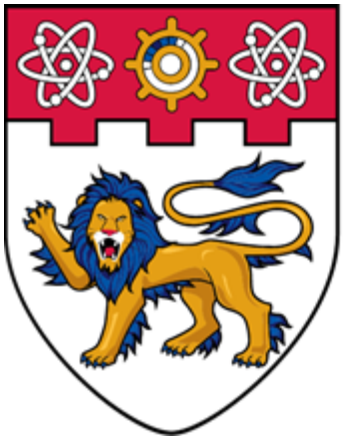


NANYANG TECHNOLOGICAL UNIVERSITY
SINGAPORE

CZ3005: Artificial Intelligence

LAB Assignment 1

By

Name: Datta Anusha

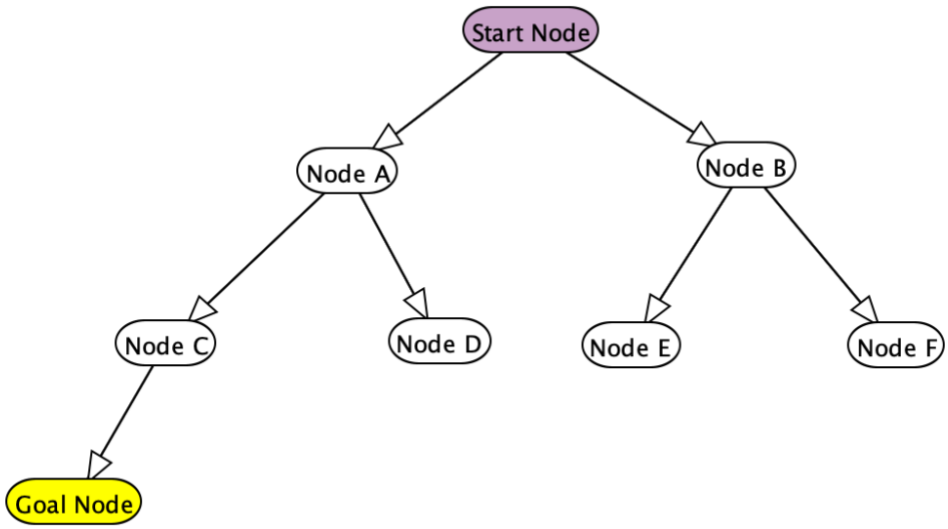
Matriculation Number: U1822948G

Lab Group: TS8

Date: 3rd March 2021

School of Computer Science and Engineering (SCSE)

Question 1(a) Give a graph where depth-first search (DFS) is much more efficient (expands fewer nodes) than breadth-first search (BFS).



Depth First Search

The DFS algorithm starts at the root node and explores as far as possible along each branch, before backtracking and traversing the other nodes.

- Order of Search
  Start Node → Node A → Node C → Goal Node

- Nodes Traversed
  4 nodes

Breadth First Search

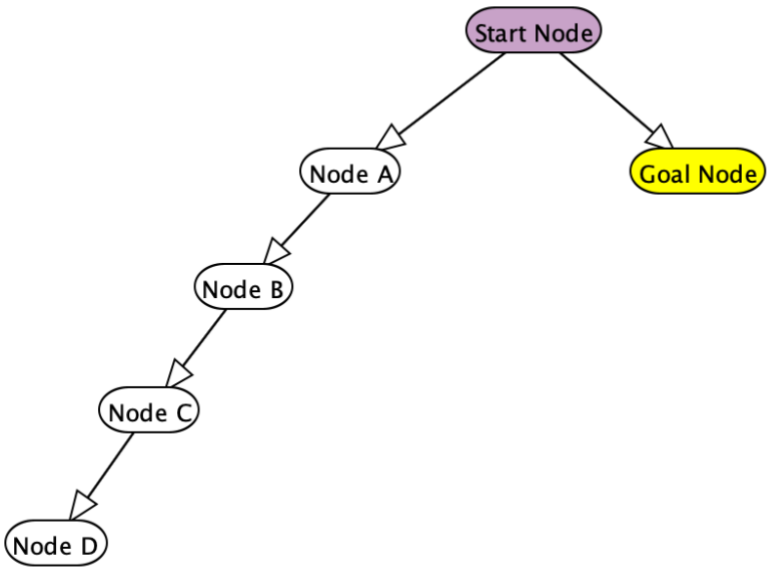
The BFS algorithm starts at the tree root, and explores all of the neighbour nodes at the present depth prior to moving on to the nodes at the next depth level.

