An Evaluation of the Rotor-Router Mechanism [Abstract]

Julian M. Rice | *UCLA* | Masuzawa Laboratory | Toshimitsu Masuzawa | *Summer 2018*
Osaka University: *Graduate School of Information Science and Technology*

I. Research Background

The rotor-router mechanism was first introduced as an evolutionary way to robustly patrol networks using mobile bots. Before this mechanism was used, mobile bots were a relatively new concept in networking and cybersecurity and would serve to patrol networks for information protection purposes by randomly choosing the next destination in a network of computers. The rotor-router algorithm, with a simple set of rules, would then be introduced by Yanovski in 2003, which ended up being significantly more robust than the aforementioned random walk. Although the algorithm was robust, many new speed upgrades have been made by modifying the initialization state of both the network and bot, and papers published in 2009, 2015, and 2016 introduced other new details and changes to the algorithm. As a result, the rotor-router mechanism is more efficient than it has ever been.

II. Purpose of Study

The primary motivation for investigating the rotor-router mechanism and network patrolling lies behind finding faster and more efficient ways for communication between computers and servers to communicate information. From inefficiently sending excessive amounts of information between a network to using mobile bots that reduce the load on a network by following a set of established rules, network patrol and data transferring efficiency has been improved by a great deal over the last few decades. Even with evolutionary rotor-router mechanism introduced in 2003, there has always been space for improvement and faster times. I take a look at some of the improvements made to the rotor-router mechanism over the last decade, run a few simulations of my own using the original 2003 mechanism, and discuss what simulations and further research should be done on newer mechanisms to further increase network patrolling efficiency.

III. Methods

The first portion of my paper covers the rotor-router mechanism and a background on its uses and mobile agents. This portion then looks specifically at the algorithm and explains each step in detail. The time evaluation for how efficient the algorithm is also explained and simulation results from Yanovski 2003 are also summarized.

The second portion of the paper looks at my own implementation of the rotor-router algorithm using C++, and contains analysis of the simulation results I obtained from the program using the graph in Figure 1.

The last portion of the paper looks at other more recent papers that have made changes, additions to the rotor-router mechanism for an increase in robustness. After evaluating the upsides and downsides of each new addition, I finally discuss the value of adding more bots into a network and what research, tests, and simulations should be done in the near future.

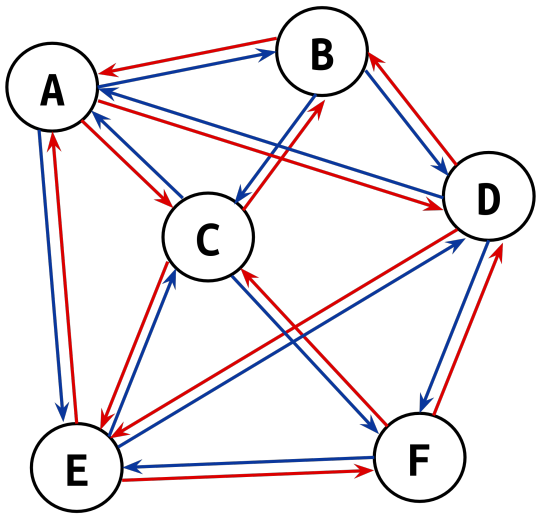


***Figure 1**: Simulation Graph*

IV. Results

As seen in Figure 1, there are 6 vertices (A, B, C, D, E, F) and 11 edges (AB, BC, AC, CF, etc.), and the program runs through various numbers of steps to calculate **[1]** the average times each vertex is visited and **[2]** the average times each edge is visited (by the bot). The last detail is **[3]** the number of steps it takes for the bot to end up in a stable sequence of movement (**stabilization p.**).

