Shortest Path Visualisation using A* Algorithm

Formal Definition:

A* algorithm A* (pronounced "A-star") is a **graph traversal and path search algorithm**, which is used in many fields of computer science due to its completeness, optimality, and **optimal efficiency.**

One major practical drawback is its space complexity, as it stores all generated nodes in memory.

Thus, in practical travel-routing systems, it is generally outperformed by algorithms which can pre-process the graph to attain better performance, as well as memory-bounded approaches; however, A* is still the best solution in many cases

In A* search algorithm, we use search heuristic as well as the cost to reach the node. Hence, we can combine both costs as following, and this sum is called as a **fitness number**.

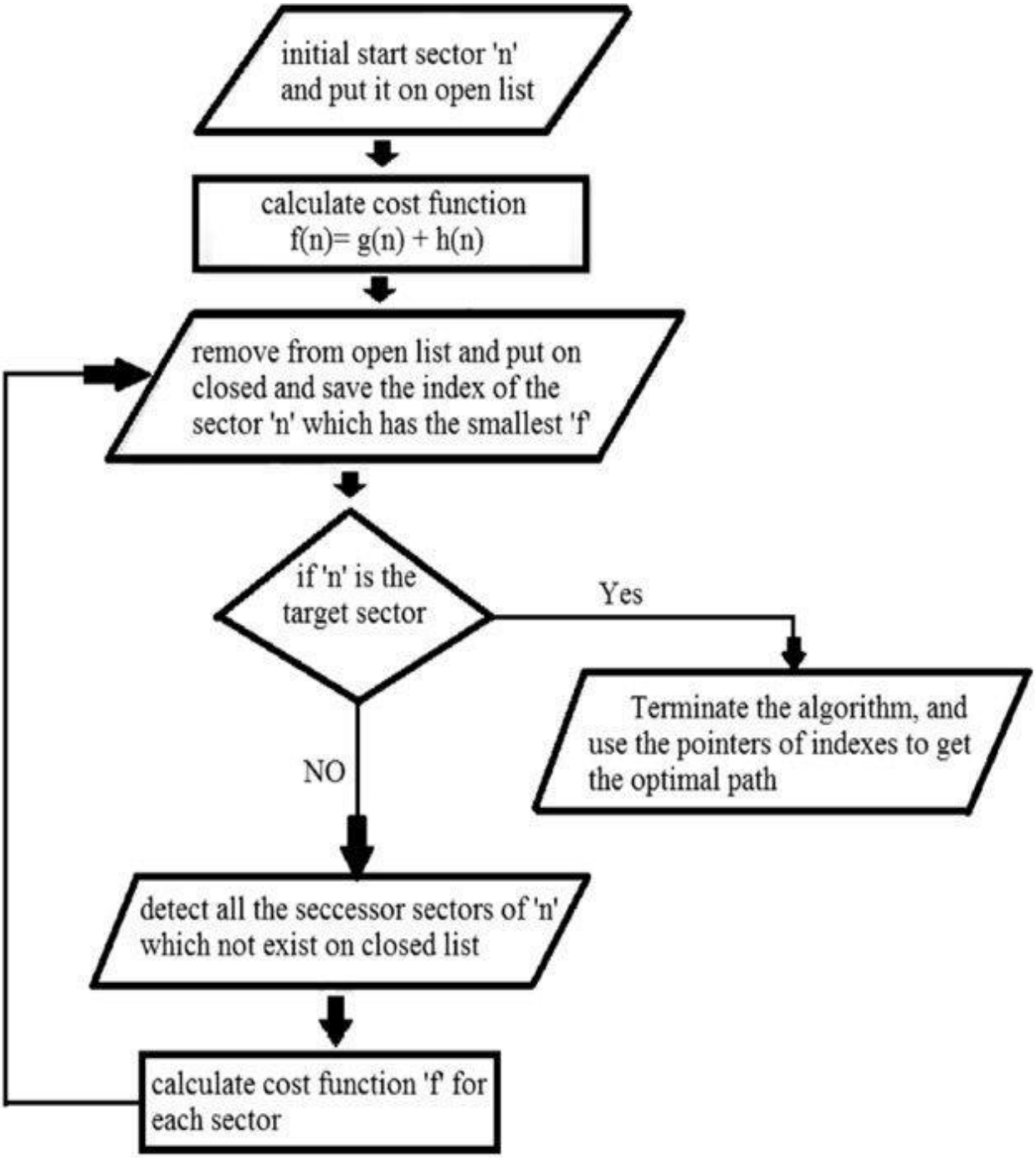
A* search is the most known form of **best-first search**. It uses **heuristic function h(n)**, and cost to reach the node n from the start state g(n).