- Order of Search
  Start Node → Node A → Node B → Node C → Node D → Node E → Node F → Goal Node

- Nodes Traversed
  8 nodes

Hence, Depth First Search is more efficient than Breadth First Search.

Question 1(b) Give a graph where BFS is much better than DFS.



Depth First Search

The DFS algorithm starts at the root node and explores as far as possible along each branch, before backtracking and traversing the other nodes.

- Order of Search
  Start Node → Node A → Node B → Node C → Node D → Goal Node

- Nodes Traversed
  6 nodes

Breadth First Search

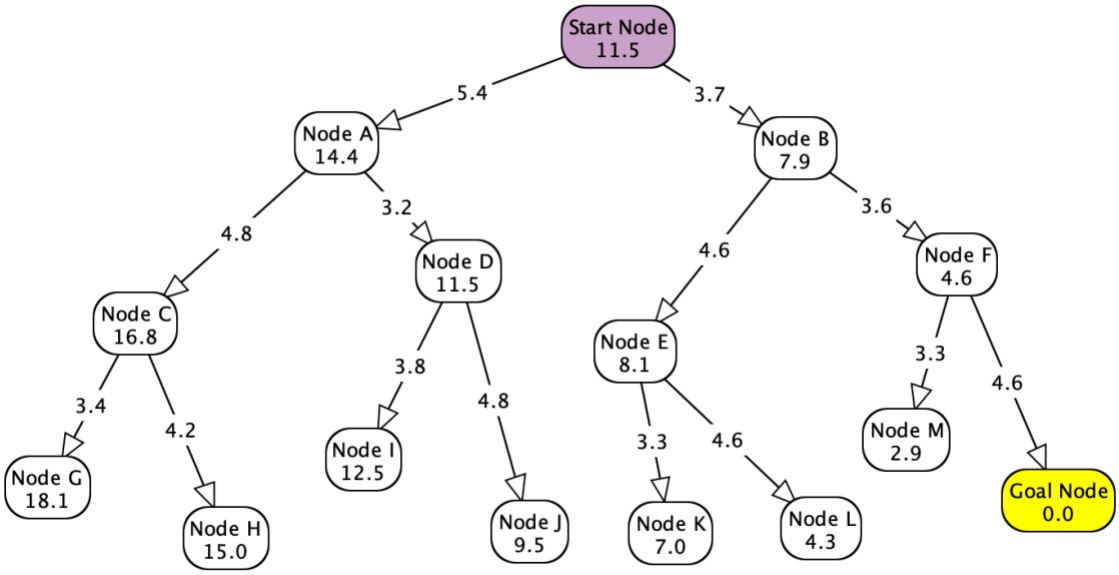
The BFS algorithm starts at the tree root, and explores all of the neighbour nodes at the present depth prior to moving on to the nodes at the next depth level.

- Order of Search
  Start Node → Node A → Goal Node

- Nodes Traversed
  3 nodes

Hence, Breadth First Search is more efficient than Depth First Search.

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



Arc costs and heuristic function values are labelled on the arcs and nodes of the graph respectively. Both the arc costs and heuristic function values are auto-generated using AIspace software, which is based on relative distance on the drawing.

Depth First Search

The DFS algorithm starts at the root node and explores as far as possible along each branch, before backtracking and traversing the other nodes.

- Order of Search
  Start Node → Node A → Node C → Node G → Node H → Node D → Node I → Node J → Node B → Node E → Node K → Node L → Node F → Node M → Goal Node

- Nodes Traversed
  15 nodes

Breadth First Search

The BFS algorithm starts at the tree root, and explores all of the neighbour nodes at the present depth prior to moving on to the nodes at the next depth level.

