << "[Step " << i+1 << "] Agent crosses the EDGE
        << cross->getStartVertex()->getName() << cross->getE
        << " and is now at VERTEX " << cross->getEndVertex()

        graph.insertEdgeToOverall(cross); //New
        destination->incrementVisited();
        start->changeIndex();
        start = destination;
    }
```

```
[Console] Please input the number of times the mobile agent
[Console] Please input the number of times the test will be
[Step 1] Agent crosses the EDGE BA and is now at VERTEX A
[Step 2] Agent crosses the EDGE AE and is now at VERTEX E
[Step 3] Agent crosses the EDGE EC and is now at VERTEX C
[Step 4] Agent crosses the EDGE CE and is now at VERTEX E
[Step 5] Agent crosses the EDGE ED and is now at VERTEX D
[Step 6] Agent crosses the EDGE DF and is now at VERTEX F
[Step 7] Agent crosses the EDGE FD and is now at VERTEX D
[Step 8] Agent crosses the EDGE DA and is now at VERTEX A
[Step 9] Agent crosses the EDGE AB and is now at VERTEX B
[Step 10] Agent crosses the EDGE BC and is now at VERTEX C
```

```
-RESULTS---------------------------
-Number of STEPS per TEST: 30
-Number of TESTS: 1
 Average Visited Count for Vertices
   A: 6
   B: 3
   C: 5
   D: 6
   E: 6
   F: 4
 Average Visited Count for Edges
   AB: 3
   AC: 2
   AD: 3
   AE: 3
   BC: 2
   BD: 2
   CE: 4
   CF: 2
```