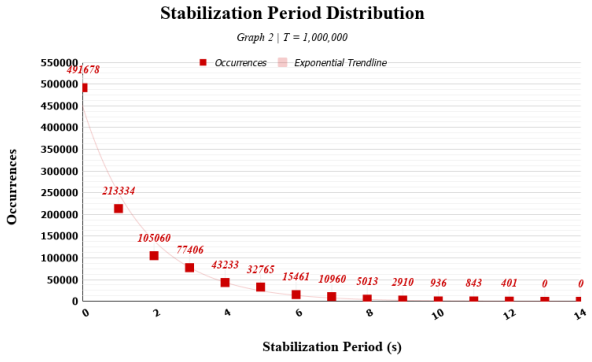
The simulation was run 10000 times for each separate amount of steps to ensure accuracy. **Table 1** and **Table 2** show the average visit data for vertices and edges when Steps **S** is equal to 100, 500, and prime number 2203.

T = 10000	Vertices					
Steps (S)	*A*	*B*	*C*	*D*	*E*	*F*
100	18.1829	13.6278	18.1900	18.1899	18.1805	13.6289
500	90.9084	68.1737	90.9162	90.9163	90.9103	68.1751
2203	400.545	300.400	400.553	400.553	400.546	300.402

***Table 1**: Average Visits (Vertex)*

T =10000	Edges					
Steps (S)	*AB*	*AC*	*AD*	*AE*	*BC*	*BD*
100	9.0951	9.0844	9.0846	9.0832	9.0973	9.0976
500	45.456	45.449	45.447	45.446	45.460	45.458
2203	200.274	200.267	200.264	200.264	200.278	200.276
Steps (S)	*CE*	*CF*	*DE*	*DF*	*EF*	
100	9.0853	9.096	9.0855	9.0962	9.0948	
500	45.450	45.463	45.448	45.461	45.460	
2203	200.269	200.282	200.267	200.280	200.279	

***Table 2**: Average Visits (Edge)*



***Graph 1**: Stabilization Period Distribution*

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
C5: FD->DA->AB->BD->DB->BA->AC->CF->FE->EA->AD->DE->EC->CA->AE->ED->DF->FC->CB->BC->CE->EF
C6: AE->EC->CB->BC->CE->ED->DE->EF->FC->CF->FD->DF->FE->EA->AB->BD->DA->AC->CA->AD->DB->BA
C7: DF->FE->EA->AC->CA->AD->DA->AE->EC->CB->BA->AB->BC->CE->ED->DB->BD->DE->EF->FC->CF->FD
C8: BA->AB->BC->CE->ED->DB->BD->DE->EF->FC->CF->FD->DF->FE->EA->AC->CA->AD->DA->AE->EC->CB
C9: DF->FE->EA->AE->EC->CB->BA->AB->BC->CE->ED->DA->AC->CF->FC->CA->AD->DB->BD->DE->EF->FD

***Table 3**: 9 Sample Euler Cycles*

A near equal amount of visits for edges and vertices (ACDE, BF) demonstrates the accuracy and reliability of the rotor-router mechanism. The stabilization period, or time it takes for the bot to lock into a forever repeating cycle, for **Figure 1** was then simulated one *million* times, with the results displayed in **Graph 1**.

Finally, I recorded the first nine (out of a million) cycles that repeat forever in **Table 3**. Each cycle starts at a vertex (ABCDEF) and visits every single edge once. This occurs after the stabilization period has ended and loops indefinitely. For example, at the end of **C1**, the edge (line) between F and C will have been traversed, and the next line to traverse will be the edge between C and B.

V. Conclusion

The second half of the paper looks specifically at changing the initialization values of each vertex and how to decrease the maximum limit for the stabilization period. In my simulation, the original rotor-router algorithm is used, and therefore a limit of O(*m**D), where *m* is the number of edges (lines, in this case, 22) and D is the **diameter** (longest *shortest path* from one vertex to another, in this case, 2), but this can be reduced to up to O(*m*) given the most optimal initialization settings.

The results obtained from the simulations that have been executed, and the analysis of different variations of the rotor-router mechanism have shown its effectiveness, and how inefficient a random walk can be. For a simple unchanging rule like the fixed ordering seen in the rotor-router mechanism to be this efficient, one can only imagine how much faster network patrolling can be achieved in the near future.