Report on Implementation of Black Hole Search in Rings

Referencing “*Time Optimal Algorithms for Black Hole Search in Rings*

*By: B. Balamohan , P. Flocchini , A. Miri , and N. Santoro”*

Hieu Nguyen

University of Ottawa

7234520

*https://github.com/hnguy028/BHS\_InRings*

**Abstract.** In the field of network exploration, black hole search is a problem where mobile entities or agents are set to find the location of a black hole – the so-called black hole is a node(s) which can be considered harmful to the users of the network. The black hole is such a node in the network that destroys incoming agents and/or message without any notification to other agents. There is a myriad of variants to the black hole search problem, and in this report, we discuss the implementation of three such algorithms in rings.

**Introduction.** In the referred paper multiple algorithms are discussed, of which two were implemented.

The first algorithm is a trivial one, which consists of two agents both of which perform cautious walk if they can return and mark the edge as “SAFE”, they return to the home base and marks it on the whiteboard. At each new safely explored edge, they must return to the home base. This can be seen to have at worst case moves for the agent and therefore an asymptotic complexity of O(n2). This algorithm is used as a means to measure the other implemented algorithms.

The second algorithm is “OptAvgTime”. Which works by pairing every node with two agents, both of which explore the ring from either side excluding the node itself, and the two pairs that do return, will determine the location of the black hole. This algorithm solves the black hole search problem in worst ideal time complexity of .

The third implemented algorithm is “OptTeamSize”. This algorithm uses only two agents and works by each agent exploring disjoint sections of a remaining unexplored node set. During the algorithm the agents alternate between two exploration strategies, big and small which determine their respective exploration areas. Since the explored areas are disjoint, only one agent will ever be exploring the area containing the black hole. This algorithm solves the black hole search problem in average ideal time complexity of .

**Design.**

All algorithms were designed bottom-up using pure Java; aside from the basic Util library the ArrayList and HashMap data structures which were used for expressing the white board, and the Random library used for generating random asynchronous “wait” times. The asynchronous aspect of the algorithms are simulated by applying a randomly generated “wait” time, for each agent while traversing an edge.

The graph designs follows the standard definition for nodes and edges, only on top of which data is defined based on the different requirements for each algorithm. For example, in the worstCase algorithm only integer information is needed to be stored on the white boards, however for OTS (OptTeamSize) it requires a little more. Such as numeric information to update the explored space and instructions to initiate role change. As such, each sub class will define its own whiteboard per algorithm.

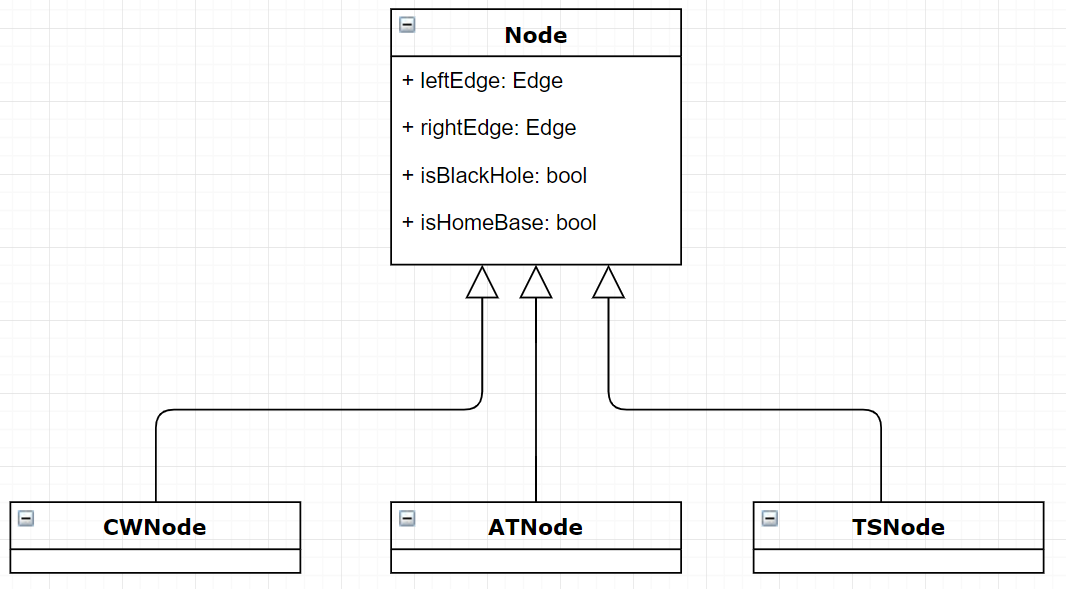


Figure 1. Depicting Node Class structure.