It has combined features of UCS and greedy best-first search, by which it solves the problem efficiently. **A* search algorithm finds the shortest path** through the search space using the heuristic function.

This search algorithm expands less search tree and provides optimal result faster. A* algorithm is like UCS except that it uses g(n)+h(n) instead of g(n).

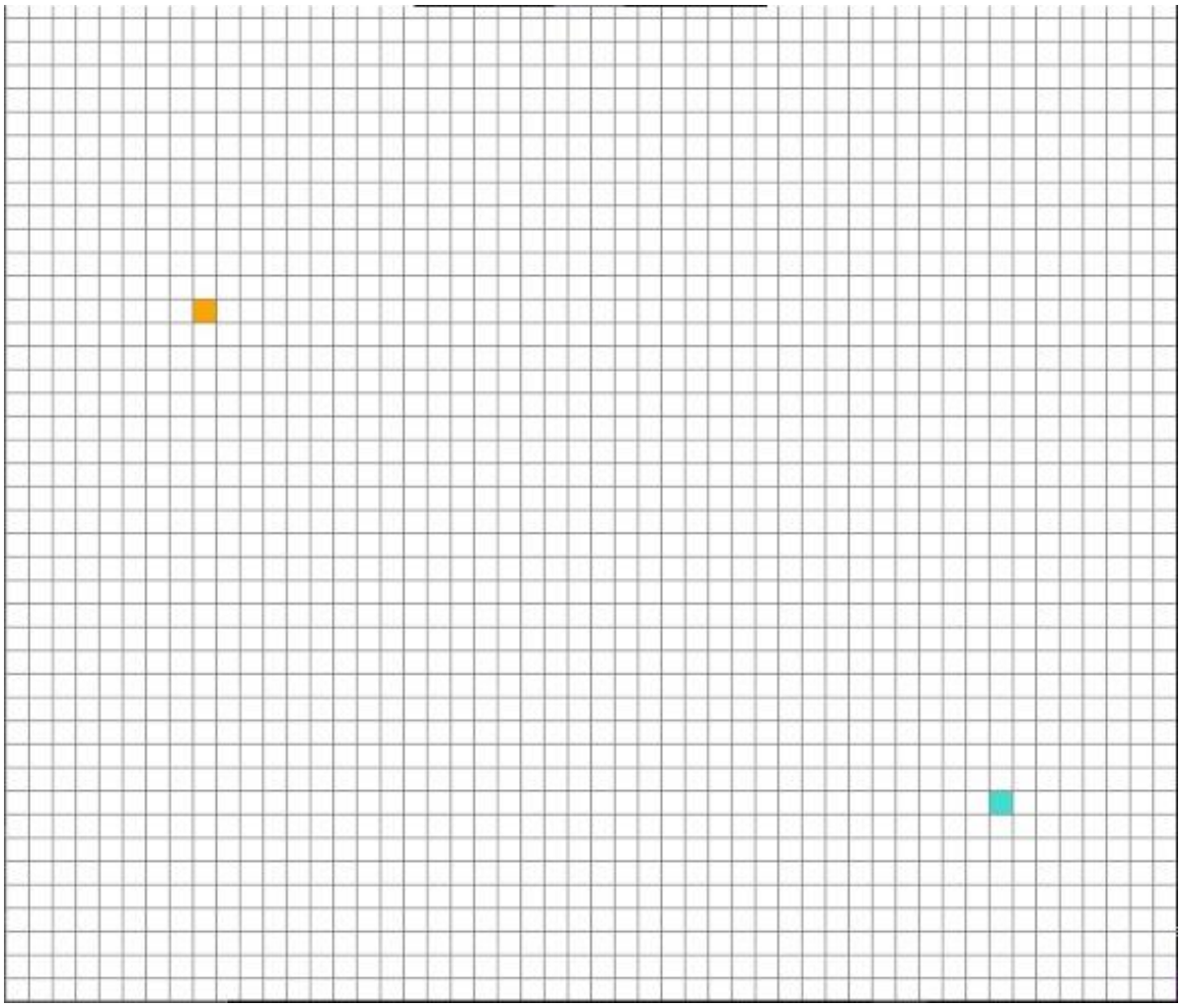
Formula: f(n) = g(n) + h(n)

