CZ3005
Artificial Intelligence

Lab Exercise 1: Problem Solving

DSAI

U1821610C

Liew Zhi Li (Sherna)

Contents

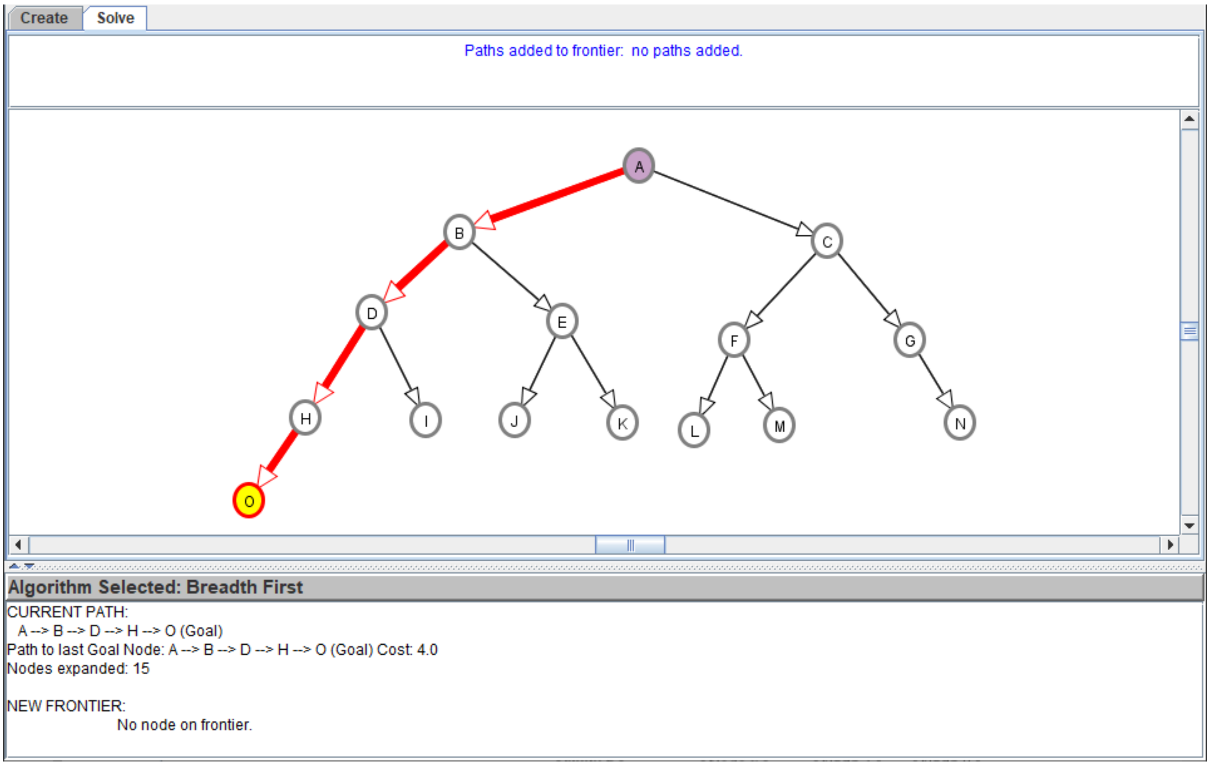
Question 1

For this question, I will be drawing a directed binary tree graph to illustrate the differences in the performance between the three popular search algorithms: Breadth-First Search (BFS), Depth-First Search (DFS) and A* Search. The search algorithm that can expand fewer nodes compared to another algorithm will be deemed as more efficient.

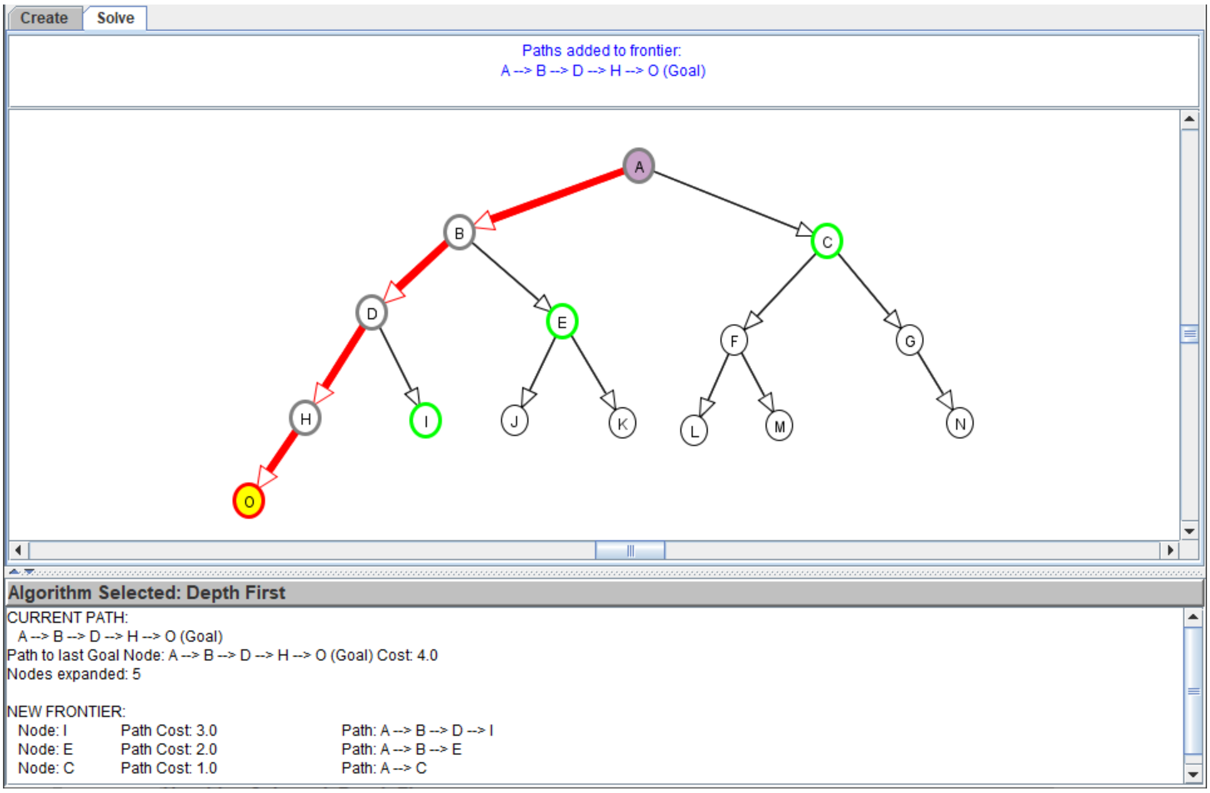
- The start node is A which is highlighted in purple.
- The goal node is highlighted in yellow.
- The order of the nodes is in alphabetical order.
- The search algorithm will expand B first before C.

1a - Give a graph where DFS is much more efficient than BFS

The graph below shows the path taken from starting node A to goal node O using BFS.



The graph below shows the path taken from starting node A to goal node O using DFS.



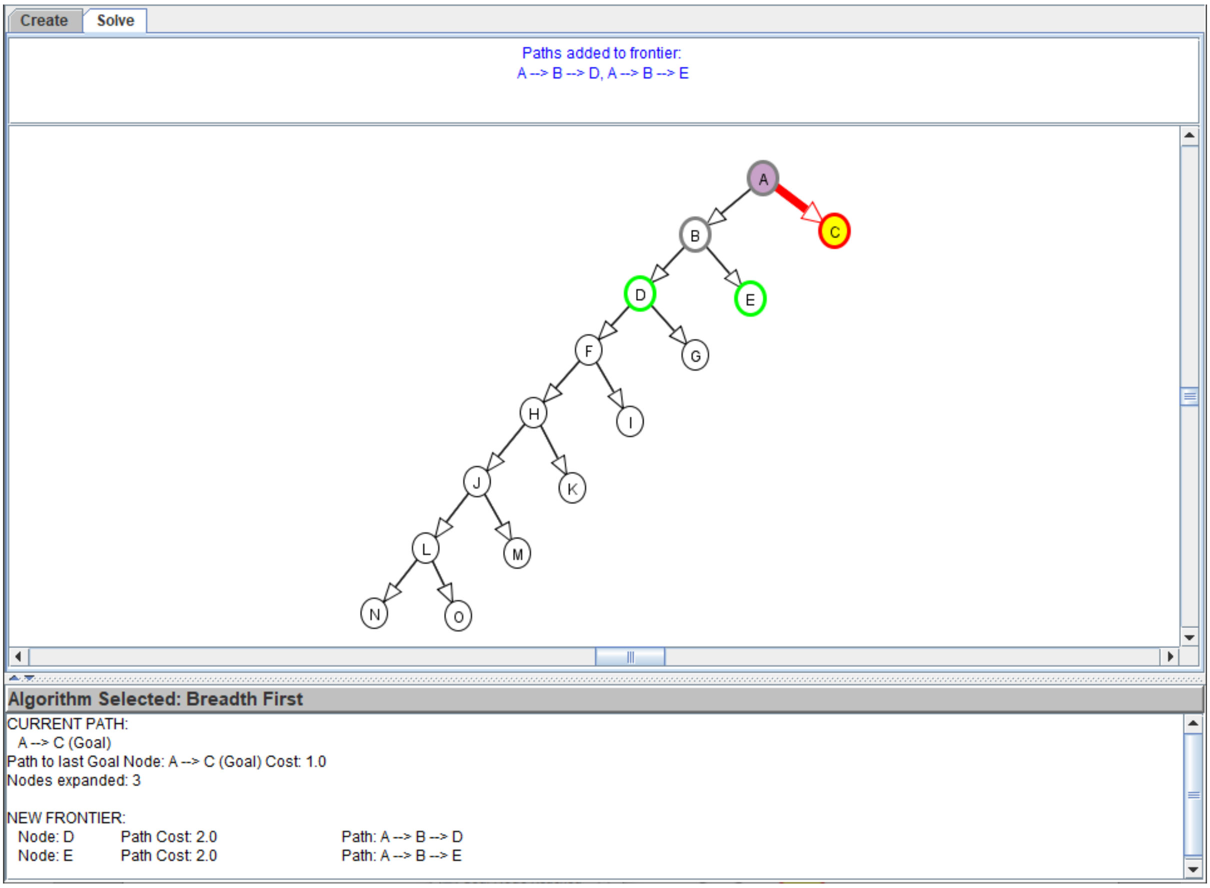
<u>Performance of BFS vs DFS</u>

	BFS	**DFS**
Nodes Expanded	15	5
Path Cost	4	4
Order Expanded	A → B → C → D → E → F → G → H → I → J → K → L → M → N → O	A → B → D → H → O

