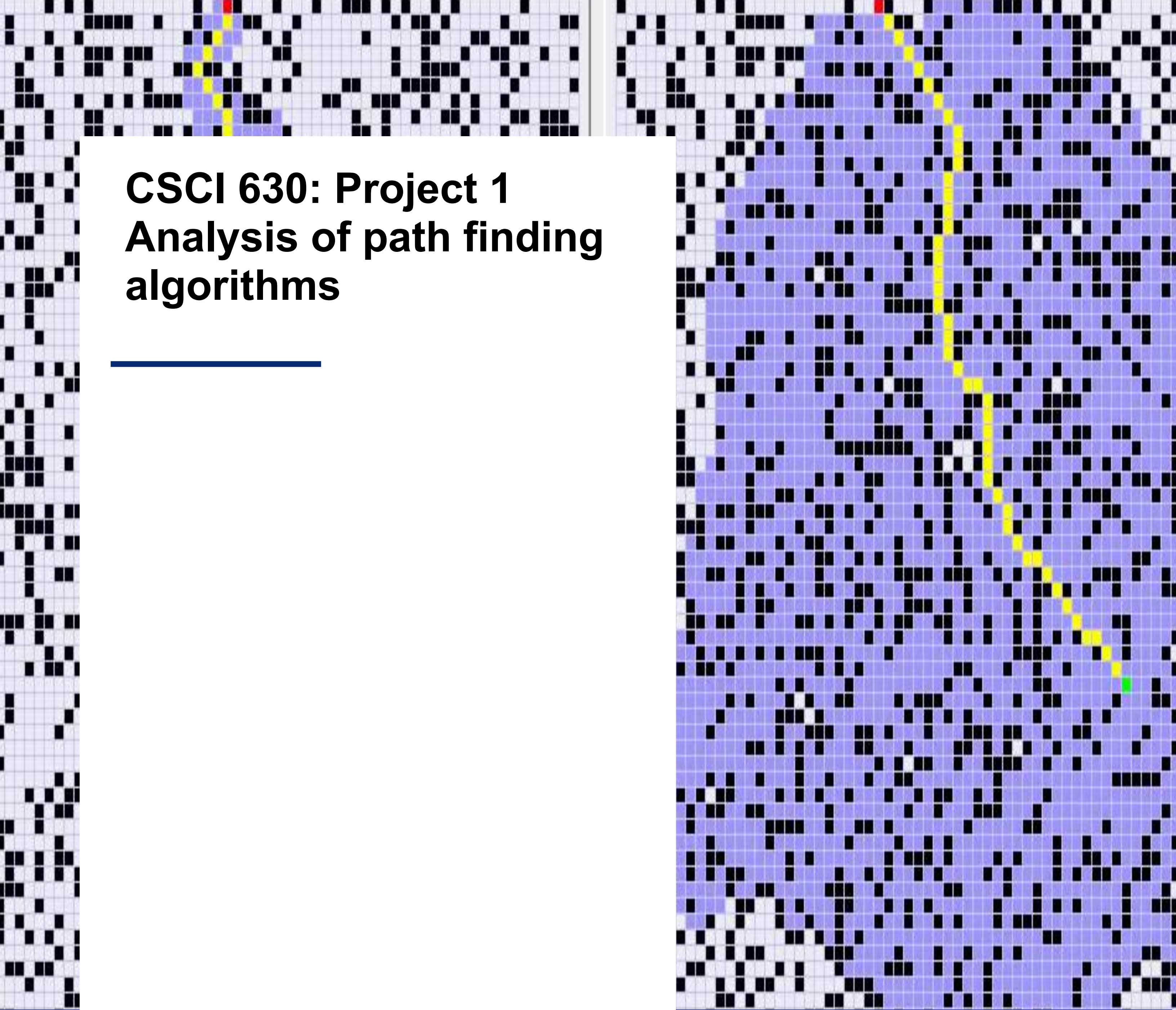


CSCI 630: Project 1 Analysis of path finding algorithms

FEBRUARY 11

Foundations of Intelligent System
Authored by: Aditi Munjekar

INDEX

Problem Statement

Implement Iterative Deepening Depth First Search (IDS) and A* using four heuristics.