Flowchart


initial start sector 'n'
and put it on open list
calculate cost function
f(n)= g(n) + h(n)
remove from open list and put on
closed and save the index of the
sector 'n' which has the smallest 'f'
if 'n' is the
target sector
Yes
Terminate the algorithm, and
use the pointers of indexes to get
the optimal path
NO
detect all the seccessor sectors of 'n'
which not exist on closed list
calculate cost function 'f' for
each sector

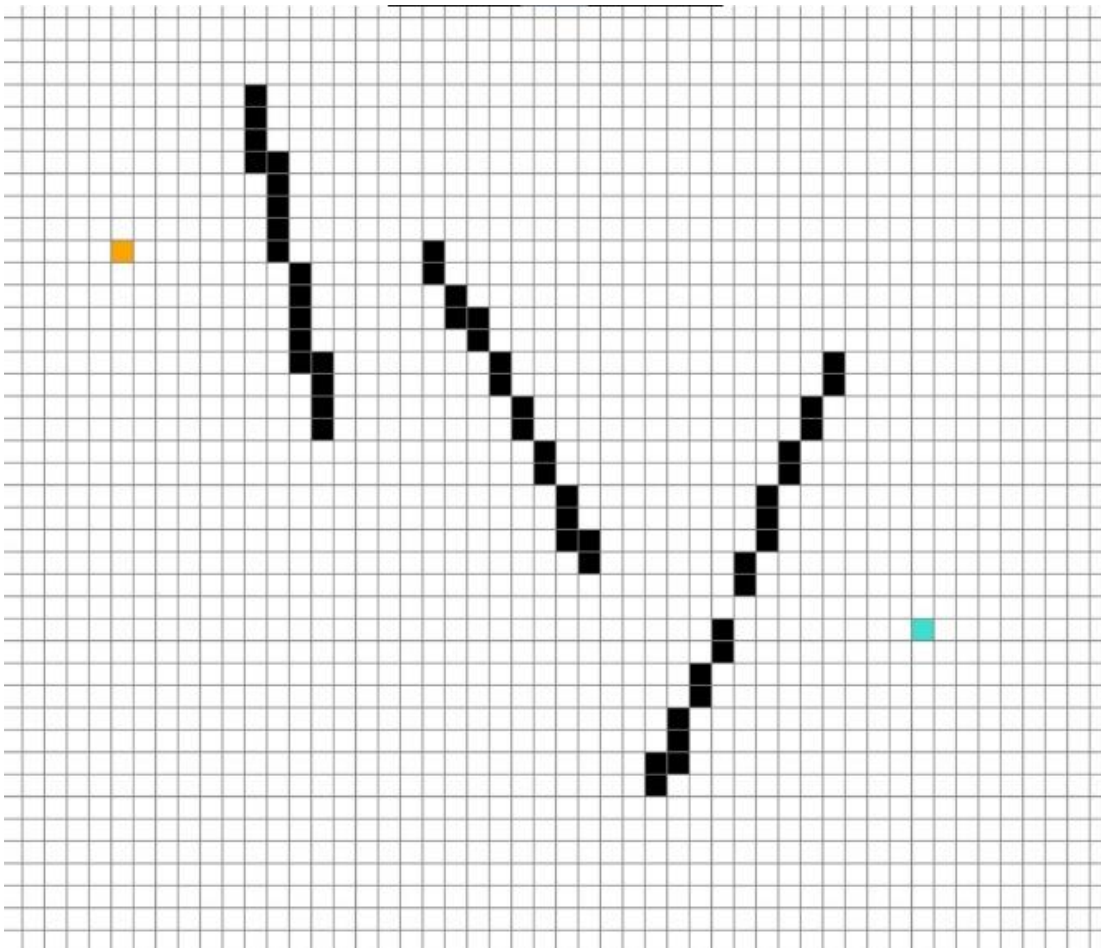
Initial State:

On start-up it is assumed that starting node shall be placed first then the final node and then finally walls will be constructed to build a maze. By right clicking any box the grid we can make it the starting node. If we click again the final node is formed which is signified by a teal-coloured box. Walls can then be placed by dragging and clicking the grid in any place desired. After which the searching algorithm can be run by pressing spacebar which will start the visualisation process.