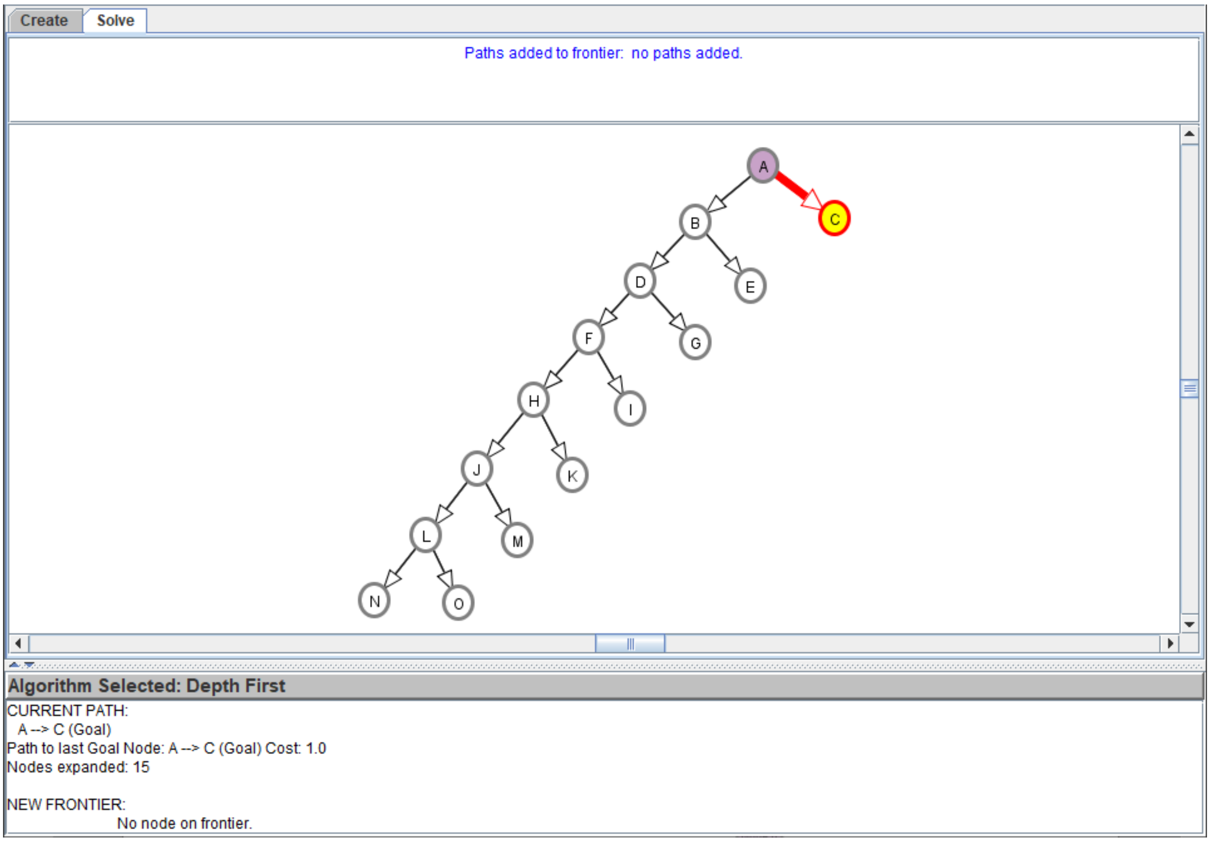
DFS performed more efficiently than BFS in this graph.

1b - Give a graph where BFS is much better than DFS.

The graph below shows the path taken from starting node A to goal node C using BFS.



The graph below shows the path taken from starting node A to goal node C using DFS.



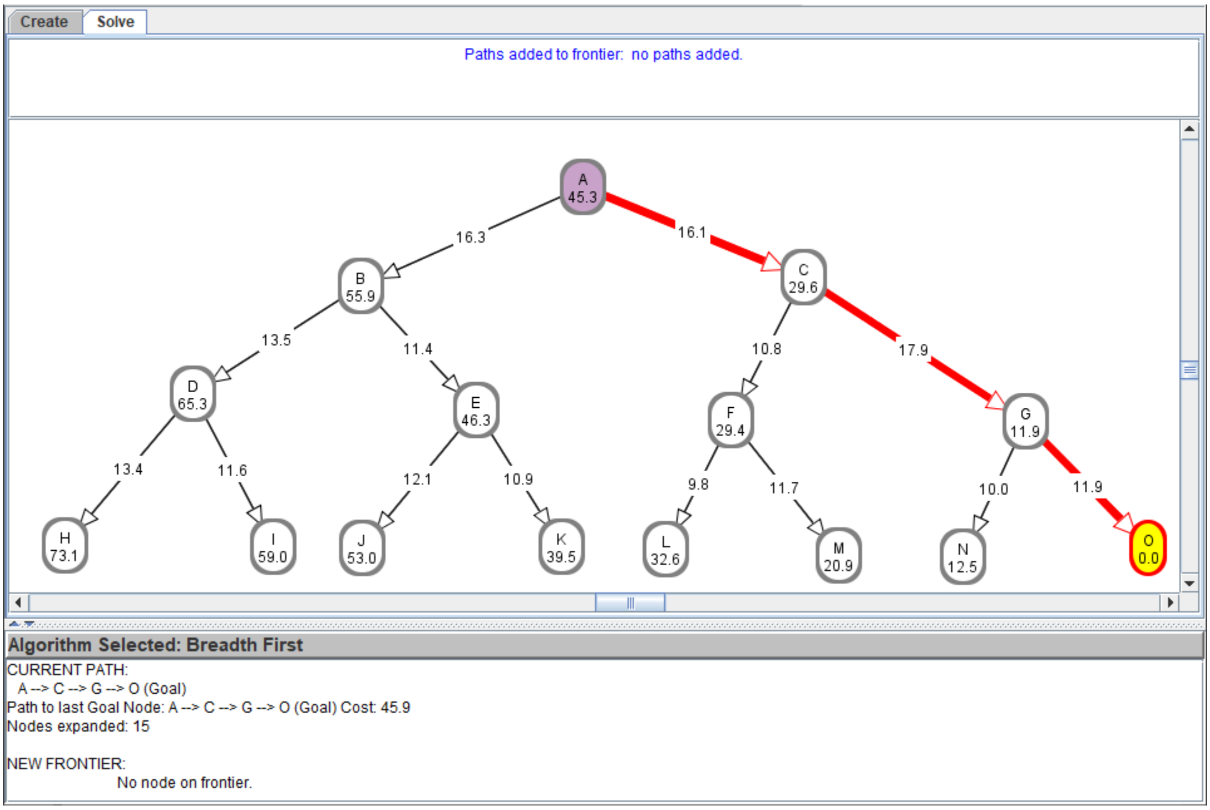
Performance of BFS vs DFS

	BFS	DFS
Nodes Expanded	3	15
Path Cost	1	1
Order Expanded	A → B → C	A → B → D → F → H → J → L →N → O → M → K → I → G → E → C