State-space description

States: It is represented by the grid who's each cell contains 0 and 1s. The cells which contain 0 are the locations which can be visited.

Initial state: (0,0) location on the maze

Actions: In this maze solving environment, there are four actions for each state: *Right, Left, Up and Down*. The algorithm cannot proceed to any cell on the maze which has 1 in it. It can only go to those cells which have 0 in their locations.

Transition model: Returns the new location on the maze. The new location is decided by the algorithm.

Goal test: This checks whether the current state matches the goal state. For example: for a 40x40 maze, the initial position is the (0, 0) and the goal position is (39, 39). In this environment, the goal test is if the algorithm has reached (39, 39) position on the maze.

Path cost: Each step costs 1, so the path cost is the number of steps in the path to reach the goal.

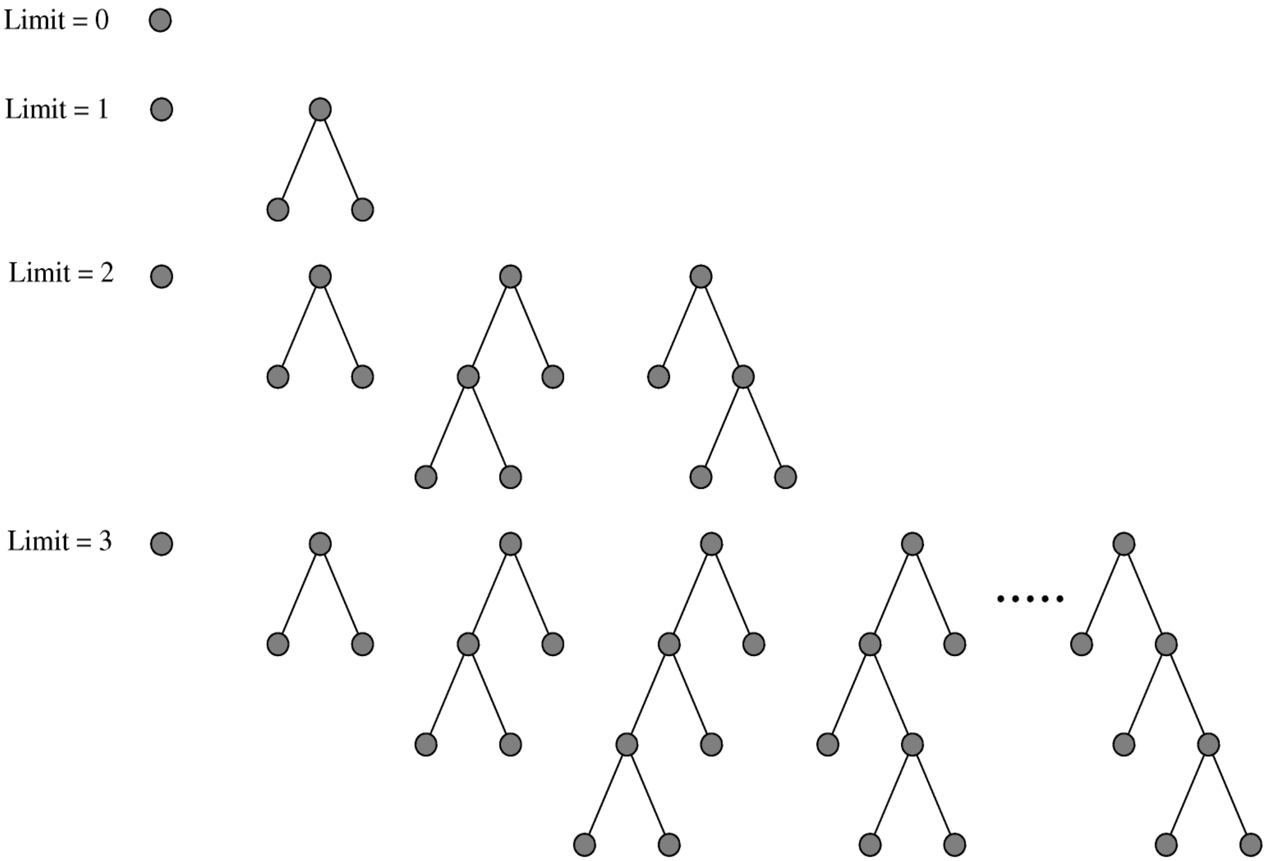
Theory

Iterative Deepening Depth First Search:

- It is a hybrid of BFS and DFS
- BFS and DFS fail in the following aspects:

  1. In DFS, we recursively look at a node's adjacent vertex. DFS may not end in an infinite search space. Also, DFS may not find the shortest path to the goal. DFS needs O(d) space, where d is the depth of search.
  2. BFS consumes too much memory. BFS needs to store all the elements in the same level. In case of a tree, the last level has N/2 leaf nodes, the second last level has N/4. So, BFS needs O(N) space.

- An improvement over BFS and DFS is IDS which implements BFS at a particular depth.