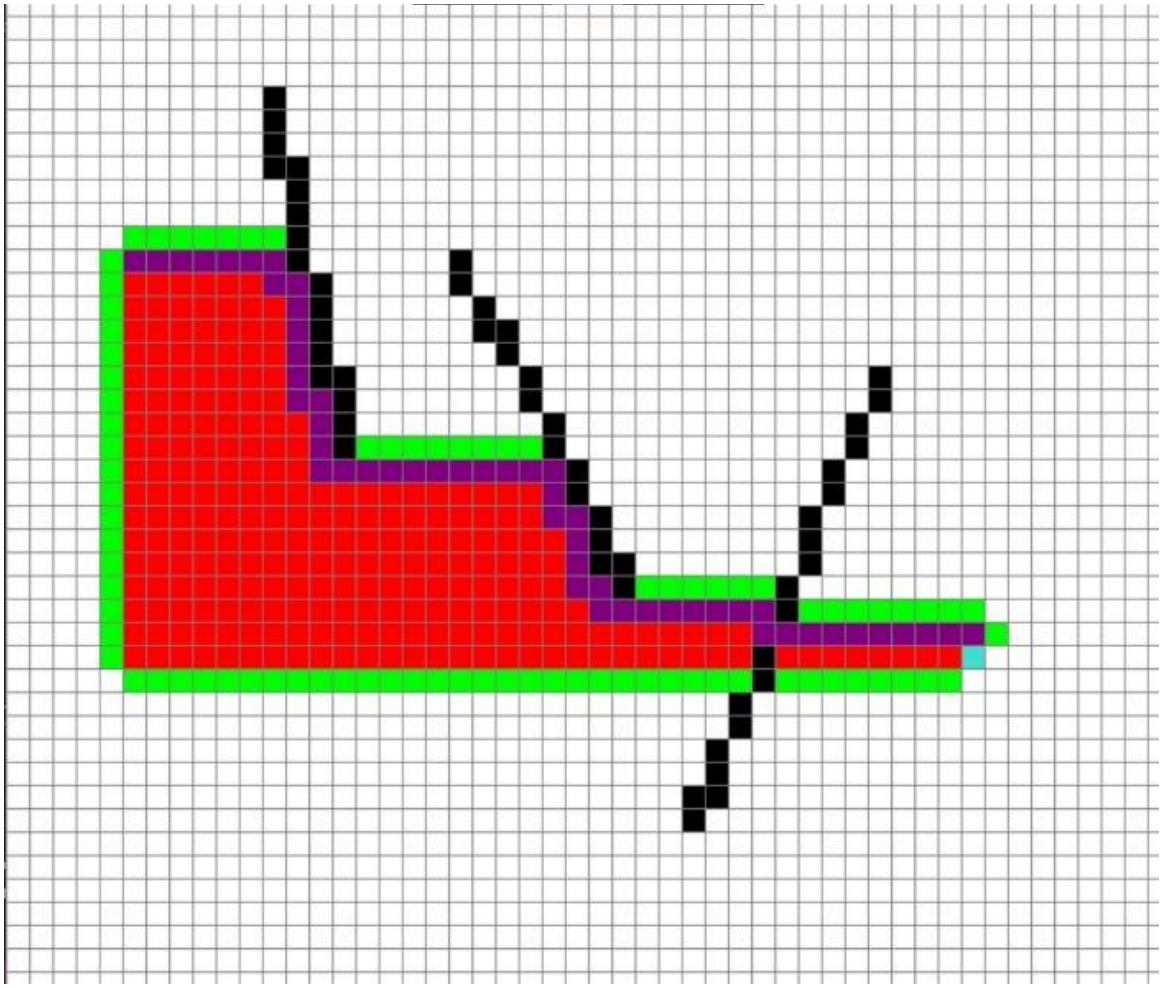
Transition State:

The application starts to visualise A* search by highlighting the grid as it starts from the initial nodes expanding in all directions with the outermost grids highlighted in red and the traversed paths highlighted in green.



Final State:

When the algorithm eventually finds the goal node then it stops expanding further and then the shortest paths out of all paths traversed up till now is rendered on the grid in purple connecting both the initial and the goal nodes



PEAS

Performance:

Running time of the application depends on the complexity of the walls formed by the user. Time and space complexity of the A* search algorithm is O(b^d) where b is the branching factor which is the number of nodes leaving the node and d is the depth of the graph.

Environment:

Environment here is the grid formed using PyGame.

Actuators:

Actuators here is the user itself who manipulates the grid setting the initial and final nodes as well as the walls.

Sensors:

Algorithm uses heuristic function that can be considered functions as it analyses the best way forward.

s