```
Stabilization Period Data
  Worst Case: 12
  Average Case: 1.24542
  Best Case: 0
```

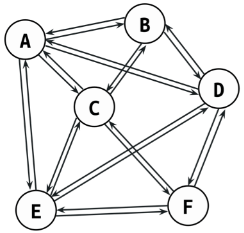
```
All Euler Cycles
  Cycle 1: AE->EC->CE->ED->DF->FD->DA->AB->BC->CF->FE->EF->FC->CA->AC->CB->BD->DB->BA->AD->DE->EA->
```

Results & Conclusion

What results did I get from running 1,000,000 tests on a graph?

[~~Research Background~~ -> ~~Simulation & Programming~~ -> Results & Conclusion -> Questions]

Results: What did I find out?



Stabilization Period Statistics

Locking in 2 or fewer steps	=> 81.01%
Locking in 5 or fewer steps	=> 96.35%
Locking in 10 or fewer steps	=> 99.88%
Average Lock-in Time	=> 1.25 steps

II. Stabilization Period & Euler Cycle

Data that applies to only one graph

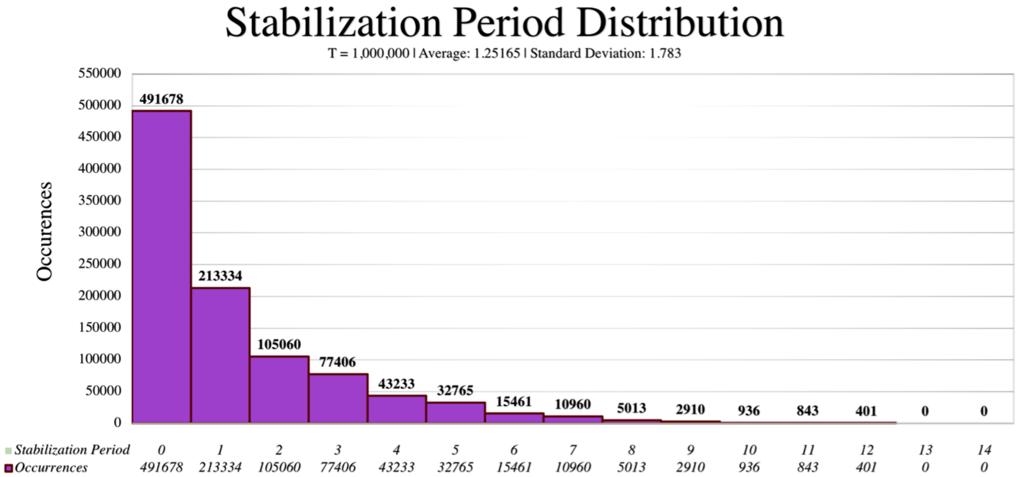
(Stabilization Period)

- Way lower than **44 step** limit.
- Often instantly locks in (49%)
- Random initialization

(Euler Cycle)

- Always different (each test)

Important: There is no way to predict what the stabilization period average by simply looking at a network. **My simulation brought** these hidden statistics to light.



Sample Euler Cycles (*T = 1,000,000*)

C1: CB->BA->AB->BC->CE->EA->AC->CF->FD->DF->FE->EC->CA->AD->DA->AE->ED->DB->BD->DE->EF->FC

C2: FC->CA->AD->DA->AE->ED->DB->BD->DE->EF->FD->DF->FE->EA->AB->BA->AC->CB->BC->CE->EC->CF

C3: DA->AC->CE->EA->AD->DB->BA->AE->EC->CF->FE->ED->DE->EF->FC->CA->AB->BC->CB->BD->DF->FD

C4: AE->ED->DE->EF->FC->CB->BC->CE->EA->AB->BD->DF->FD->DA->AC->CF->FE->EC->CA->AD->DB->BA

Paper 1: Yanovski
Paper 2: Ilcinkas
Paper 3: Menc
Paper 4: Chalopin

Upgrading the Rotor-Router Mechanism

Enhancing the RR Mechanism

(Initialization of a Graph)

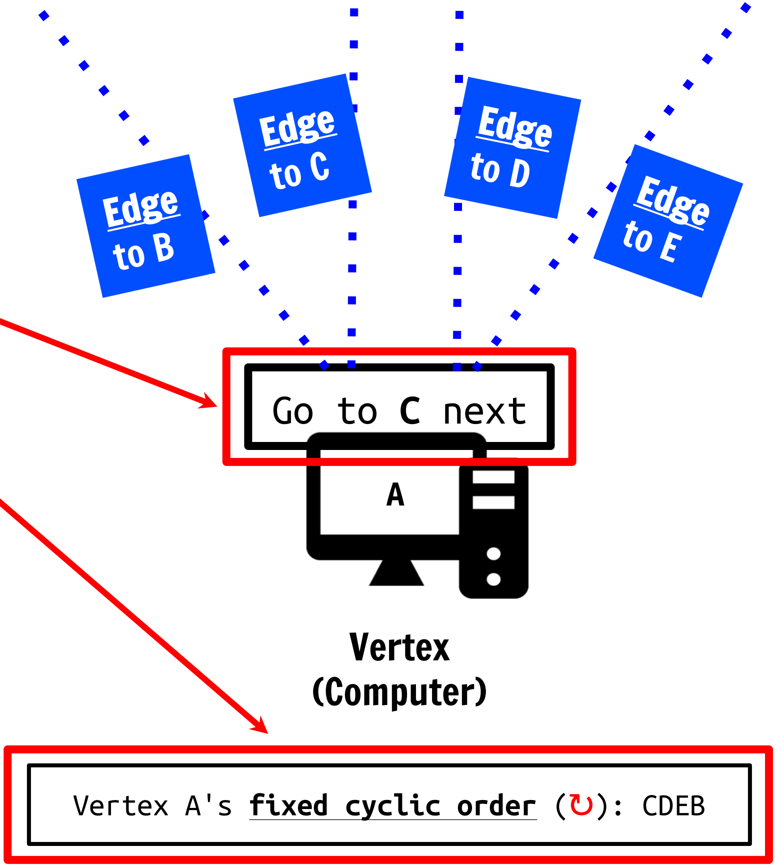
- Customizing the starting value
- Customizing the fixed order

(Extra Additions to the Agent)

- Making the agents **smarter**
- Giving agents a **mind** (memory)

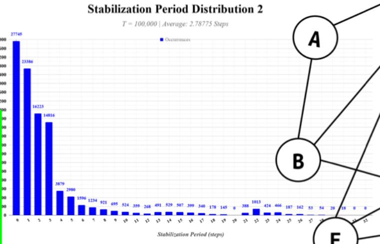
(Increasing the Quantity)

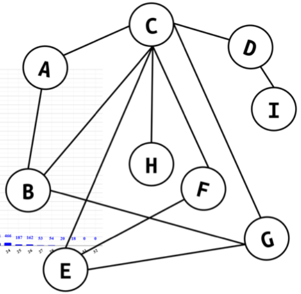
- Multiple Agents in a Graph



Conclusion