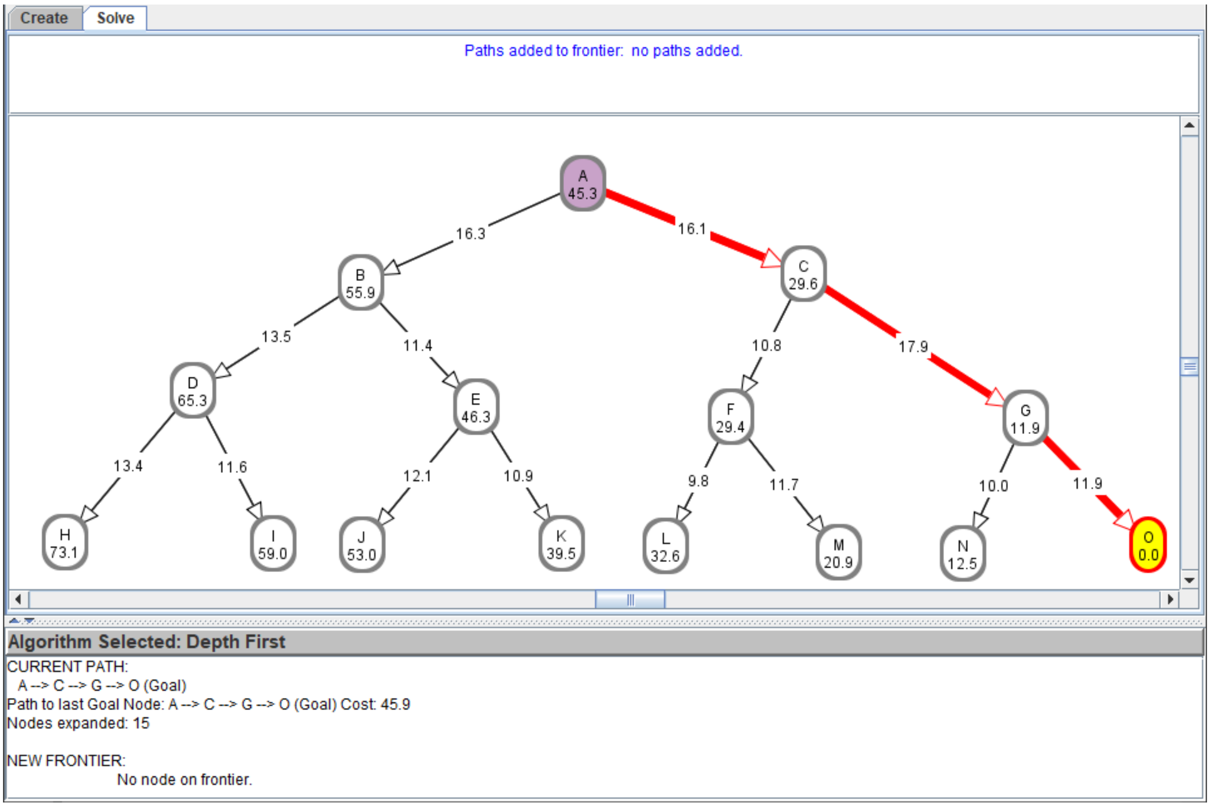
BFS performed more efficiently than DFS in this graph.

1c - Give a graph where A* search is more efficient than either DFS or BFS.

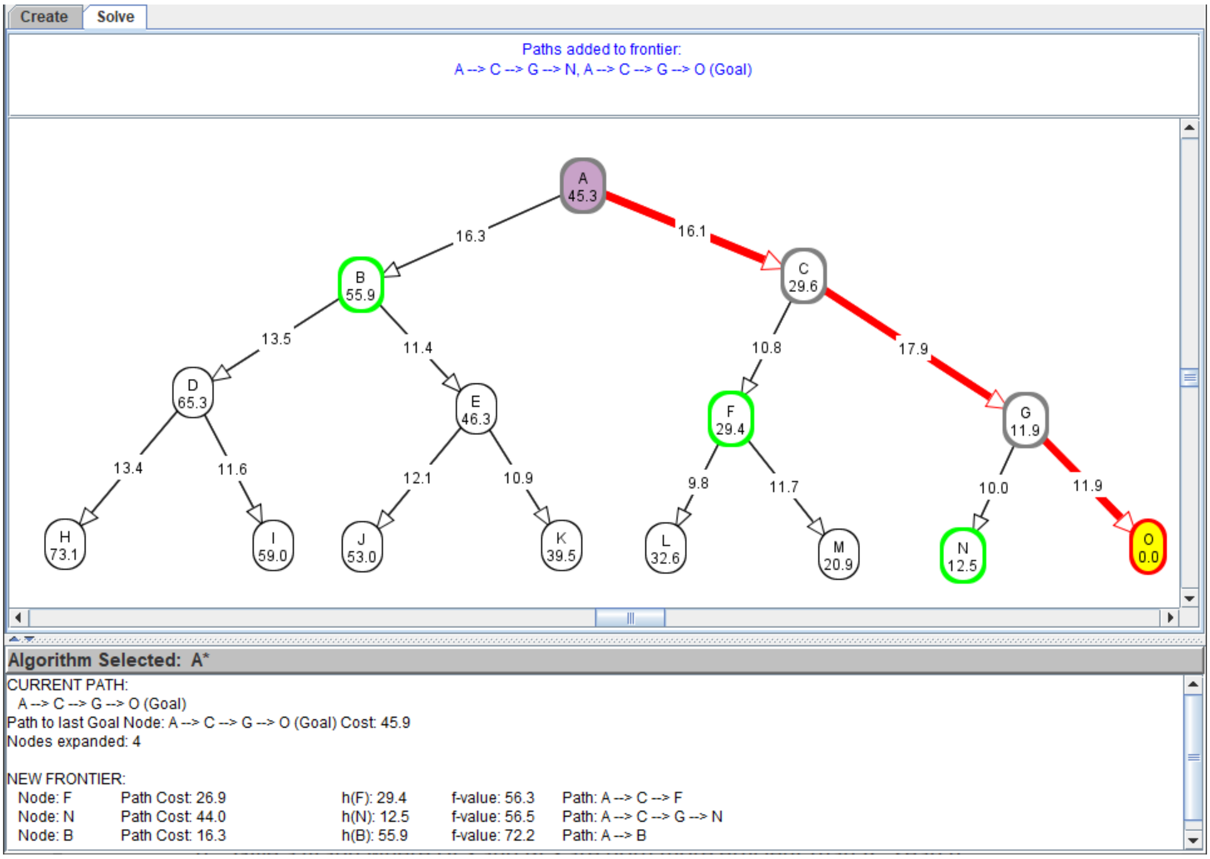
The graph below shows the path taken from starting node A to goal node O using BFS.



The graph below shows the path taken from starting node A to goal node O using DFS.



The graph below shows the path taken from starting node A to goal node O using A*.



Performance of BFS vs DFS vs A*

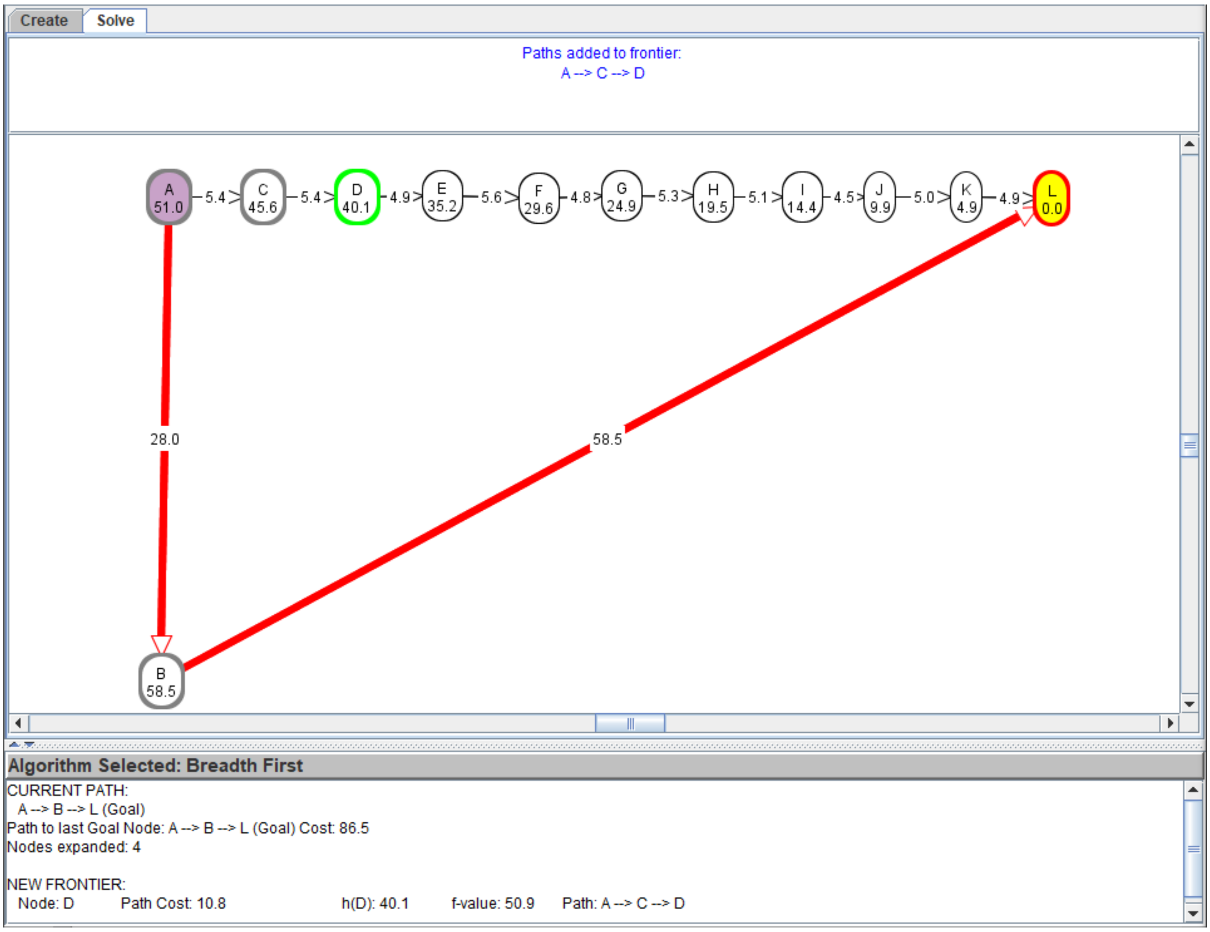
	BFS	DFS	A*
Nodes Expanded	15	15	4
Path Cost	45.9	45.9	45.9
Order Expanded	A → B → C → D → E → F → G → H → I → J → K → L → M → N → O	A → B → D → H → I → E → J → K → C → F → L → M → G → N → O	A → C → G → O

Note: A* uses node heuristics. BFS and DFS ignores heuristics.