- Order of Search
  Start Node → Node A → Node B → Node C → Node D → Node E → Node F → Node G → Node H → Node I → Node J → Node K → Node L → Node M → Goal Node

- Nodes Traversed
  15 nodes

A* Search

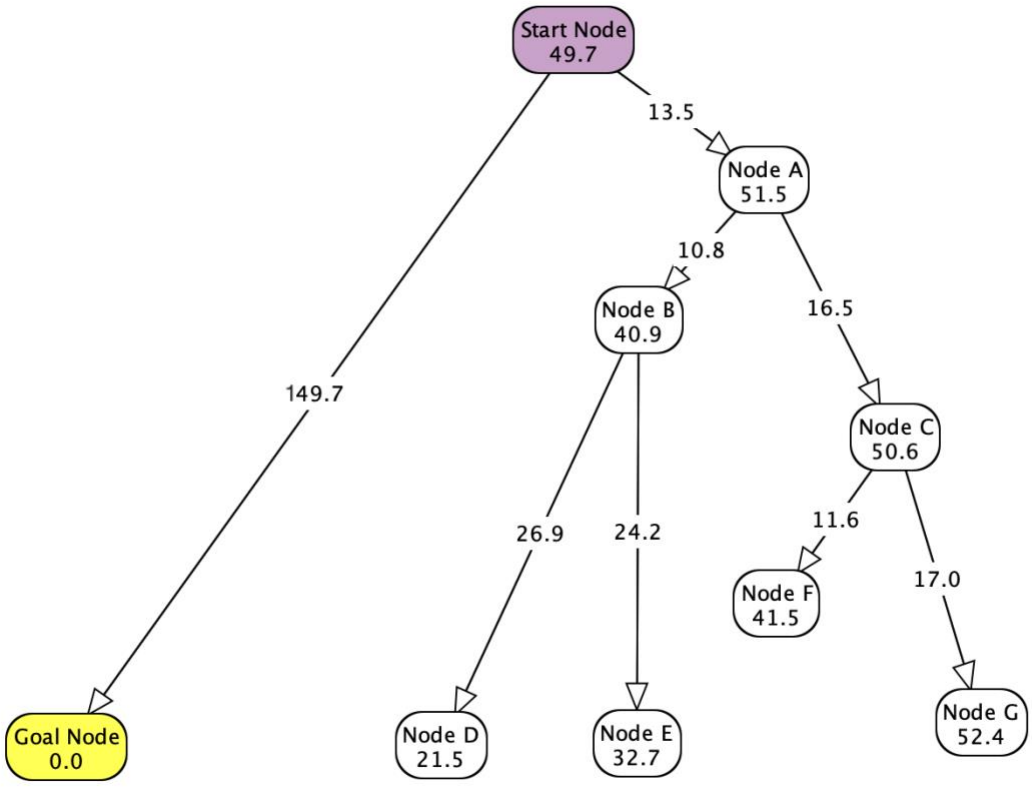
A* is an informed search algorithm, as it uses both path cost function and a heuristic function to compute the solution. The evaluation function f(n) of A* search can be represented as [$f(n) = g(n) + h(n)$]

- Order of Search
  Start Node → Node B → Node F → Goal Node

- Nodes Traversed
  4 nodes

Hence, A* Search is more efficient than both Breadth First Search and Depth First Search.

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



Arc costs and heuristic function values are labelled on the arcs and nodes of the graph respectively. Both the arc costs and heuristic function values are auto-generated using AIspace software, which is based on relative distance on the drawing.

Depth First Search

The DFS algorithm starts at the root node and explores as far as possible along each branch, before backtracking and traversing the other nodes.

- Order of Search
  Start Node → Goal Node

- Nodes Traversed
  2 nodes

Breadth First Search

The BFS algorithm starts at the tree root, and explores all of the neighbour nodes at the present depth prior to moving on to the nodes at the next depth level.

