

School of Computer Science & Engineering

CZ3005 Artificial Intelligence

Lab Report 01

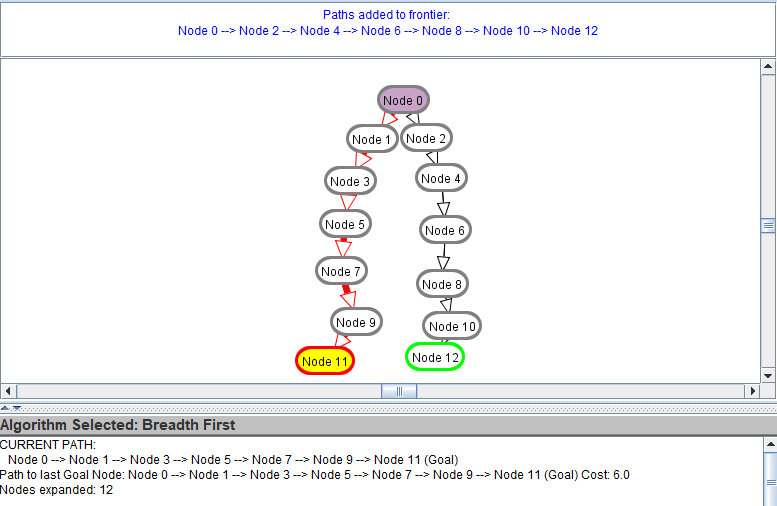
Benjamin Tan

SSP3

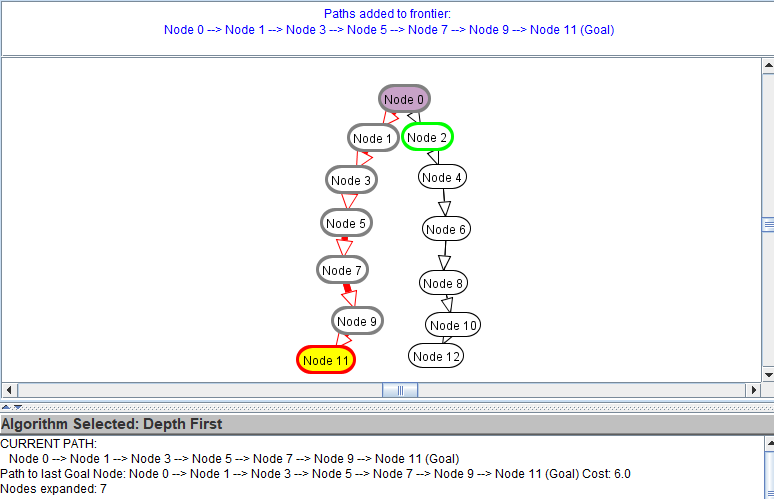
1) Each answer that is part of this question will be answered with a simple graph that is searched through by the algorithms mentioned. Note that start nodes are highlighted in purple and goal nodes are highlighted in yellow.

1a)

Here is a graph searched with BFS, with 12 expanded nodes.

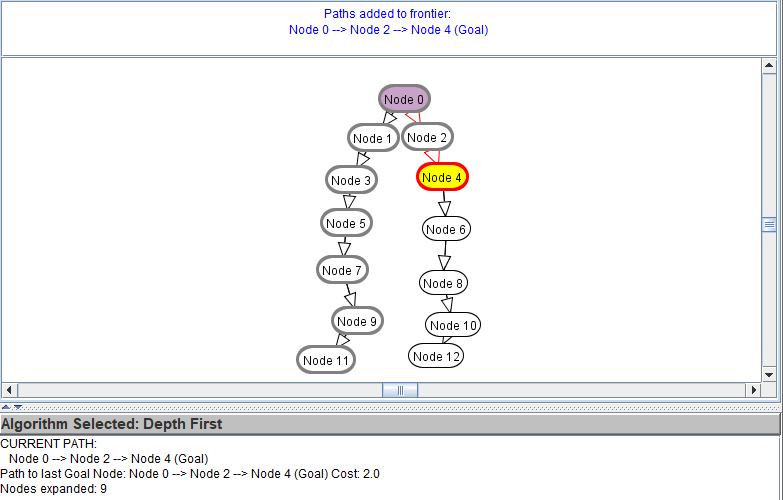


Here is the same graph searched with DFS, with 7 expanded nodes.