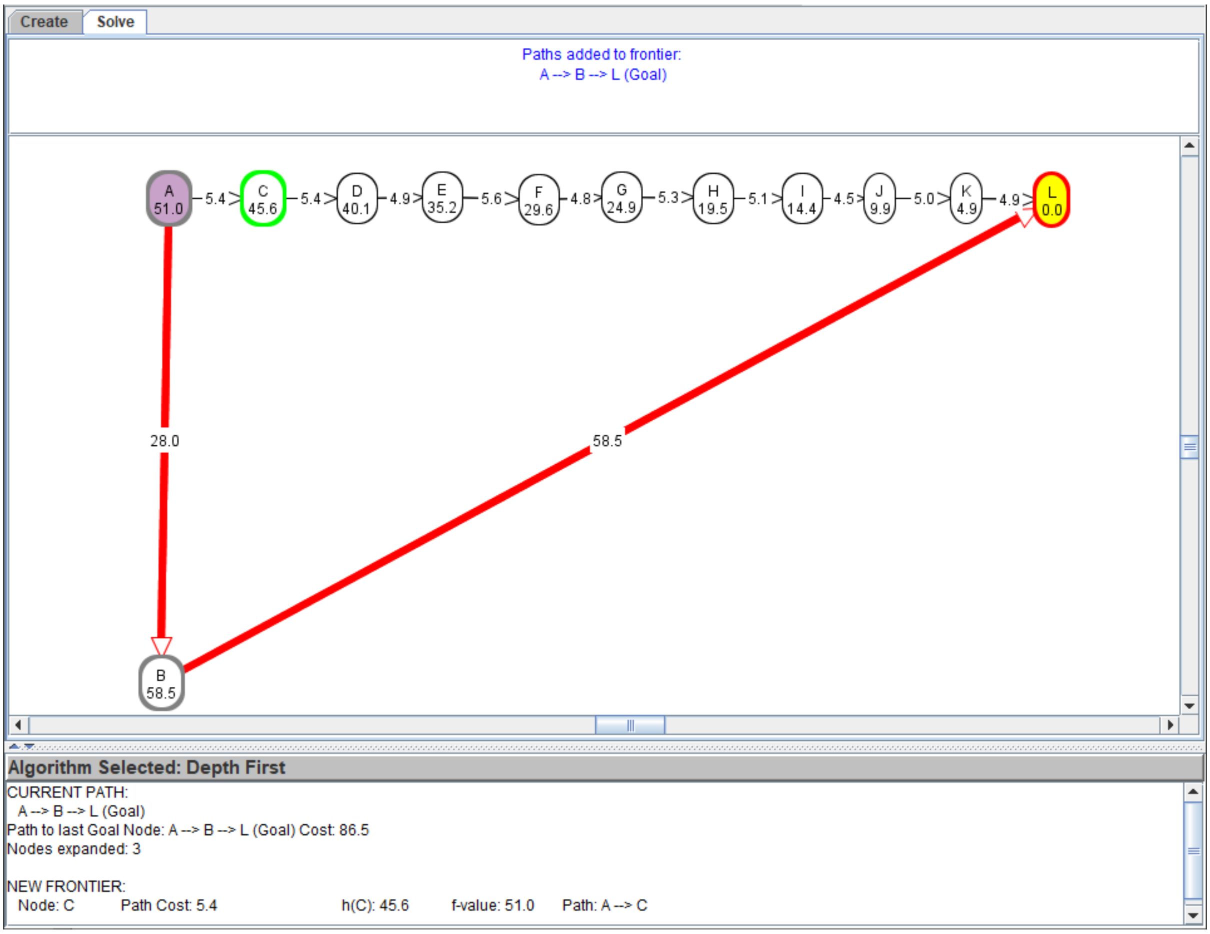
A* performed more efficiently than both BFS and DFS in this graph.

1d - Give a graph where DFS and BFS are both more efficient than A* search.

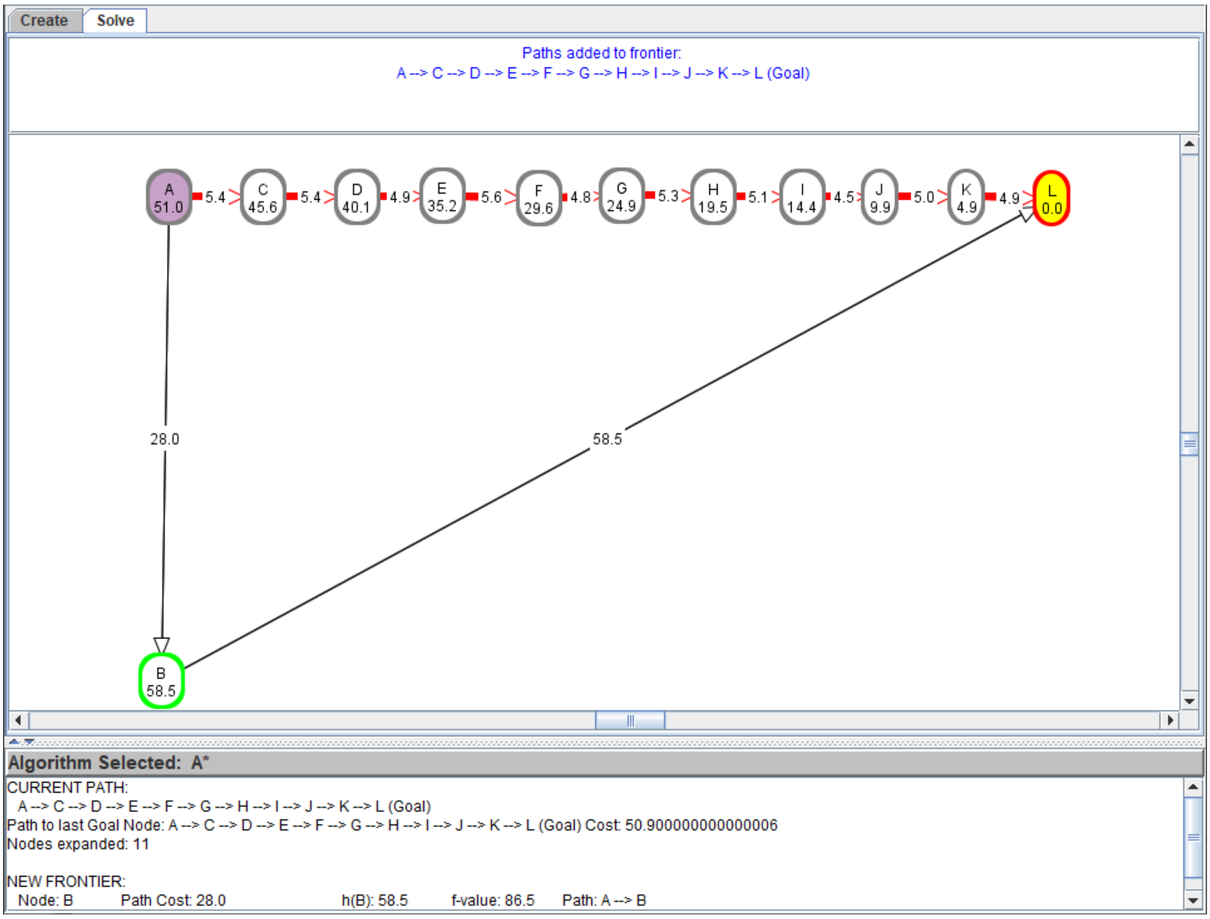
The graph below shows the path taken from starting node A to goal node L using BFS.



The graph below shows the path taken from starting node A to goal node L using DFS.



The graph below shows the path taken from starting node A to goal node L using A*.



Performance of BFS vs DFS vs A*

	BFS	DFS	A*
Nodes Expanded	4	3	11
Path Cost	86.5	86.5	50.9
Order Expanded	A → B → C → L	A → B → L	A → C → D → E → F → G → H → I → J → K → L

Note: A* uses node heuristics, however BFS and DFS ignores heuristics.

Both BFS and DFS performed more efficiently than A* in this graph.

Question 2

2a - What is the effect of reducing h(n) when h(n) is already an underestimate?