- Order of Search
  Start Node → Goal Node

- Nodes Traversed
  2 nodes

A* Search

A* is an informed search algorithm, as it uses both path cost function and a heuristic function to compute the solution. The evaluation function f(n) of A* search can be represented as [$f(n) = g(n) + h(n)$]

- Order of Search
  Start Node → Node A → Node B → Node D → Node E → Node C → Node F → Node G → Goal Node

- Nodes Traversed
  9 nodes

Hence, both Breadth First Search and Depth First Search are more efficient than A* Search.

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

As mentioned above, A* Search uses both path cost function and a heuristic function to compute the solution. The evaluation function f(n) of A* search can be represented as:

$$f(n) = g(n) + h(n)$$

Additionally, we know that A* search is:

- Optimal, as it uses an admissible heuristic to find the most optimal path
- Complete, as long as there is not an infinite number of nodes to be explored

Hence, as long as h(n) is lower (underestimation) than the actual cost from a given node to the goal node, A* search is still guaranteed to be optimal and complete. This is because the algorithm will still be successful in maintaining a priority queue of potential node expansions that it will consider, ranked by their computed optimality. The algorithm will continue the search until it finds a route to the goal node that is much more optimal than the other paths.

However, the larger the underestimation of the heuristic, the more A* search will behave like uniform-cost search, and might be very inefficient. This is because the other nodes, which are non-optimal, might appear to be more optimal than they truly are. As such, A* search will be misguided into traversing these paths, as it traverses further and further away from goal node, and thus, expanding more nodes than required.

To study the effect of heuristic accuracy, we shall consider the three following cases:

- No heuristic underestimation
- Heuristic underestimation of 50%
- Heuristic underestimation of 100%

No heuristic underestimation

In Figure 1.1, arc costs and heuristic function values are labelled on the arcs and nodes of the graph respectively. Both the arc costs and heuristic function values are auto-generated using AIspace software, which is based on relative distance on the drawing.

A* Search:

- Order of Search
  Start Node → Goal Node

- Nodes Traversed
  2 nodes

With only 2 node traversed, the path taken from the start node to goal node is optimal given no heuristic underestimation.

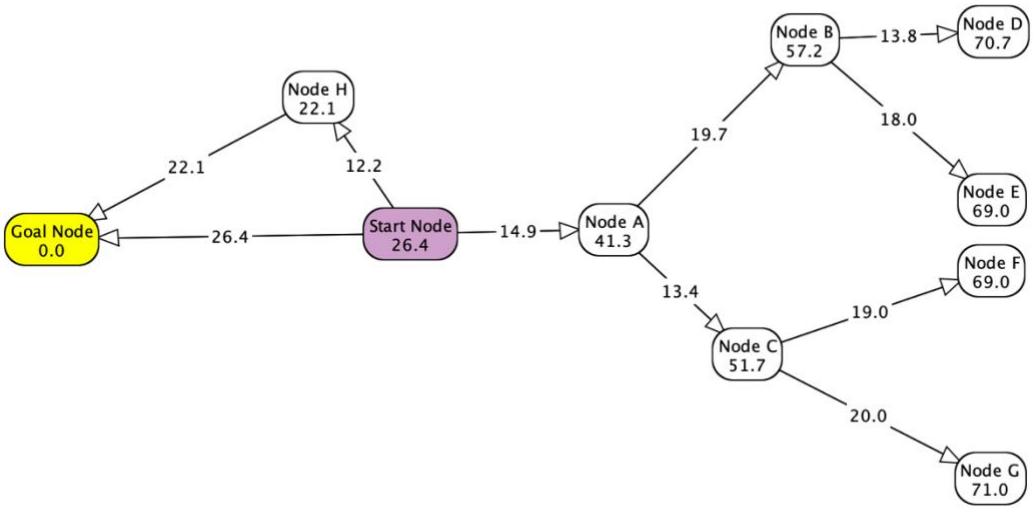


Figure 1.1: A* Search with exact h(n) auto generated with AIspace

Heuristic underestimation of 50%

Using the graph Figure 1.1, graph in Figure 1.2 has been derived by implementing an underestimation of h(n) by 50%.

