# ASSIGNMENT-1 REPORT

By: Kirtan Khichi

# Comparative Analysis of Search Algorithms on a Randomly Connected City Graph

I have taken 5 pairs of nodes mentioned below to analyse each Algorithm on a random connection city graph.

* Total number of nodes in the complete graph: 45
* Total number of edges in the complete graph: 990
* Total number of nodes in the random connected graph: 45
* Total number of edges in the random connected graph: 96
* Graph behaviour: Undirected Graph.
* Dropout chosen for random connected graph: 0.9

**Algorithm analysis: -**

I have taken these 5 pairs for the analysis of the DFS and BFS algorithms:

1. Patna to Rajsamand
2. Calicut to Arrah
3. Hanamkonda to Patna
4. Jodhpur to Belagavi
5. Lucknow to Jodhpur

Case 1: For the starting node Patna and target node Rajsamand.

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **Nodes** | **Algorithms** | **Elapsed Time (in ms)** | **RAM Used** | **Path length** | **Path found** | **Remarks** |
| **Patna**  **to**  **Rajsamand** | DFS | 0.8323 | 184 | 9 | yes | DFS is the fast from the BFS in this case, although DFS have a greater path length. BFS and DFS uses same RAM but, BFS finds the shortest path. A\* uses the more amount of RAM and GBFS taking less time than A\*. Here we can see that RAM in DFS is greater than double value of RAM in GBFS but the time Elapsed is not much differ. |
| BFS | 0.9344 | 184 | 3 | yes |
| GBFS | 0.8178 | 88 | 3 | yes |
| A\* | 0.9357 | 120 | 3 | yes |

Case 2: For the starting node Calicut and target node Arrah.

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **Nodes** | **Algorithms** | **Elapsed Time (in ms)** | **RAM Used** | **Path length** | **Path found** | **Remarks** |
| **Calicut**  **To**  **Arrah** | DFS | 1.1378 | 376 | 34 | yes | As we can see that in case of BFS we have highest RAM among all and if we compare RAM of BFS with RAM of GBFS then it is just 2.5 times greater but the elapsed time 0.2ms. A\* is taking so much time if see according to RAM, A\* has only 120 RAM and GBFS have 184 but still GBFS is fast then A\*. |
| BFS | 0.9780 | 472 | 7 | yes |
| GBFS | 0.7434 | 184 | 11 | yes |
| A\* | 0.8943 | 120 | 5 | yes |

Case 3: For the starting node Hanamkonda and target node Patna

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **Nodes** | **Algorithms** | **Elapsed Time (in ms) \*** | **RAM Used\*** | **Path length** | **Path found** | **Remarks** |
| **Hanamkonda**  **To**  **Patna** | DFS | 1.1474 | 376 | 35 | yes | As we can see in the case of DFS and BFS Ram is same but the path length differs and the time is also varying so much. In this case BFS is more efficient than DFS. GBFS is taking less elapsed time than A\* but, RAM is more in the GBFS not in A\*. From this we can say that GBFS is faster than A\*. |
| BFS | 0.8671 | 376 | 9 | yes |
| GBFS | 0.8408 | 184 | 11 | yes |
| A\* | 0.8816 | 120 | 5 | yes |

Case 4: For the starting node Jodhpur and the target node Belagavi

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **Nodes/Properties** | **Algorithms** | **Elapsed Time (in ms)** | **RAM Used** | **Path length** | **Path found** | **Remarks** |
| **Jodhpur**  **To**  **Belagavi** | DFS | 0.8735 | 120 | 11 | yes | As we can see BFS is using much larger RAM than DFS and BFS has higher elapsed time than DFS but, DFS have longer path length than BFS. GBFS is stuck at dead end and taking 0.87 ms time but A\* gave the path and takes less time. From this we can say that A\* is complete algorithm but GBFS is not. |
| BFS | 0.8951 | 472 | 7 | yes |
| GBFS | 0.8713 | 248 | 0 | no |
| A\* | 0.8375 | 120 | 6 | yes |

Case -5: For the starting node Lucknow and target node Jodhpur

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **Nodes** | **Algorithms** | **Elapsed Time (in ms)** | **RAM Used** | **Path length** | **Path found** | **Remarks** |
| **Lucknow**  **To**  **Jodhpur** | DFS | 0.8398 | 472 | 45 | yes | As we can see DFS is using much larger RAM than DFS but, still DFS has less time elapsed than BFS. GBFS is also taking more RAM than A\* but time elapsed is lesser in case of GBFS than A\*. |
| BFS | 0.9930 | 88 | 2 | yes |
| GBFS | 0.8717 | 248 | 0 | no |
| A\* | 0.8852 | 120 | 4 | yes |

# The below table shows the average time taken and memory taken by different algorithms for all the test cases.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Properties** | **Algorithms** | **Elapsed Time (in ms)** | **RAM Used** | **Remarks** |
| **Average** | DFS | 0.96616 | 305.6 | On average, GBFS is the fastest and also using more less RAM than algorithm. BFS finds the shortest paths but is slower than DFS and uses the most RAM. GBFS and A\* use the least RAM, with GBFS being faster than A\*. |
| BFS | 0.97352 | 318.4 |
| GBFS | 0.829 | 190.6 |
| A\* | 0.88686 | 120 |

**PROS AND CONS ANALYSIS:**

**Pros of DFS:**

* This algorithm uses less memory as compared to BFS because it traverses in the one branch as deep as possible along then backtrack. As you can see in the case1 and case4 length of the visited list is equal to length of the path list.
* DFS is an efficient algorithm to find a path in the connected graph.

**Cons of DFS:**

* DFS is not an optimal algorithm so it will not guarantee the shortest path.
* Some time to find the path it can traverse the big part of the graph. Example in case5 we have traversed the whole graph to find the path.

**Pros of BFS:**

* BFS is guaranteed to give the optimal path between two nodes.
* In BFS we can say that the path from A to B is the same as path from B to A but in the case of DFS we can’t say.
* Example: - In case5 we have visited 45 nodes to find the path between nodes in the case of DFS but in case BFS we have to search for 3 nodes only.

**Cons of BFS:**

* BFS is inefficient when our nodes are deep inside the graph. Example in case4, we have visited the whole graph level to reach the target node.

**Pros of GBFS:**

* In this algorithm we never do backtracking so it is an efficient algorithm and takes less time.
* In this algorithm the length of visited list and path list is always equal.
* This algorithm has only one path, not a bunch of paths.

**Cons of GBFS:**

* This algorithm is not complete; it can be stuck at a dead end. For Example, in case2 we are stuck at a dead end and we have found Morena as dead end.
* Heuristic function should be admissible. If the heuristic function is not accurate then it might lead to a non-optimal solution.

**Pros of A\* Star:**

* This algorithm is complete and always gives an optimal path.
* If we check the path from A to B and the path from B to A is always similar.
* It never stuck at a dead end.
* The traversal is guided by a Heuristic function that helps to converse to the optimal path. We can see that the number of visited nodes may be high but the path has a smaller number of nodes in all the cases.

**Cons of A\* star:**

* This algorithm is very expensive, a time and space taking algorithm.