Thus, at depth 0 only one node is visited. At depth 1, the node at depth 0 as well as at 1 will be visited and so on…

A* algorithm:

- Path-finding intelligent algorithm
- It is called intelligent because it uses heuristics which is the distance of the current node from the goal
- A* uses the least cost f(n) which is calculated as: f(n) = g(n) + h(n)

  where:

  g(n) = the movement cost to move from starting point to current node

  h(n) = estimated cost from current node to the goal using any of the following heuristics: Euclidean, Manhattan and random

1. Manhattan distance: The sum of absolute values of differences in the goal's x and y coordinates and the current cell's x and y coordinates respectively:

   abs (current_node.x - goal.x) + abs (current_node.y - goal.y)

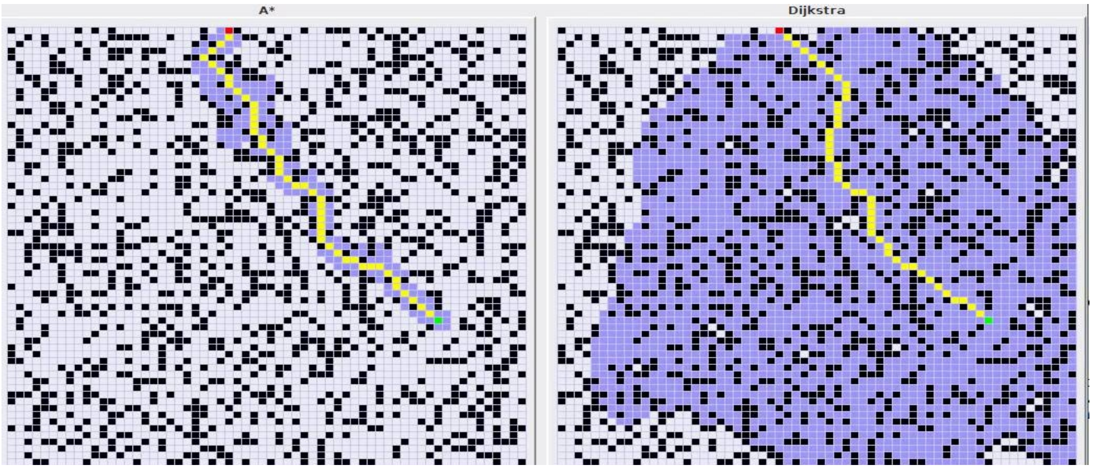
2. Euclidean distance: It is the distance between the current cell and the goal cell using the distance formula:

   sqrt {abs (current_node.x - goal.x)**2 + abs(current_node.y - goal.y)**2}

3. Random distance: This function will return any random value between the given range

For a clear understanding of how A* works, following is the comparison between

A* and Dijkstra's (no heuristics):



Admissible heuristics:

An admissible heuristics is the one that never overestimates the cost to reach the goal. As g(n) is the actual path to reach the current from the start and f(n) = g(n) + h(n), f(n) would never overestimate the true cost of a solution as it already has the heuristics added to it.

