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 an asymptotic complexity of $O(n^2)$. This is used as a means to measure the other implemented algorithms.

The second algorithm is “OptAvgTime”. This algorithm works by pairing every node with two agents, which explore the ring from either side excluding the node itself, and the two pairs that do return define the black hole.

The third algorithm is “OptTeamSize”. This algorithm uses only two agents and works by each agent exploring disjoint sections of the remaining unexplored node set. The agents alternate between two exploration strategy big and small which determine their exploration area.

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.

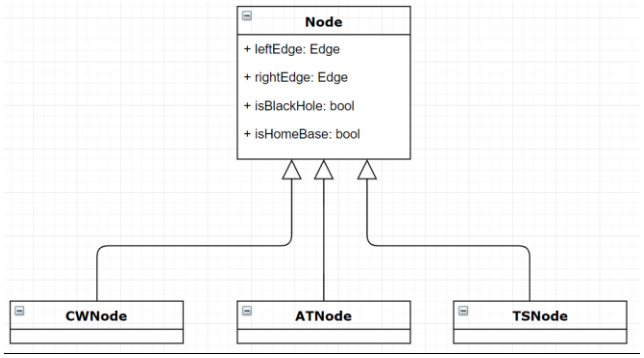

Figure 1. Depicting Node Class structure.

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.

```
while(!finished) {
    foreach Agent {
        Agent.performAction(currentState)
    }
}
```

Figure 2. Basic skeleton of loop implemented for the algorithms

Experimental Results.

Algorithm	Average Time (in milliseconds)
Worst Case Cautious Walk (WCCW)	852
Optimal Average Time (OAT)	11947.5
Optimal Team Size (OTS)	17

Table 1. Average actual run times (in milliseconds)

Table 1 summarizes the average actual run times of each algorithm on a ring size of 5005 nodes, over 100 simulations.

Algorithm	Average Time (in 1 ideal time unit)
Worst Case Cautious Walk (WCCW)	13635657
Optimal Average Time (OAT)	6920
Optimal Team Size (OTS)	30024

Table 2. Average ideal time per each algorithm

Table 2 summarizes the average ideal times of each algorithm on a ring size of 5005 nodes, over 100 simulations. It can be seen that the order given from the ideal time, follows that of the ideal time complexities of the referenced paper.

Discussion.

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. As such for algorithm *"OptAvgTime"* which requires *n-1* agents, the run time will increase proportionally to the size of the ring in our experiments. However when comparing *"OptTeamSize"* and the worst case algorithm, we can see that since they both only require 2 agents, the actual time can be seen to follow the ideal time order of the algorithms, in the sense that *"OptTeamSize"* should do better.

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 does conform 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

ideal time, and the *"OptAvgTime"* should resolve to be the better algorithm on our random execution set; as shown in Table 2.

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 however can only compare the worst-case algorithm and the *"OptTeamSize"* as they both share the number of agents as a control variable. As one would expect the *"OptTeamSize"* does win over the worst-case in both ideal time and actual time. In terms of ideal time for all three algorithms, they result as projected from the referred paper, with *"OptAvgTime"* beating out *"OptTeamSize",* and worst-case coming in last.