A* Search:

- Order of Search
  Start Node → Node H → Goal Node

- Nodes Traversed
  3 nodes

In this case it may be observed that A* search will reach the goal node by expanding 3 nodes, which is 1 more (Node H) than in the case above. Despite

Node H being explored, it is discovered that the optimal path is still directly from the start node to the goal node, and thus, A* search is still accurate and optimal, but slightly less efficient.

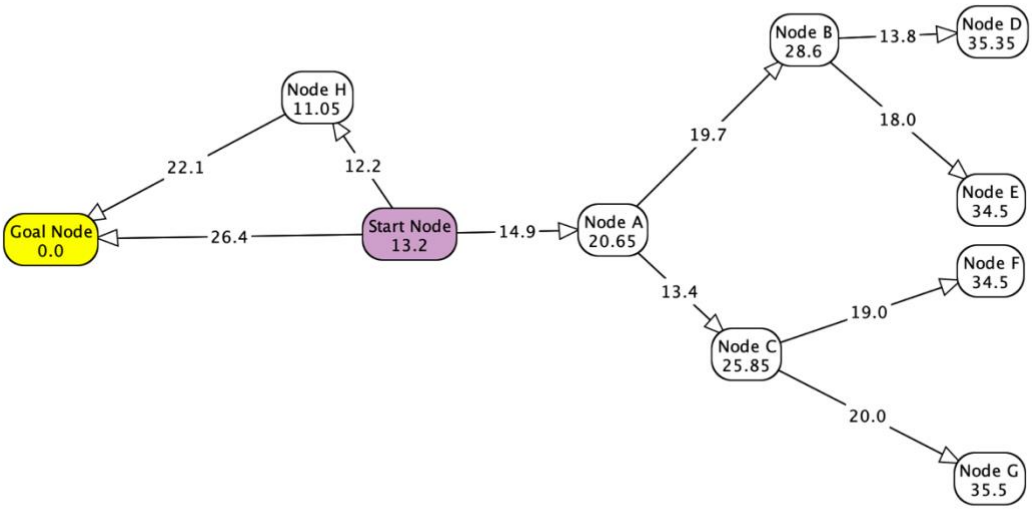


Figure 1.2: A* Search with h(n) underestimation of 50%

Heuristic underestimation of 100%

Using the graph Figure 1.1, graph in Figure 1.2 has been derived by implementing an underestimation of h(n) by 50%.

In the extreme case where the underestimation is 100%, h(n) will hold a very small value that may be approximated to 0. Hence, we observe that the evaluation function f(n) = g(n) when h(n) → 0, and that A* search will behave like Dijkstra's greedy algorithm, which is still guaranteed to find the shortest optimal path, but can be very inefficient as it expands multiple nodes that don't lead to the optimal path.

A* Search:

- Order of Search
  Start Node → Node H → Node A → Node C → Node F → Node B → Node D → Node E → Node G → Goal Node

- Nodes Traversed
  10 nodes

In this case it may be observed that A* search will reach the goal node by expanding 10 nodes, which is much higher than in the cases above and very inefficient. However, the optimal path is still directly from the start node to the goal node, and thus, A* search is still accurate and optimal.