1. Manhattan heuristic: For our maze, in which we are not allowed to move diagonally, the Manhattan heuristic is the upper bound for the admissible heuristic. All the heuristics below this heuristic is admissible but above this is not admissible.
2. Euclidean heuristic: For our maze, in which we are not allowed to move diagonally, the Euclidean heuristic will be admissible. We will get the shortest path in this heuristic.
3. Random heuristic: For our maze, in which we are not allowed to move diagonally, this heuristic may or may not be admissible. It depends on the random value which is generated.
4. Constant heuristic: For our maze, in which we are not allowed to move diagonally, this kind of heuristic is admissible but it is not consistent.

Working of the code:

A*:

A priority queue is maintained which contains all the neighbors of a particular cell on the maze. Graph is the basic data structure for this algorithm which maintains a dictionary containing the vertices of the cells and their costs. First, the start and the end vertices are added in the graph. After that the start item is also added in the queue. While the queue is not empty the neighbors of the current node are explored, while checking if it is on a walkable terrain. The child node is added on the graph with it's cost. Then a child node with a better cost is removed from the queue.

IDS:

This is a recursive algorithm which goes upto a certain depth and explores all nodes on that depth. If it finds the goal on a particular level before reaching the maximum depth then it returns the number of nodes explored. It explores all the nodes on the walkable terrain. The DLS () function is called recursively on a particular depth.

Performance Metrics:

For A*: The performance metric for A* can be summarized in the following tables. The tables record the nodes generated for every heuristic.

Heuristic: Constant

Dataset size	Maze no.	Nodes generated	Nodes visited	Path length	Branching factor
Small	1	2442	921	79	1.103
Medium	2	178395	59480	603	1.02
Medium	3	178029	59430	599	1.02
Small	4	1386	527	81	1.09

Heuristic: Manhattan

Dataset size	Maze no.	Nodes generated	Nodes visited	Path Length	Branching factor
Small	1	2246	838	79	1.102
Medium	2	178386	59476	603	1.02
Medium	3	178028	59429	599	1.02
Small	4	974	369	81	1.08