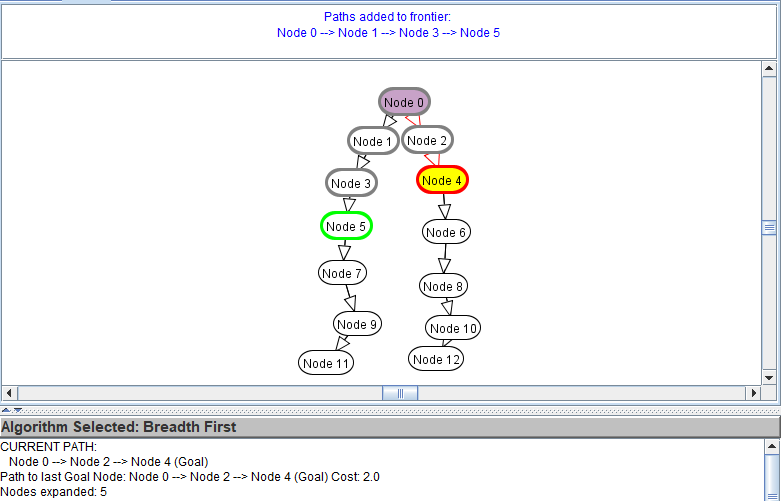


1b)

Here the graph is searched with DFS, with 9 expanded nodes.

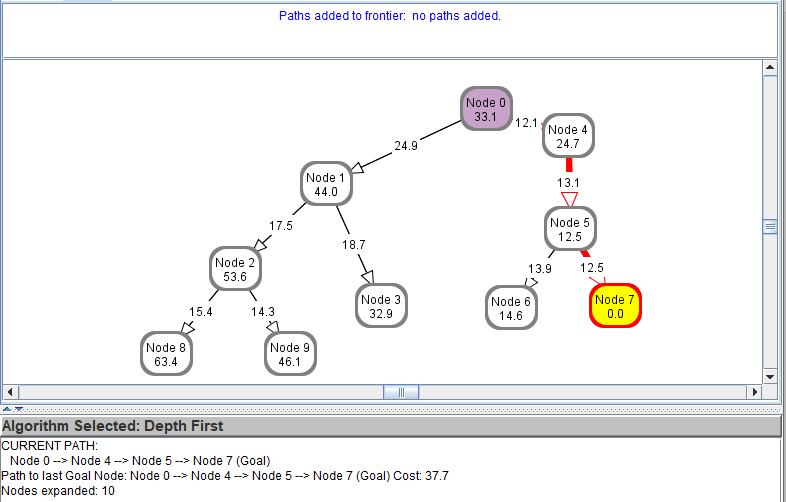


Here is the same graph searched with BFS, with 5 expanded nodes.

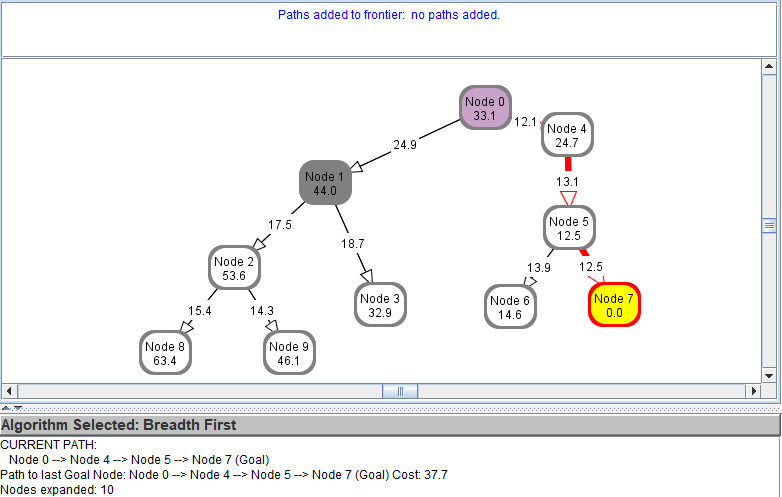


1c)