```
Stabilization Period Data
  Worst Case: 30
  Average Case: 2.78775
  Best Case: 0
```



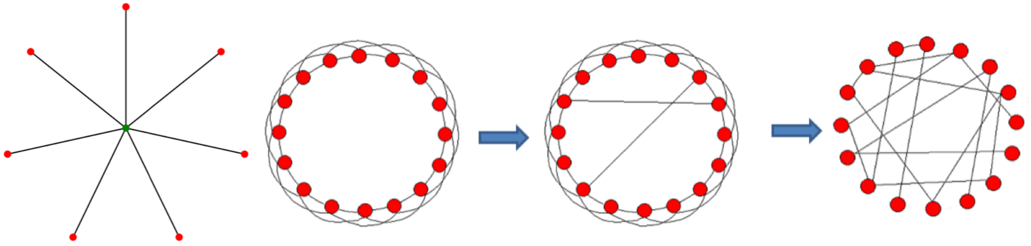


(Thoughts & the Good Parts)

- Seeing the Rotor-Router Mechanism in action = !!
- Writing the simulation from the ground up to obtain data
- Creating animations, figures to understand the algorithm = !!
- Understanding the reliability behind RR, and the details behind mobile agents, networks

(The Further-Research Part)

- Simulations on a variety of graphs (stars, small world, etc)
- Designing a random graph generator that works (!!)
- Investigating stabilization period & more with *multiple agents*.
- Creating a simulation program for newly proposed versions of the Rotor-Router mechanism (!!!!)



(Some) References

- V.A. Pham and A. Karmouch, *Mobile software agents: An overview, IEEE Commun*. Mag., 36(7):26-37, July 1998
- J. Cao and S. K. Das, *Mobile Agents in Networking and Distributed Computing*, *John Wiley & Sons*., 4-16, 2012
- J. Spencer and J. Cooper, *Deterministic Random Walks on the Integers*, *Renyi Institute*, 2-3
- V. Yanovski and I. A. Wagner, *A Distributed Ant Algorithm for Efficiently Patrolling a Network*, *Algorithmica*, 2003
- E. Bampas and L. Gasieniec, *Euler Tour Lock-in Problem in the Rotor-Router Model*, *International Symposium on Distributed Computing*, 423-435, 2009
- D. Ilcinkas, E. Bampas et al, *Robustness of the Rotor-Router Mechanism, Algorithmica*, 78:869-895, 2017
- A. Menc, On the power of one bit: *How to explore a graph you cannot backtrack?*, *Researchgate*, 2015
- H. dai and K. Flannery. *Improved length lower bounds for reflecting sequences*. In *Computing and Combinatorics*, volume 1090 of *Lecture Notes in Computer Science*, pages 56-67. Springer Berlin Heidelberg, 1996.
- A. Borodin, W.L Ruzzo et al, *Lower bounds on the length of universal traversal sequences*, *Journal of Computer and System Sciences*, 45(2):180-203, 1992
- J. Chalopin, S. Das, P. Gawrychowski, A. Kosowski, A. Labourel, and P. Uznanski, *Lock-in Problem for Parallel Rotor-Router Walks*, *Researchgate*, July 2014
- D. J Watts, Small Worlds-*The Dynamics of Networks Between Order and Randomness, Princeton University Press*, Princeton, NJ, 1999

Total: ~140 pages

Questions (?)

From the mechanism itself to how I ran simulations, please feel free to ask me anything!

[~~Research Background~~ -> ~~Simulation & Programming~~ -> ~~Results & Conclusion~~ -> Questions]