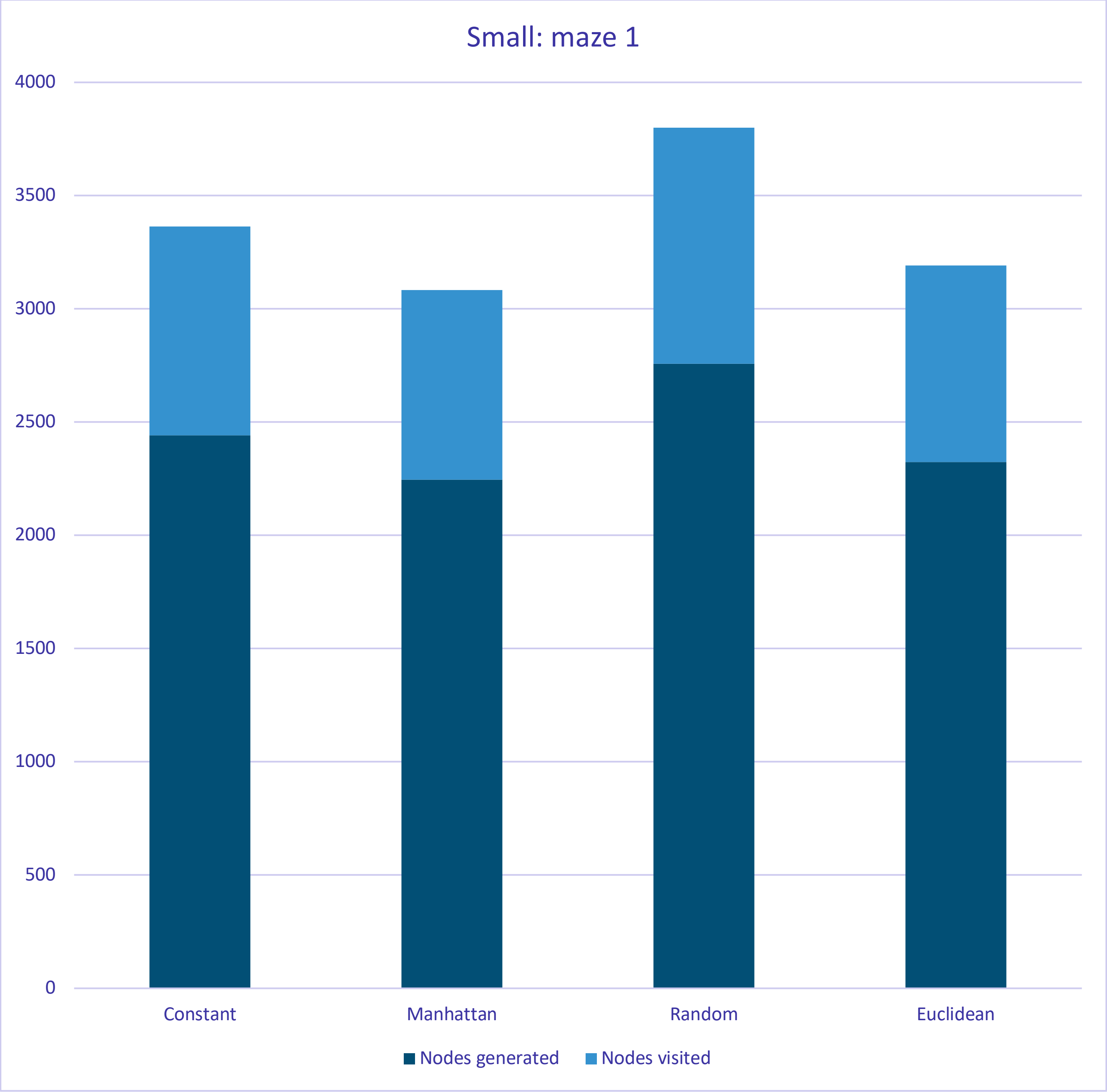
Heuristic: Random

Dataset size	Maze no.	Nodes generated	Nodes visited	Path length	Branching factor
Small	1	2757	1043	81	1.102
Medium	2	193458	64751	603	1.02
Medium	3	216430	72615	599	1.02
Small	4	3358	1276	81	1.105