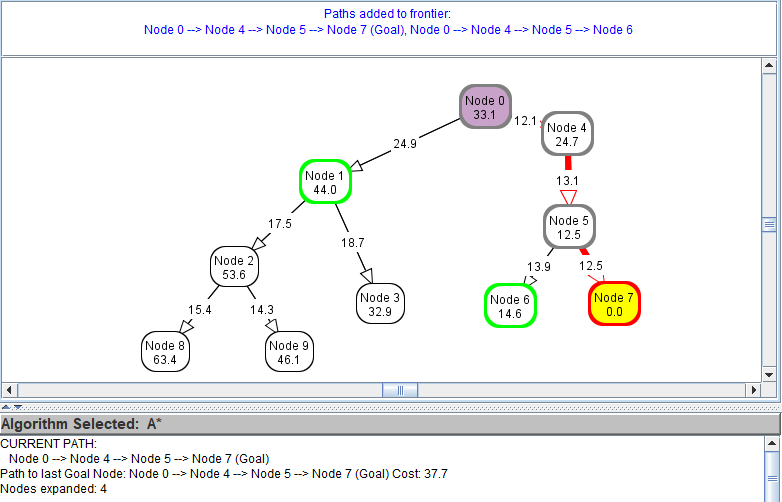
Here is a graph searched with DFS. Number of expanded nodes is 10.



Here is the same graph searched with BFS. Number of expanded nodes is 10.

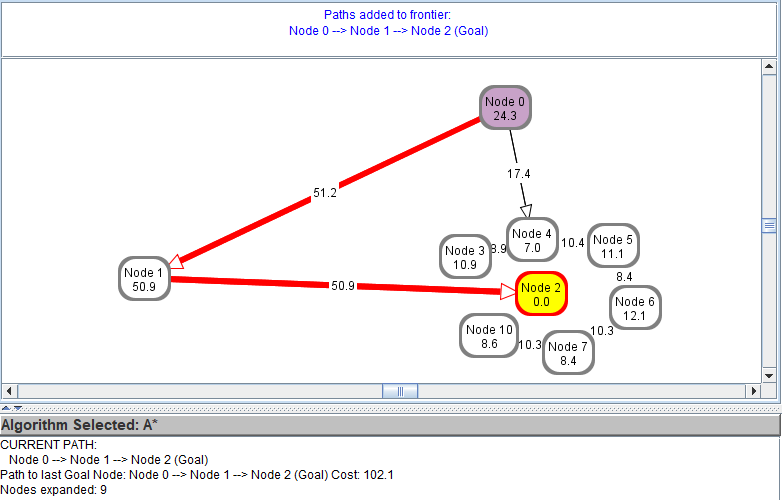


Here is the same graph searched with A\* search, with 4 expanded nodes.

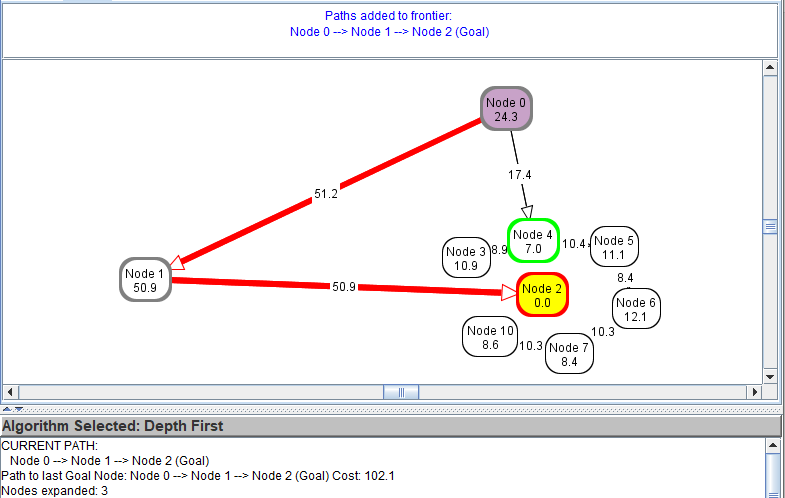


1d)

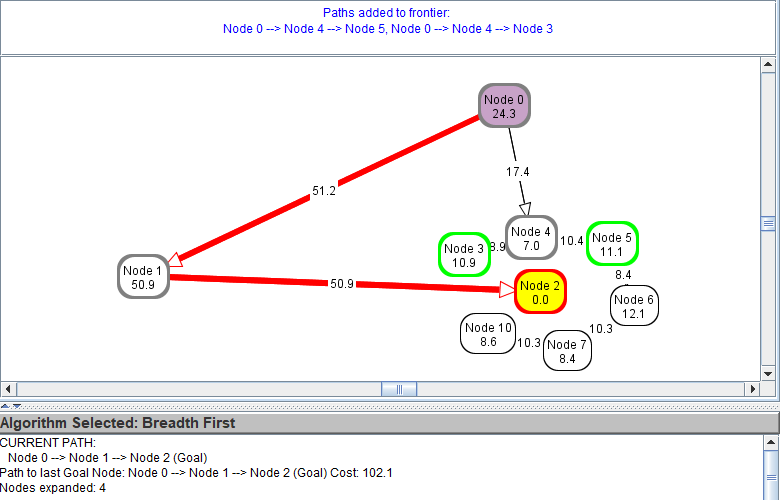
Here is a graph searched through with A\* search. Number of expanded nodes is 9.