The majority of the decisions/instructions are within the Agent class, which determines its next action from its current state. The implemented algorithms work on a loop where each agent performs its respective actions given the current state and continues until a termination condition has been reached.

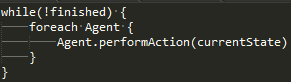


Figure 2. Basic skeleton of loop implemented for the algorithms

**Experimental Results.**

|  |  |  |  |
| --- | --- | --- | --- |
| ***Ring Size*** | ***Worst Case Cautious Walk*** | ***Optimal Team Size*** | ***Optimal Average Time*** |
| *1,000* | *148,345,044* | *2,469,529* | *144,697* |
| *3,000* | *1,341,631,140* | *7,016,018* | *847,549* |
| *5,000* | *3,972,181,353* | *13,022,150* | *2,085,795* |
| *7,000* | *13,444,512,210* | *19,522,165* | *4,058,971* |

Table 1. Depicting the runtime of average actual runtime of each algorithm on different ring sizes in Nano seconds

Table 1 summarizes the average actual run times of each algorithm on different ring sizes, over 100 simulations. Since we are simulating algorithms that are designed for distributed systems, we will normalize the data to the number of agents. We divide the average total elapsed time of the algorithms with their respective number of agents, to find the average run time of the algorithm:

|  |  |  |  |
| --- | --- | --- | --- |
| ***Ring Size*** | ***Worst Case Cautious Walk*** | ***Optimal Team Size*** | ***Optimal Average Time*** |
| *1,000* | *584,068* | *5,992* | *1,495* |
| *3,000* | *5,436,634* | *17,992* | *4,428* |
| *5,000* | *14,203,753* | *29,992* | *7,260* |
| *7,000* | *28,973,230* | *41,992* | *10,504* |

Table 2. Depicting the runtime of average ideal runtime of each algorithm on different ring sizes in ideal time units

Table 2 summarizes the average ideal times of each algorithm on the sample ring sizes, over 100 simulations. Each simulation is run on randomly generated black hole and home base locations. Comparing the ideal run times of each algorithm we can see that it follows the trend of the asymptotic complexity presented in the referred paper.

Figure 3. Depicts the average actual runtime of each algorithm as a function of ring size.

Since the Worst-case Cautious walk algorithm extremely overshadows the results of the remaining algorithms, we will omit it, so we can see how the run times of those algorithms relate.

Figure 4. Depicts the average actual runtime of each algorithm as a function of ring size, representation of figure 3, while omitting WCCW algorithm

Figure 5. Depicts the average ideal time of algorithm OTS and OAT, while omitting WCCW.

We see in figure 5, the function of average ideal time by increase in rings size has a more linear representation that more closely resembles the algorithms asymptotic complexity, compared to the normalized actual times depicted in figure 4. It can be argued that this is due to some hidden calculations that are counted in the actual runtime as part of the machine itself.

Figure 6. Normalized runtime of each algorithm in Nano seconds, on a ring of 1000 nodes.

Figure 6 shows a slice of the above graphs at a ring size of 1000, viewing the same graph at a higher ring size will result in the WCCW overshadowing the other two algorithms. We see that the relative relation of the algorithm runtimes follows the correct trend, in terms of OAT resulting in the optimal runtime, and WCCW be the worst.

From the experimental results we see that the run time of *Algorithm “OptAvgTime”* takes a significant amount of time compared to the other algorithms. This can be explained by the experiments being run locally on a single machine, therefore all agent movements and decisions are calculated and processed as part of the runtime. Once we normalized the runtimes, we can then see that the time then begins to follow the expected trend of its theoretical asymptotic time complexity.

Now when we look at the experimental runs counting ideal time; where each edge traversal takes only *“1-time unit”,* we can see that this more closely conforms to the order of ideal time complexities defined in the paper. We see that as the reference point, the worst-case algorithm should indeed be the worst case in both ideal time and normalized actual time. Both the OTS and OAT algorithms also follow suit in ideal time, as a slight curve in the actual time.

**Conclusion.**

From our experimental results, the relative ideal times of each algorithms runs hold with the asymptotic complexities presented in the paper. The actual times show a slight increasing curve as the number of nodes grow, this however can be attributed to a hardware fault or limitation. In relation to each other, as one would expect the *“OptAvgTime”* does win over both worst-case and OTS in both ideal time and actual time. In terms of ideal time all three algorithms also, the result show that they do conform slightly if not exactly to their respective complexities defined in the paper, though the minor flaws can be attributed implementation flaws and possible areas that could be optimized in terms written code and data structures used.