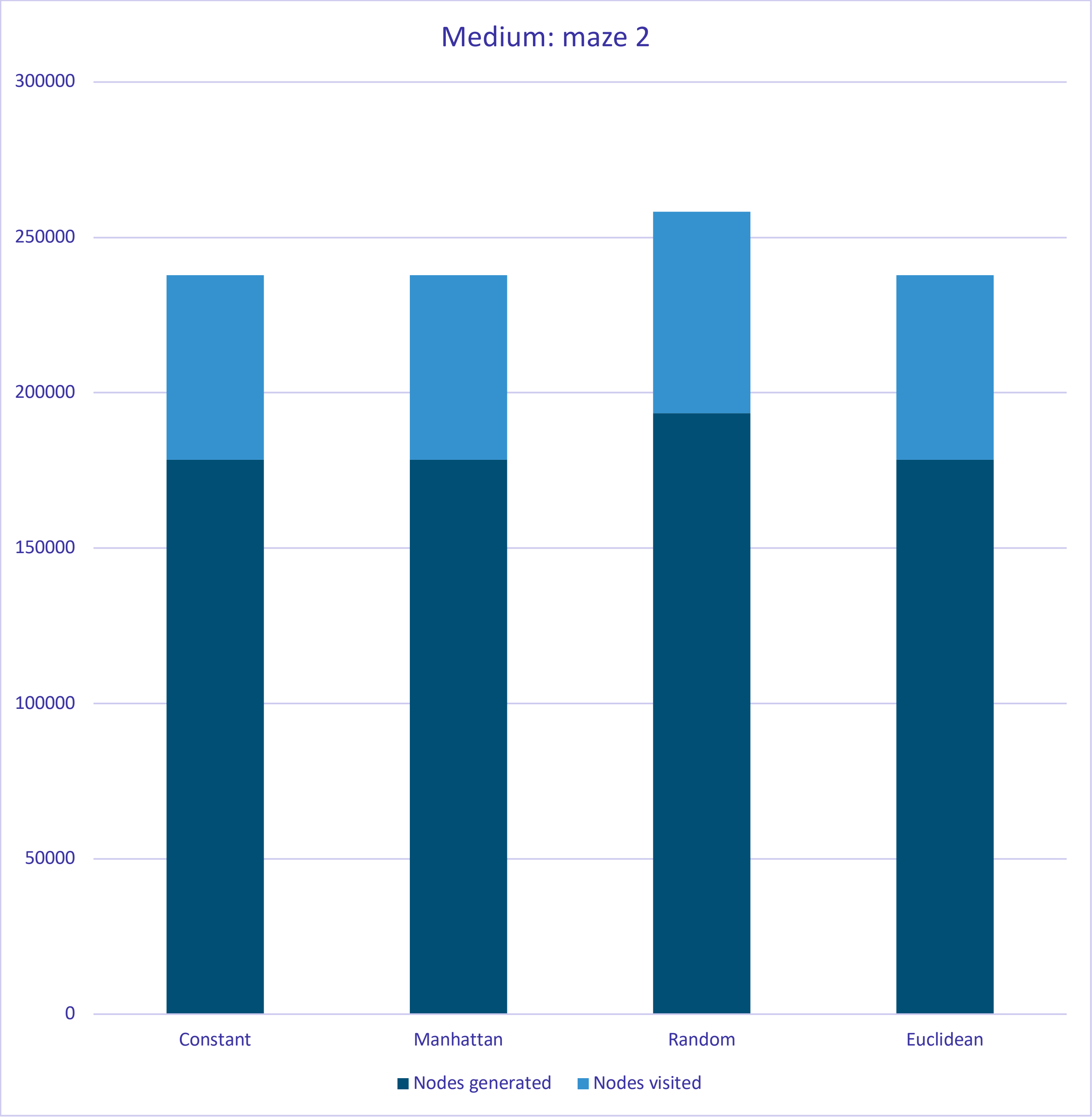
Heuristic: Euclidean

Dataset size	Maze no.	Nodes generated	Nodes visited	Path length	Branching factor
Small	1	2323	868	79	1.103
Medium	2	178395	59480	603	1.02
Medium	3	178028	59429	599	1.0203
Large	3	17014930	5786551	6027	1.002

The comparison between nodes generated and visited for maze **1** in **Small** dataset :