Here is the same graph searched with DFS. Number of expanded nodes is 3.



Here is the same graph searched with BFS. Number of expanded nodes is 4.



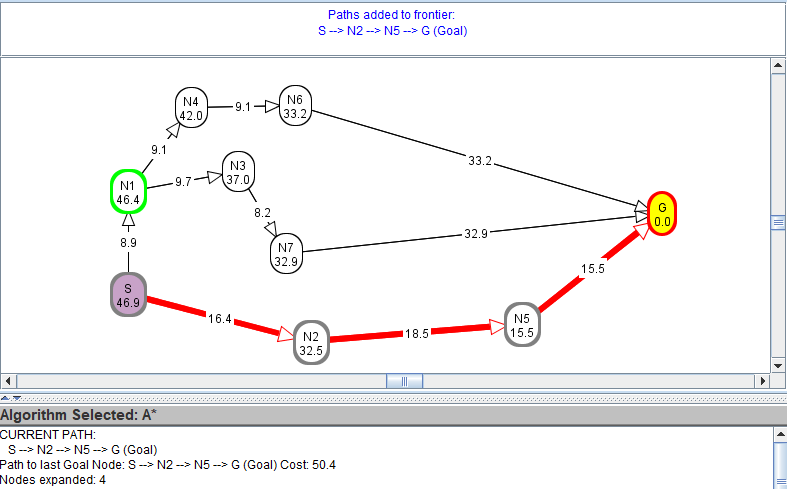
2a)

Conjecture

Given that h(n) is already an underestimate and we reduce h(n) by some factor x, A\* search is still admissible, that is to say, the real path cost is not less than h(n). Assume that the real cost f(n) of a node to reach the goal state is g(n)+h(n). When h(n) is reduced, then the evaluation function will become g(n) + x\*h(n), where x<1. Consider a node with an optimal path with cost g(n1) when reaching the goal state and a non-optimal path with g(n2), and so g(n1) < g(n2). For the search to find the optimal path, f(n1) < f(n2) must be true, and so requires g(n2) - g(n1) > h(n1) – h(n2) to be true. The statement still remains true when we reduce the heuristic costs by a factor of x, where x < 1. We can therefore deduce that the search is still optimal, meaning that it will always return the shortest path to the goal. However, the efficiency of the search may be impacted. The search will be more likely to expand paths with the smallest g(n), which may not necessarily lead to the path to the goal, leading to more nodes expanded. For example, if there is a path leading to N1 with less cost than another path leading to N2, then the algorithm will be more likely expand N1 node first, even though N2 is the only node that leads to the goal state.

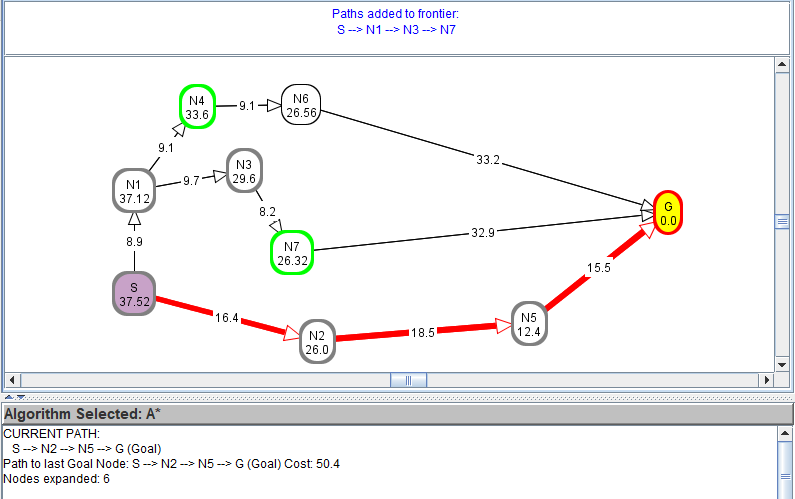
Empirical evidence

The example below shows a A\* search with the h(n) values of the nodes at original values. Optimal path is S-> N2 -> N5 -> G, with a cost of 50.4.