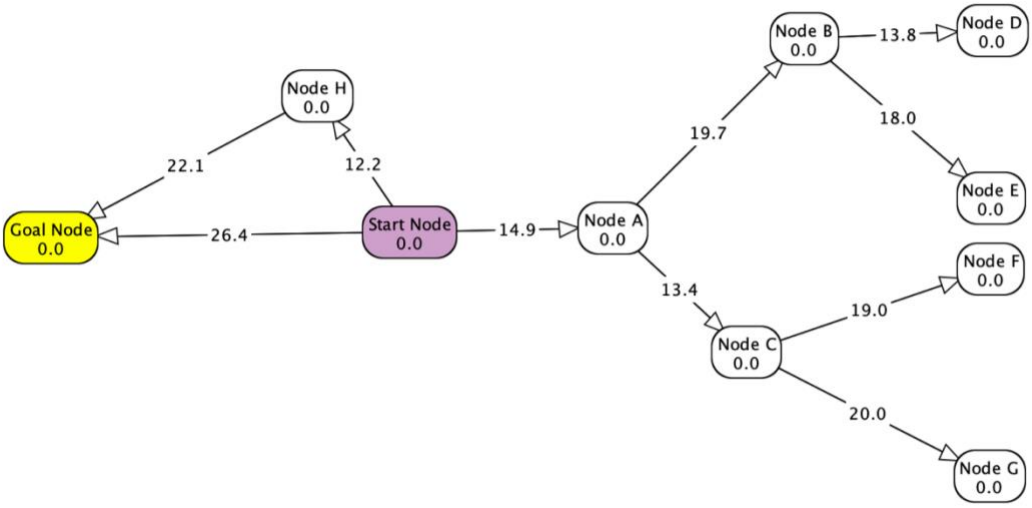


Figure 1.3: A* Search with h(n) underestimation of 100%

Conclusion

Hence, for an underestimation of h(n) which is actual cost from a given node to the goal node, it can be concluded that A* search shall

- Always return a complete and optimal solution
- Be increasingly inefficient as underestimation of heuristic value increases

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

If h(n) is equal to the exact distance from given node to the goal node, that would be the ideal case as the heuristic would be perfectly accurate. Hence, the A* algorithm would always traverse the most optimal path, yielding the highest efficiency in computing the solution.

As mentioned above, A* search is guaranteed to produce an optimal and complete solution. Also, A* Search uses both path cost function and a heuristic function to compute the solution. The evaluation function f(n) of A* search can be represented as:

$$f(n) = g(n) + h(n)$$

This idea can be further explored by considering the graph in Figure 2 below. In Figure 2, arc costs and heuristic function values are labelled on the arcs and nodes of the graph respectively. Both the arc costs and heuristic function values are auto-generated using AIspace software, which is based on relative distance on the drawing.

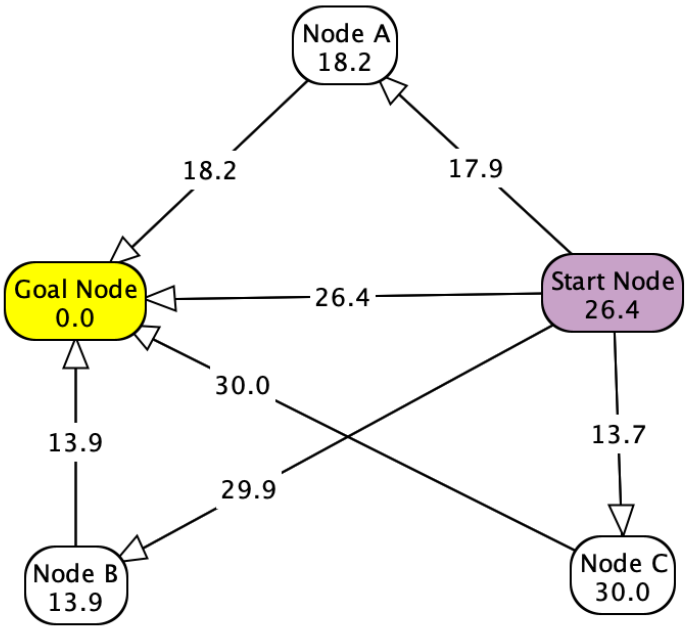


Figure 2: A* Search with exact h(n) auto generated with AIspace

Start Node → Goal Node (Optimal Path)

The optimal solution for the graph in Figure 2 is the direct path from the start node to the goal node.

- A* Search Path Cost = 26.4
- A* Search Evaluation Function [f(n) = g(n) + h(n)]
  f(n) = 26.4 + 0.0 = 26.4

Start Node → Node A → Goal Node

A* search will not traverse this path:

- A* Search Path Cost = 17.9 + 18.2 = 36.1
  Since 36.1 is larger than 26.4, it is not the optimal path.