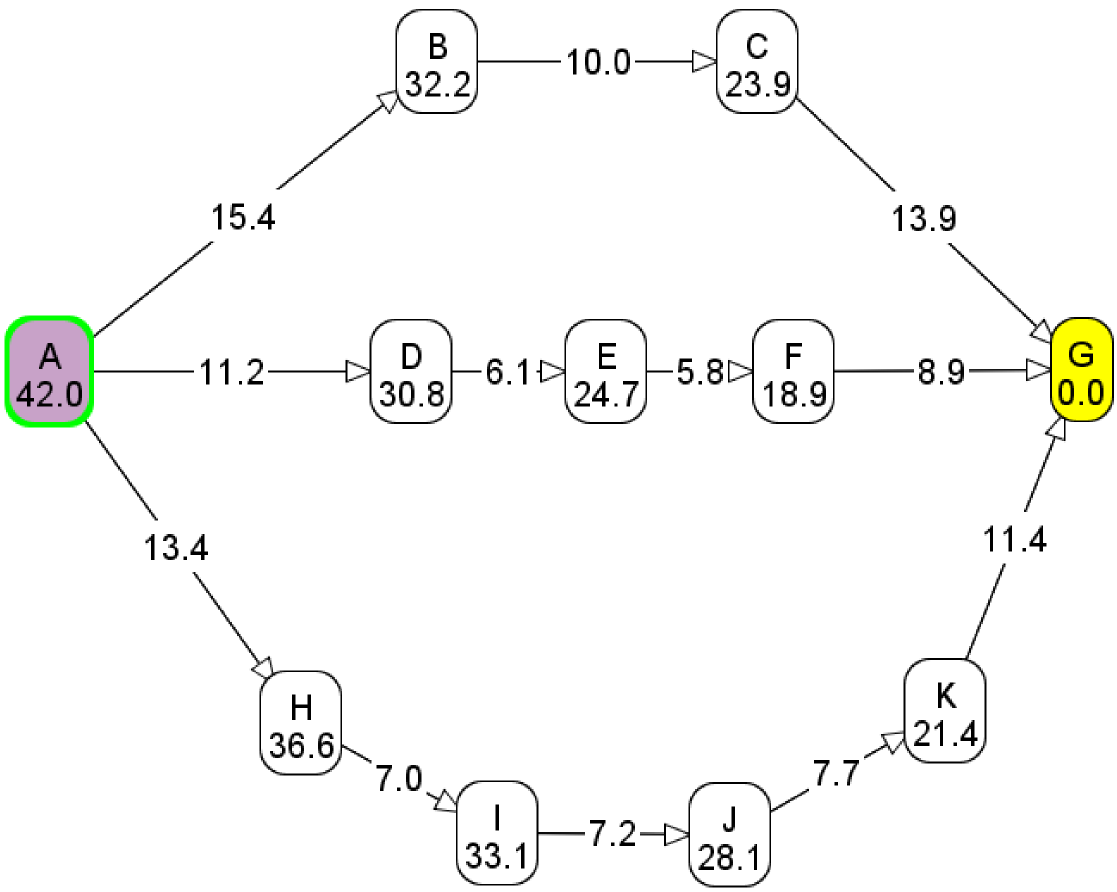
Conjecture

When h(n) is already an underestimate and we reduce h(n), it does not affect the admissibility of A*. A* remains guaranteed to find a solution and the first solution found is optimal. However, the efficiency of A* may be reduced as it expands more nodes during its search.

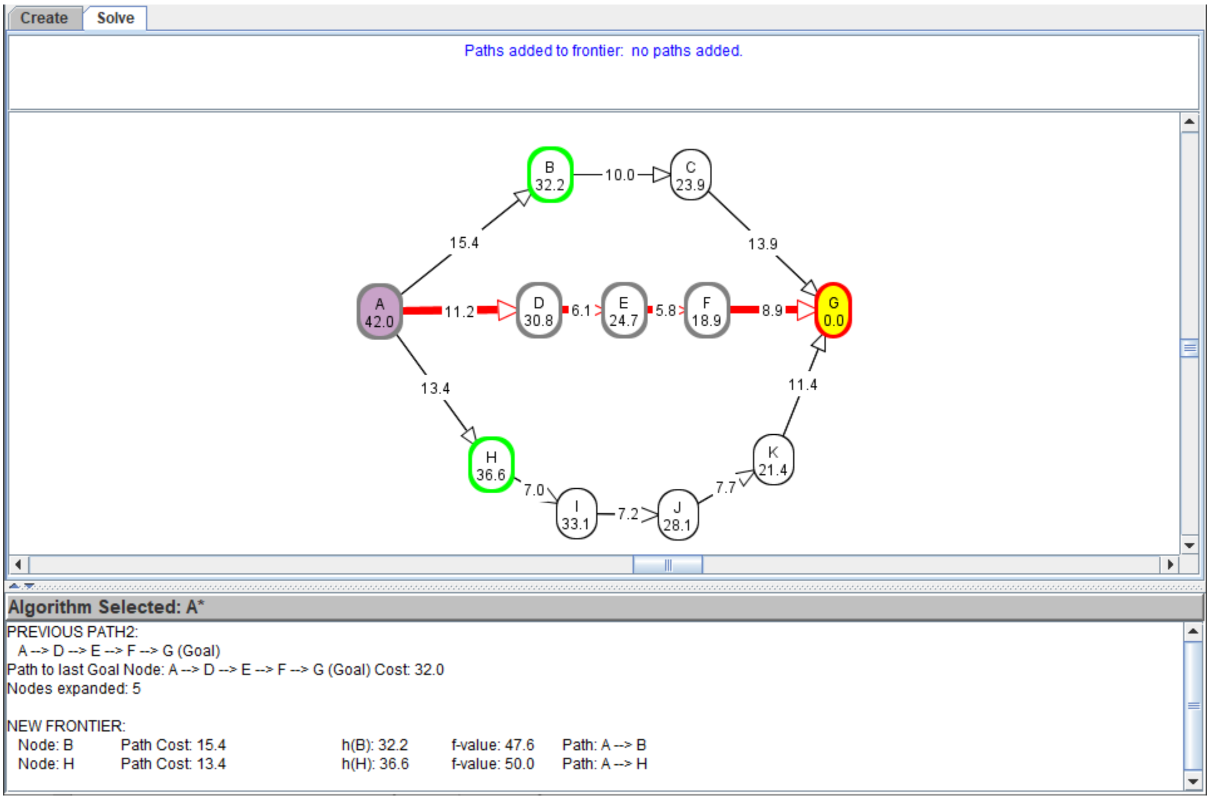
Empirical Evidence

The following graph shown below has node A as the start node and node G as the goal node with original h(n) values. There are three paths from A to reach the goal node G:

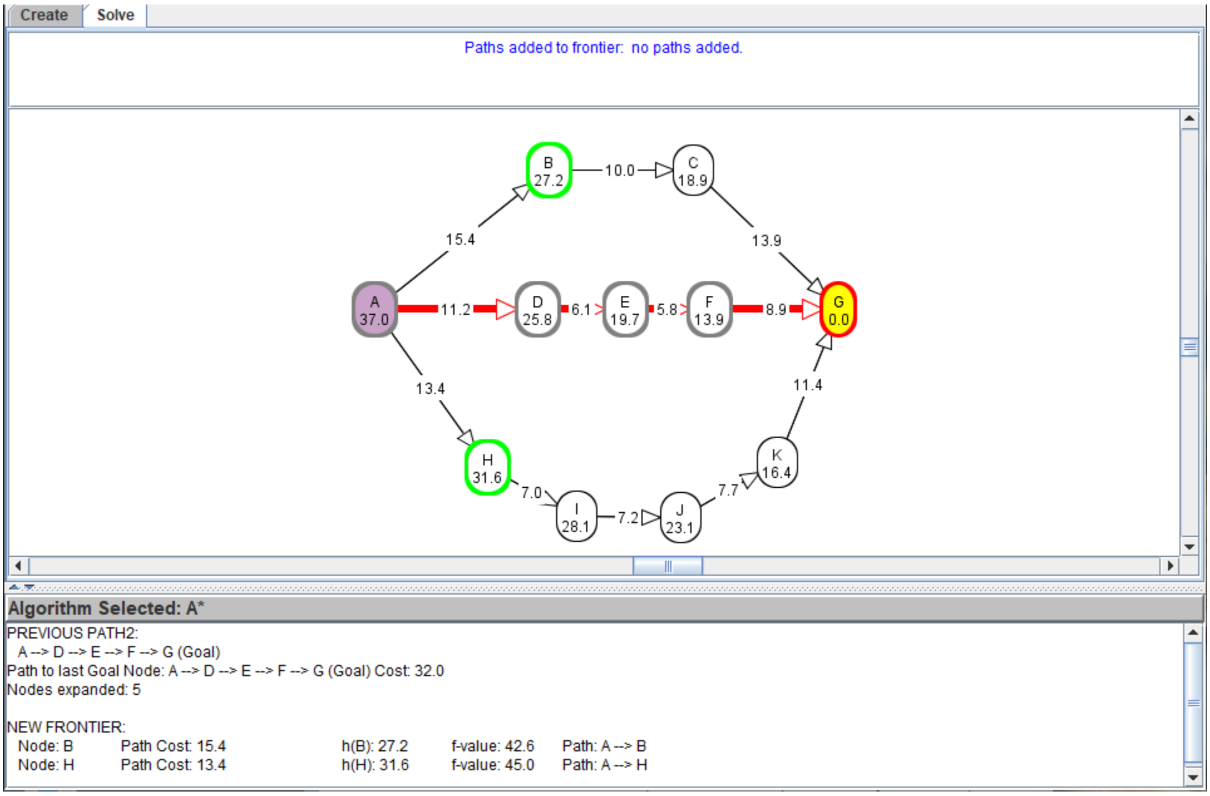
- Path A → B → C → G
  - Cost: 39.3
- Path A → D → E → F → G (Optimal path)
  - Cost: 32
- Path A → H → I → J → K → G
  - Cost: 46.7