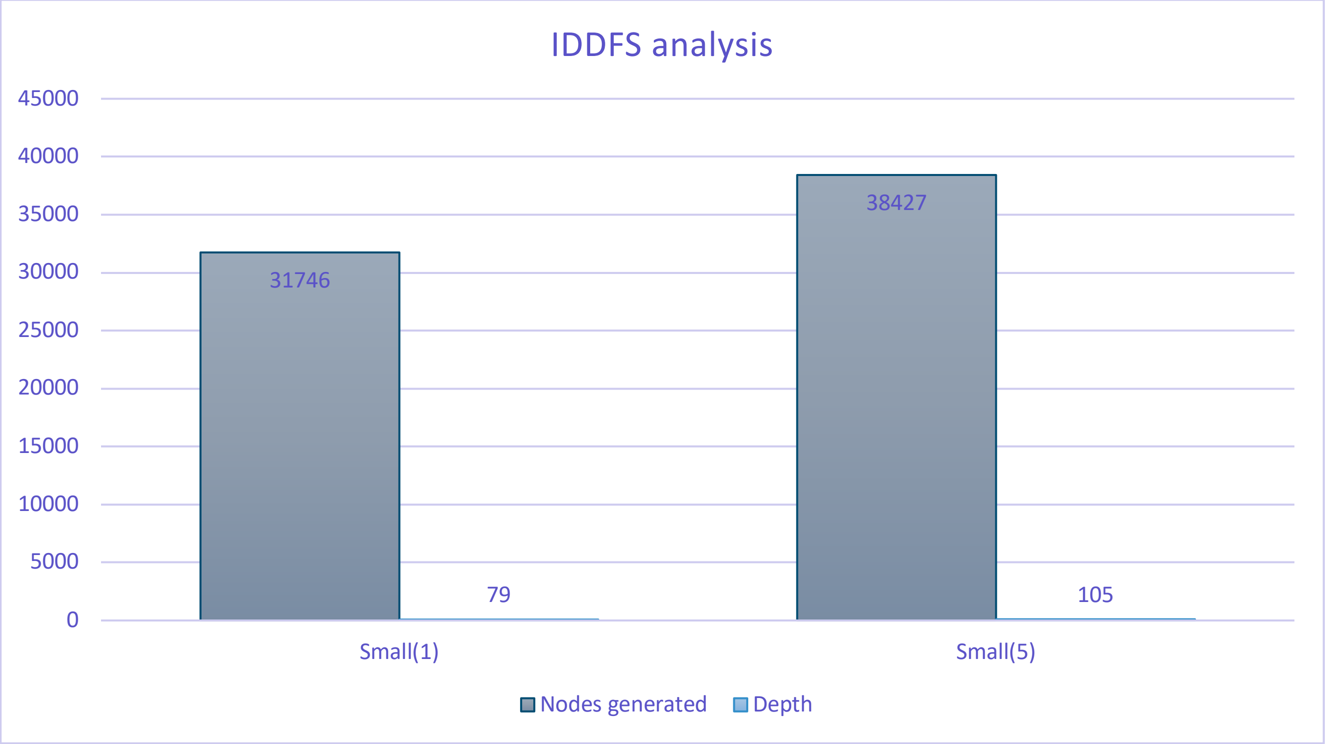


The comparison between nodes generated and visited for **maze 2** in **Medium** dataset :



For IDS:

Dataset size	Maze no.	Nodes generated	Depth	Branching factor	Goal reached?
Small	1	31746	79	1.14	Yes
Small	5	38427	105	1.105	Yes
Medium	2	--------------	236(in 5 minutes)	-------------	Not in 5 minutes
Medium	3	--------------	235(in 5 minutes)	-------------	Not in 5 minutes
Large	2	--------------	240(in 5 minutes)	------------	Not in 5 minutes
Large	3	--------------	242(in 5 minutes)	-------------	Not in 5 minutes



Conclusion:

- A* being an informed search algorithm is very fast as compared to iterative deepening depth first search (IDDFS) which becomes very slow for a large dataset.
- Since in our algorithm, we are not allowed to move diagonally, Manhattan heuristic performs the best taking the least number of nodes to reach the goal for A* algorithm.
- Manhattan heuristic is followed by the constant, Euclidean and Random heuristics.
- The IDS algorithm becomes very slow for the increasing size of the dataset