- A* Search Evaluation Function [f(n) = g(n) + h(n)]
  Compute evaluation function value from start node to Node A,
  f(n) = 17.9 + 18.2 = 36.1

As the evaluation function f(n) of the start node to Node A is larger than that of the direct path, the direct path is chosen instead, and there is actually no need to further calculate the f(n) from Node A to the goal node.

Start Node → Node B → Goal Node

A* search will not traverse this path:

- A* Search Path Cost = 29.9 + 13.9 = 43.8
  Since 43.8 is larger than 26.4, it is not the optimal path.

- A* Search Evaluation Function [f(n) = g(n) + h(n)]
  Compute evaluation function value from start node to Node B,
  f(n) = 29.9 + 13.9 = 43.8

As the evaluation function f(n) of the start node to Node B is larger than that of the direct path, the direct path is chosen instead, and there is actually no need to further calculate the f(n) from Node B to the goal node.

Start Node → Node C → Goal Node

A* search will not traverse this path:

- A* Search Path Cost = 13.7 + 30.0 = 43.7
  Since 43.7 is larger than 26.4, it is not the optimal path.

- A* Search Evaluation Function [f(n) = g(n) + h(n)]
  Compute evaluation function value from start node to Node C,
  f(n) = 13.7 + 30.0 = 43.7

As the evaluation function f(n) of the start node to Node C is larger than that of the direct path, the direct path is chosen instead, and there is actually no need to further calculate the f(n) from Node C to the goal node.

Conclusion

Hence, from the above cases, it may be concluded that with exact heuristics it is impossible to find a node such that the evaluation function f(n) value of the node is smaller than that of a node on the optimal path. Therefore, exact heuristics, where h(n) is equal to the actual cost from a node to the goal node, will always lead to A* search algorithm finding optimal path with highest efficiency.

In conclusion, A* search shall:

- Always return a complete and optimal solution
- More efficient than if there's heuristics underestimation

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

As mentioned above, the evaluation function f(n) of A* search can be represented as:

$$f(n) = g(n) + h(n)$$

As such, the larger the overestimation of h(n), the more A* search will behave like greedy search. Hence, it might or might not find the shortest path, which makes it a non-optimal approach. However, it can be very fast and can reduce search space considerably.

To study this effect of heuristic overestimation, we shall consider the two following cases:

- No heuristic underestimation
- Heuristic overestimation of 50%

No heuristic underestimation

In Figure 3.1, arc costs and heuristic function values are labelled on the arcs and nodes of the graph respectively. Both the arc costs and heuristic function values are auto-generated using AIspace software, which is based on relative distance on the drawing.