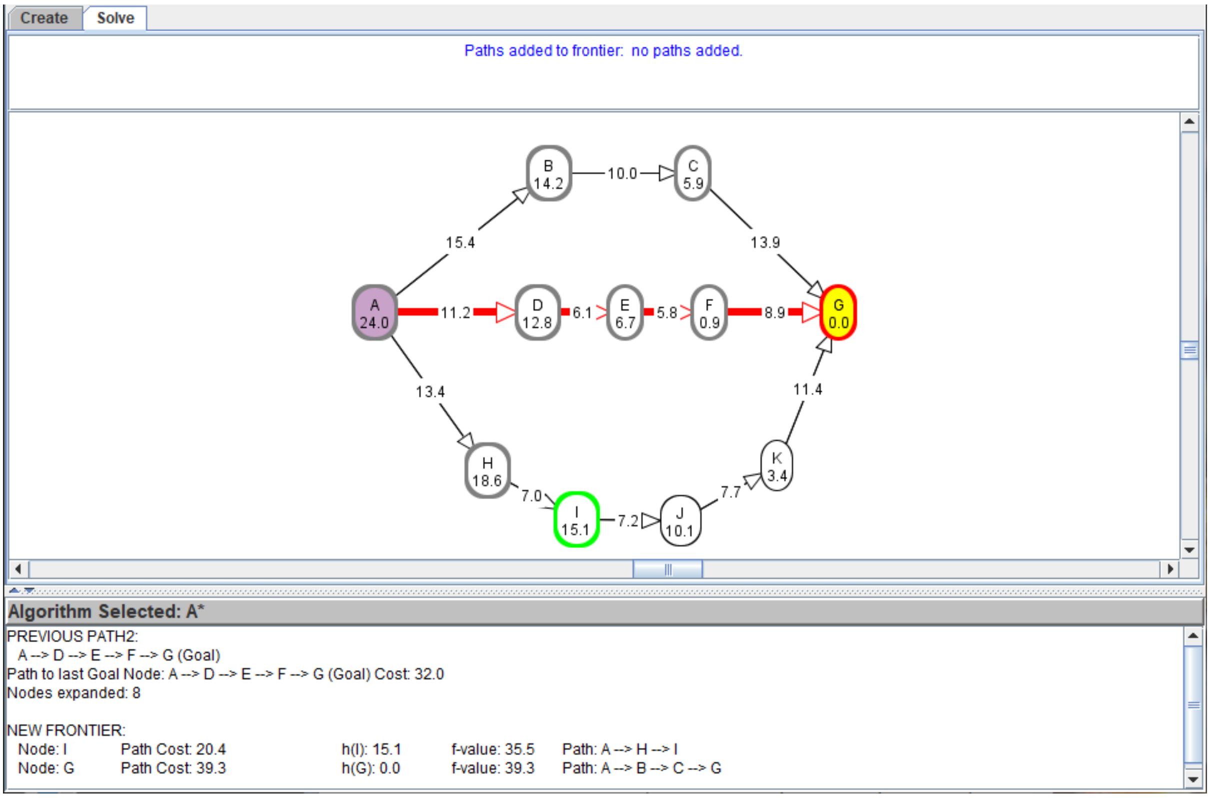
Experiment 1: By running A* on this graph, it returns the optimal path A → D → E → F → G, with the number of nodes expanded = 5.



Experiment 2: When we reduce the h(n) of all nodes (except the goal node) by a small constant 5, A* still returns the same result as the preceding experiment.



Experiment 3: When we reduce the h(n) of all nodes (except the goal node) by a large constant 18, A* returns the optimal path A → D → E → F → G. However, the number of nodes expanded has increased to 8. Thereby reducing the efficiency of A*.



Conclusion

The A* search algorithm uses an admissible heuristic to estimate the cost of reaching the goal state from the current node n. The evaluation function for A* is: $f(n) = g(n) + h(n)$ where $f(n)$ = the evaluation function, $g(n)$ = the cost from start node to current node n, $h(n)$ = the estimate cost from current node to goal node.

For a heuristic to be admissible, the estimated cost must always be lower than or equal to the actual cost of reaching the goal state.

When $h(n)$ is an underestimate, it is an admissible heuristic. Thus, when we reduce $h(n)$ further, it remains an admissible heuristic. With an admissible heuristic, A* is guaranteed to find the optimal solution path.

When $h(n)$ is reduced, it may become less efficient and expand more nodes. However, it is not always the case as it is when the $f(n)$ is reduced such that $f(n_1) \geq f(n_2)$ for some nodes n_1 and n_2, where $n_1$is along the optimal path and n_2 is along a different, non-optimal path. Thus, the non-optimal path along n_2 is expanded.

If $h(n)$ is the exact distance, then $f(n_1) \leq f(n_2)$ for all nodes n_1 and n_2. Thus, nodes along the non-optimal path are not expanded.

In Experiment 1, using the original $h(n)$ values, A* returned the solution where the nodes expanded along the optimal path A → D → E → F → G with number of nodes expanded = 5.

In Experiment 2, using $h(n)$ values that are reduced by a small constant 5, A* returned the same results as Experiment 1. This is because $f(n)$ values of node B = 42.6 and node H = 45, which are higher than the $f(n)$ value of G, which is 32.

Algorithm Selected: A*

PREVIOUS PATH2:
A --> D --> E --> F
NEW FRONTIER:

Node: G	Path Cost: 32.0	h(G): 0.0	f-value: 32.0	Path: A --> D --> E --> F --> G
Node: B	Path Cost: 15.4	h(B): 27.2	f-value: 42.6	Path: A --> B
Node: H	Path Cost: 13.4	h(H): 31.6	f-value: 45.0	Path: A --> H

In Experiment 3, using $h(n)$ values that are reduced by a large constant 18, A* returned the optimal path A → D → E → F → G. However, the number of nodes expanded = 8. In this case, it expanded an additional 3 nodes: B, C, H. This happened because the $f(n)$ values of node B = 29.6 and H = 32 and C = 31.3, which are lower than the $f(n)$ value of G, which is 32.

Algorithm Selected: A*

PREVIOUS PATH2:
A --> D --> E --> F
NEW FRONTIER:

Node: B	Path Cost: 15.4	h(B): 14.2	f-value: 29.6	Path: A --> B
Node: H	Path Cost: 13.4	h(H): 18.6	f-value: 32.0	Path: A --> H
Node: G	Path Cost: 32.0	h(G): 0.0	f-value: 32.0	Path: A --> D --> E --> F --> G

Algorithm Selected: A*

PREVIOUS PATH2:
A --> B
NEW FRONTIER:

Node: C	Path Cost: 25.4	h(C): 5.9	f-value: 31.3	Path: A --> B --> C
Node: H	Path Cost: 13.4	h(H): 18.6	f-value: 32.0	Path: A --> H
Node: G	Path Cost: 32.0	h(G): 0.0	f-value: 32.0	Path: A --> D --> E --> F --> G

In conclusion, the above experiments have shown that when $h(n)$ is already an underestimate, a further reduction does not affect the admissibility. A* still guarantees to find the optimal solution, thus its search accuracy remains unchanged. However, the efficiency of A* may sometimes be reduced depending on how much $h(n)$ is reduced.

2b - How does A* perform when h(n) is the exact distance from n to a goal?

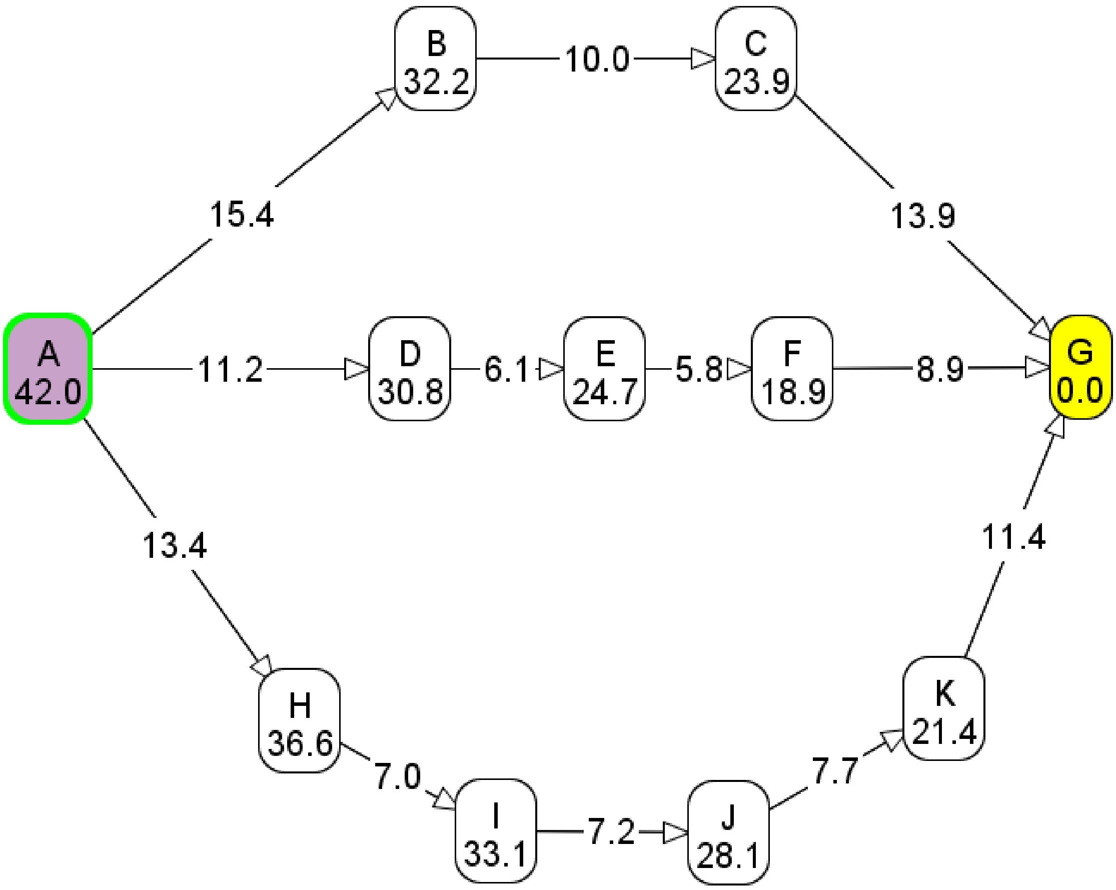
Conjecture

When h(n) is the exact distance from n to a goal, A* is guaranteed to expand the nodes along the optimal path in all cases.