- Optimal Path

  Start Node → Node D → Node C → Node B → Node A → Goal Node

- Path Cost

  7.5 + 10.8 + 12.0 + 9.9 + 10.6 = 50.8

- Order of Search

  Start Node → Node D → Node C → Node B → Node A → Goal Node

- Nodes Traversed

  6 nodes

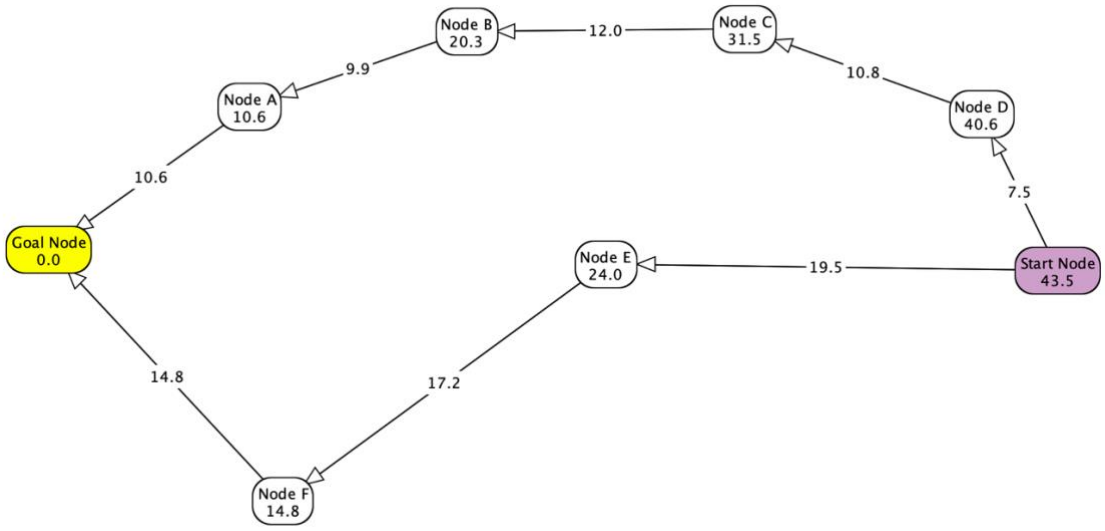


Figure 3.1: A* Search with exact h(n) auto generated with AIspace

Heuristic overestimation of 50%

Using the graph Figure 3.1, graph in Figure 3.2 has been derived by implementing an overestimation of h(n) by 50%.

For this case, A* search will result in the traversal of a non-optimal path as shown below:

- Path Cost

  19.5 + 17.2 + 14.8 = 51.5

- Order of Search

  Start Node → Node E → Node F → Goal Node

- Nodes Traversed

  4 nodes

As observed, in comparison to exact heuristics, an overestimation can sometimes result in a non-optimal path as the solution, but at the same time, being faster and more efficient as it expands less nodes.

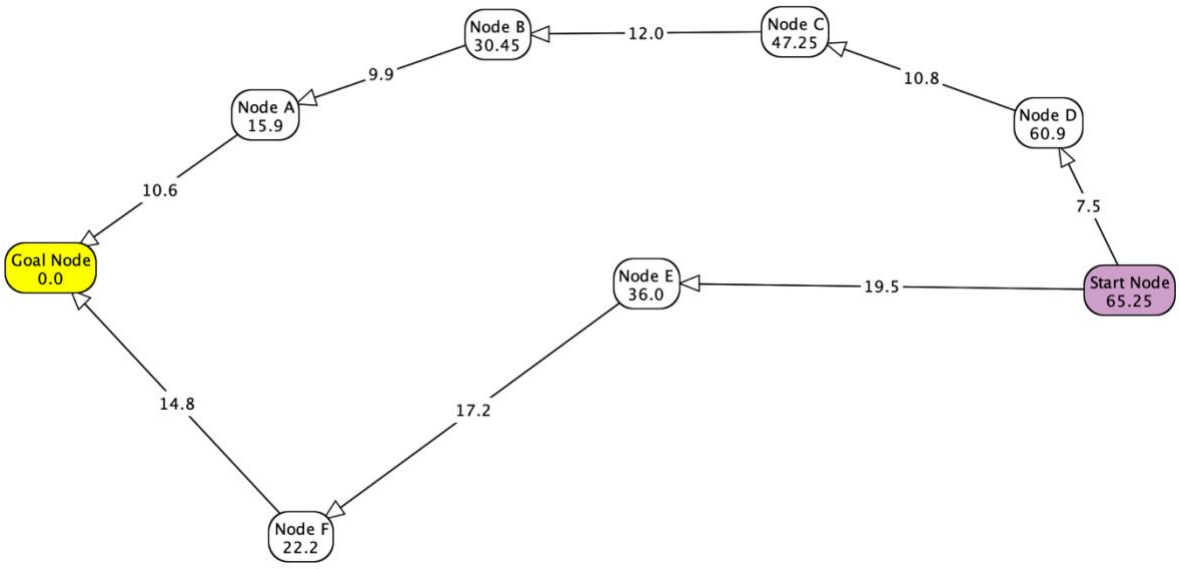


Figure 3.2: A* Search with exact h(n) overestimation of 50%

Conclusion

Hence, for an overestimation in heuristic function h(n), which is the actual cost from a given node to the goal node, it may be concludes that A* search shall:

- Sometimes produce an non-optimal solution
- Sometimes be more efficient