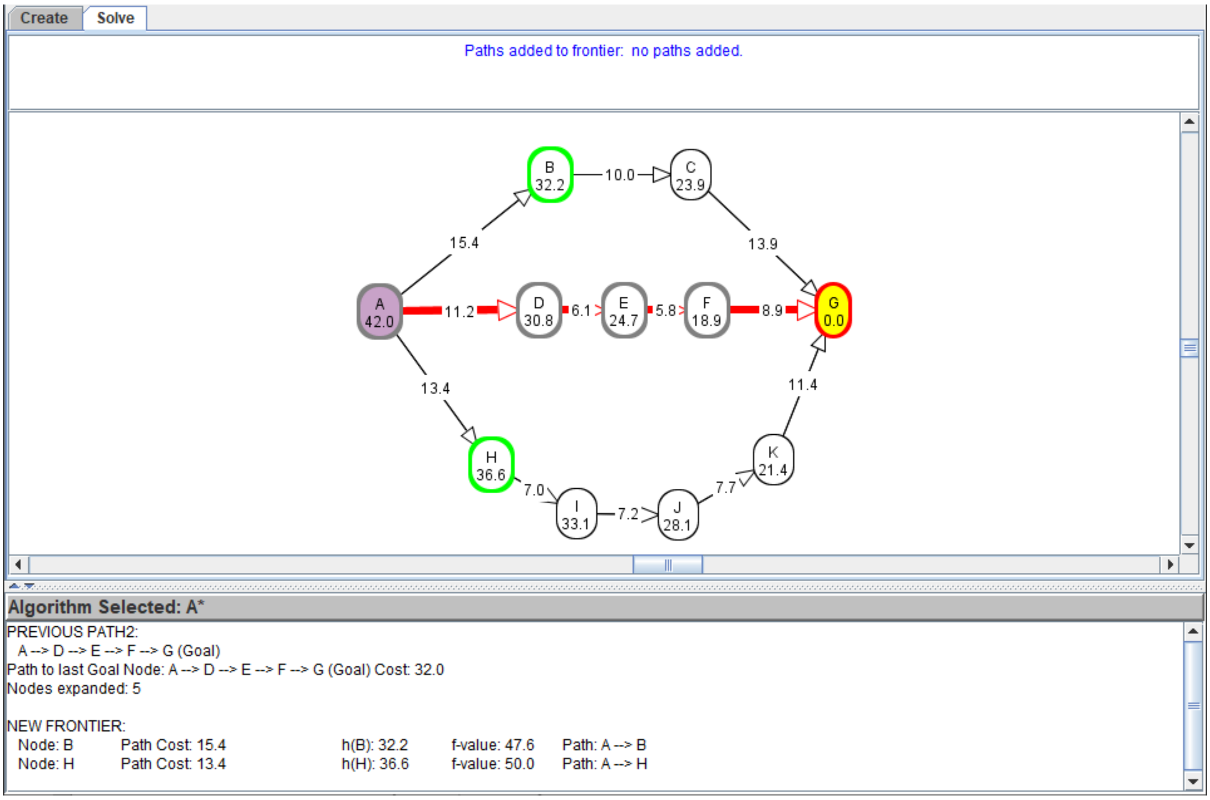
Empirical Evidence

The following graph shown below has node A as the start node and node G as the goal node with original h(n) values. There are three paths from A to reach the goal node G:

- Path A → B → C → G
  - Cost: 39.3
- Path A → D → E → F → G (Optimal path)
  - Cost: 32
- Path A → H → I → J → K → G
  - Cost: 46.7

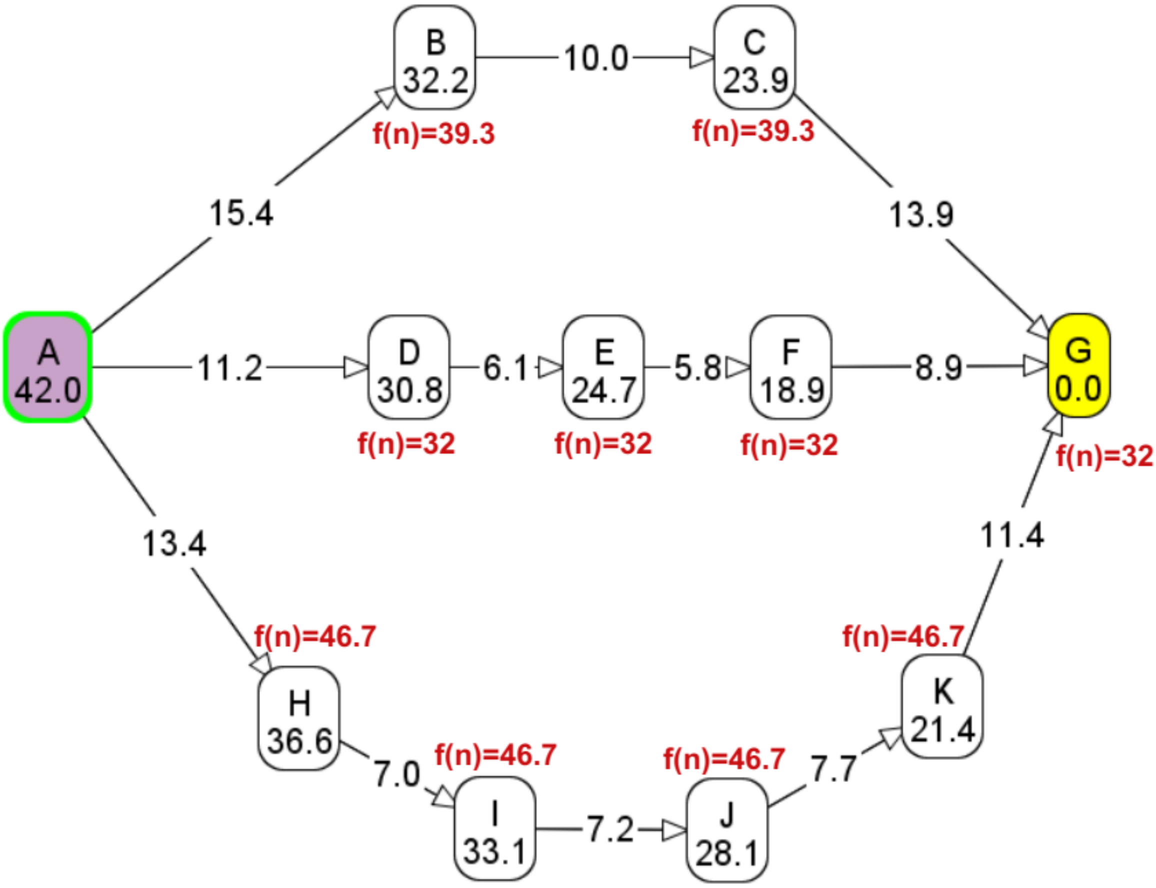


Experiment 1: By running A* on this graph, it returns the optimal path A → D → E → F → G, with the number of nodes expanded = 5.



Conclusion

When h(n) is the exact distance from n to a goal, h(n) is the actual cost to reach the goal. In other words, the $f(n)$ values of nodes along a path is the actual path cost for the solution.



The A* search algorithm will expand the nodes along the path with the lowest $f(n)$ value. The efficiency of A* remains unaffected. Thus, A* is guaranteed to expand nodes along the optimal path with the best efficiency in all cases.

2c - What happens if h(n) is not an underestimate?

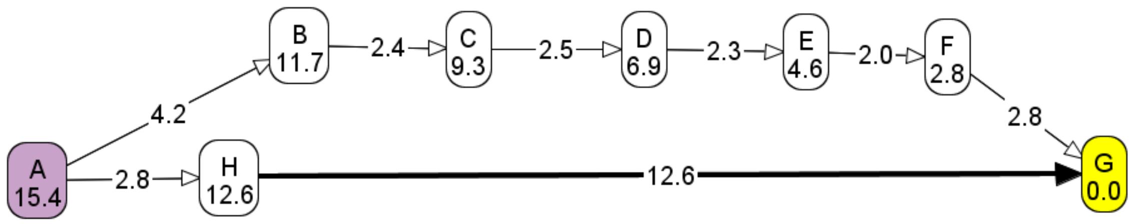
<u>Conjecture</u>

When h(n) is not an underestimate, the admissibility of A* is not guaranteed. In other words, A* is not guaranteed to find the optimal solution.

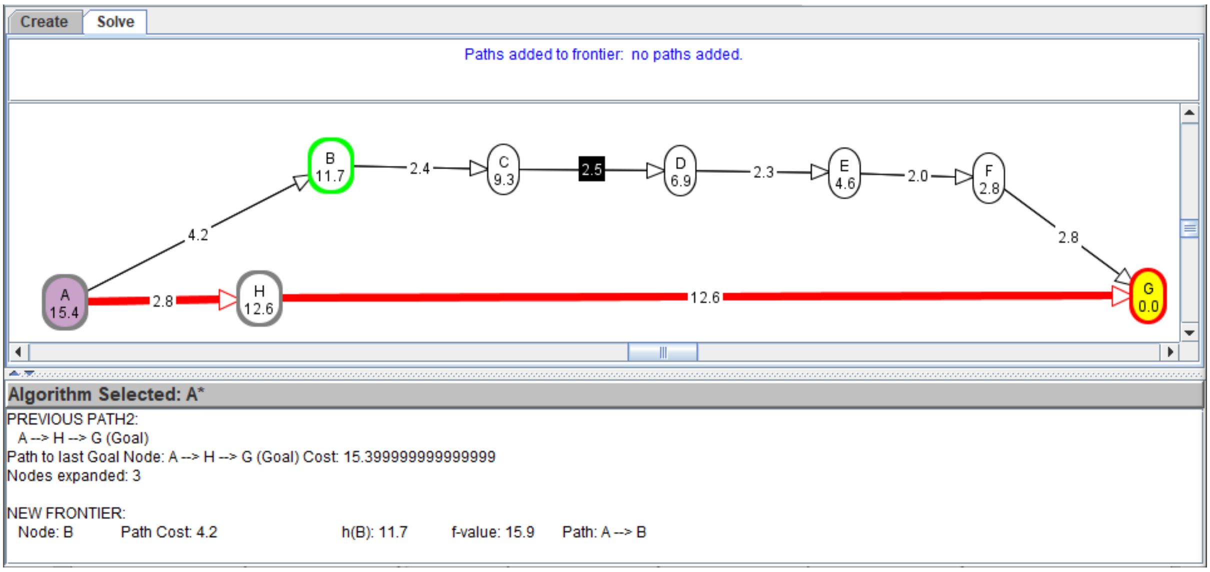
<u>Empirical Evidence</u>

The following graph shown below has node A as the start node and node G as the goal node with original h(n) values. There are two paths from A to reach the goal node G:

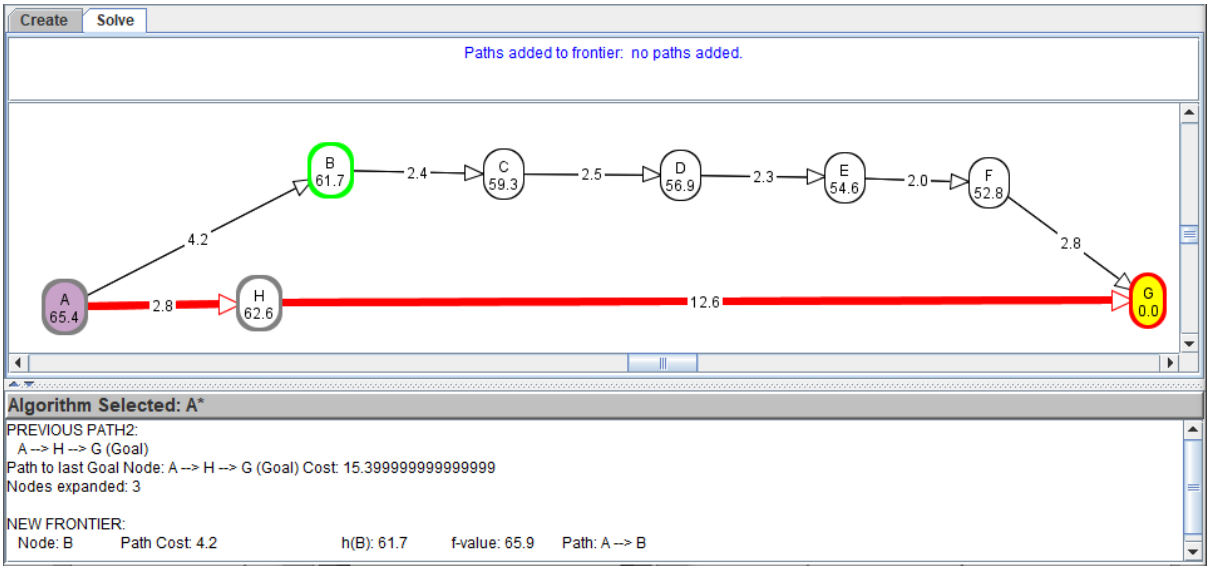
- Path A → H → G (Optimal path)
    - Cost: 15.4
- Path A → B → C → D → E → F → G
    - Cost: 16.2



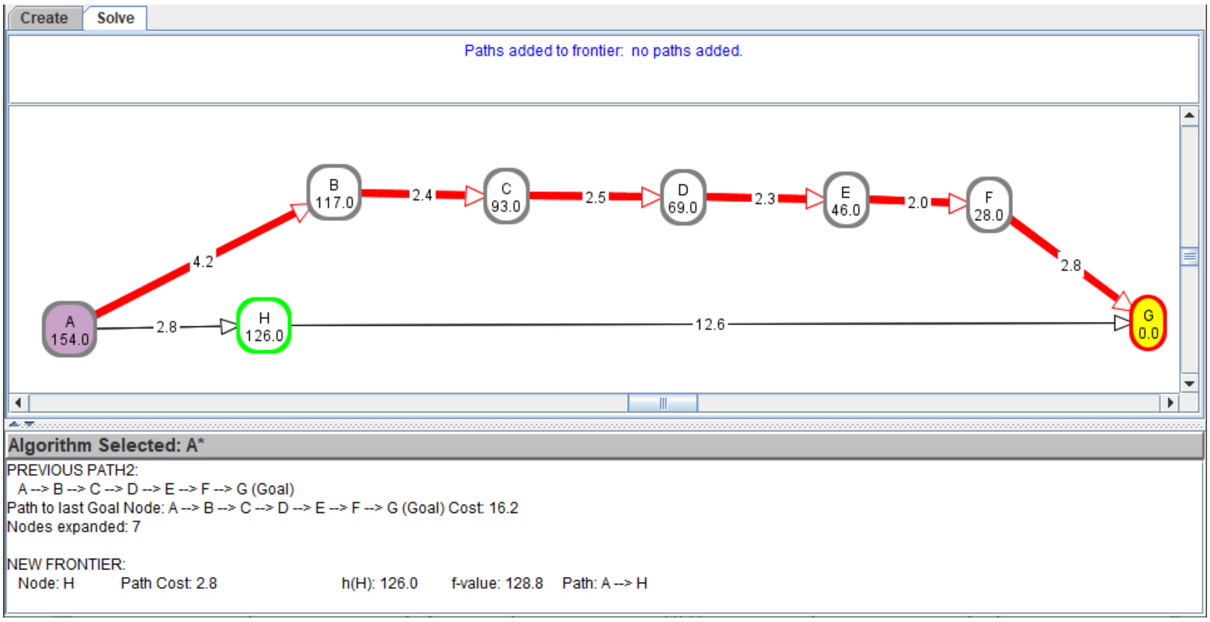
Experiment 1: By running A* on this graph, it returns the optimal path A → H → G, with the number of nodes expanded = 3.



Experiment 2: When we increment the h(n) of all nodes (except the goal node) by 50, A* still returns the same result as the preceding experiment.



Experiment 3: When we multiply the h(n) of all nodes (except the goal node) by a factor of 10, A* returns the non-optimal path A → B → C → D → E → F → G as the solution, with the number of nodes expanded = 7.

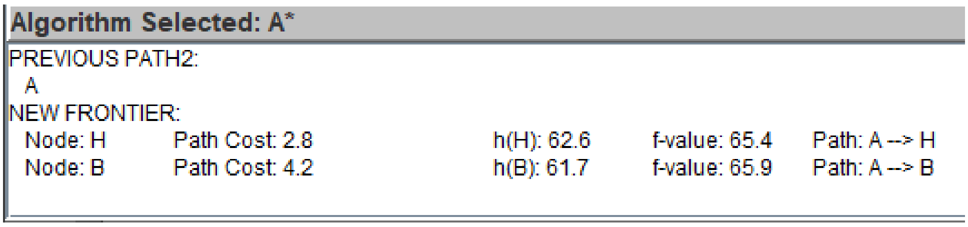


Conclusion

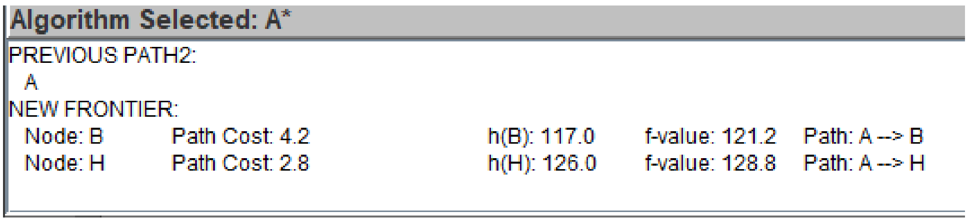
When $h(n)$ is not an underestimate, then $h(n)$is an overestimate. Thus, $h(n)$ becomes a non-admissible heuristic, and A* could potentially overlook the optimal solution due to the overestimation in $f(n)$.

In Experiment 1, using the original $h(n)$ values, running A* returned the optimal path with the number of nodes expanded = 3.

In Experiment 2, using $h(n)$ values that are incremented by a constant 50, A* returned the same results as Experiment 1. This is because $f(n)$ value of node B = 65.9, which is higher than the $f(n)$ value of H, which is 65.4. Thus, A* chose to remove node H from the frontier to expand the goal node G along the optimal path.



In Experiment 3, using $h(n)$ values that are multiplied by a large factor of 10, caused h(n) to overestimate and become a non-admissible heuristic. As a result, A* returned the non-optimal path A → B → C → D → E → F → G as the solution. This happened because $f(n)$ value of node B = 121.2, which is lower than the $f(n)$ value of H = 128.8. Thus, A* chose to remove node B from the frontier to expand the subsequent nodes C, D, E, F, G along the non-optimal path.



In conclusion, the above experiments have shown that an overestimate for $h(n)$ causes A* search accuracy to worsen as A* cannot guarantee to find the optimal solution in all cases.