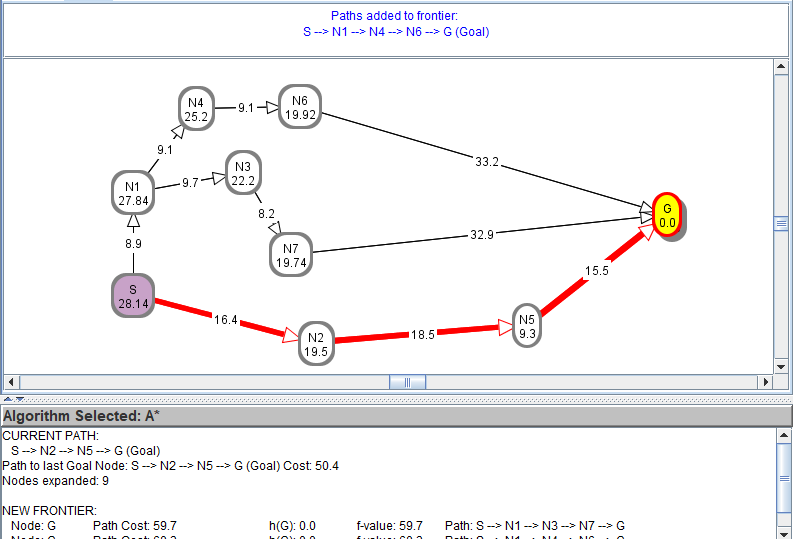
Here the optimal path S->N2->N5->G is found, with 4 expanded nodes.

When we reduce h(n) by a factor of 0.8 in the above graph, we get:



Here we still get the optimal path, but the number of expanded nodes increases from 4 to 6.

Further reducing h(n) by a factor of 0.75 leads to:



Again, the optimal path is found, but the number of expanded nodes increases from 6 to 9.

Conclusion

Reducing h(n) when h(n) is already an underestimate still guarantees the optimality of A\* search, since the search is still admissible. However, the search becomes less efficient and expands more nodes. Note that it only becomes less efficient in general; for example, for example, if the best path to the goal has a g(n) than the alternate paths, then even after reducing h(n), the only nodes expanded will be those along the best path, and thus A\* search would still work at max efficiency.

In our examples, efficiency decreased because the alternate paths had lower initial g(n) than the other paths, and so the algorithm expanded the alternate paths instead.

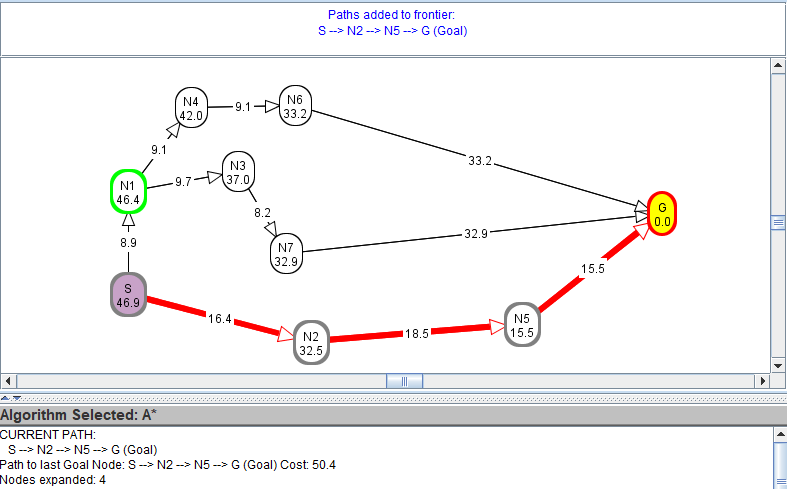
2b)

Conjecture

When h(n) is the exact distance to a goal, A\* search will always find the least cost path, since it is still admissible, meaning that the real path cost is not less than h(n). However, A\* search will not always expand nodes in the most efficient manner. This is especially likely if the exact distance heuristic is misleading – that is, there is no path leading to the goal node from a specific node, even though that specific node has the closest actual distance to the goal node. However, it will usually expand nodes in the most efficient manner.

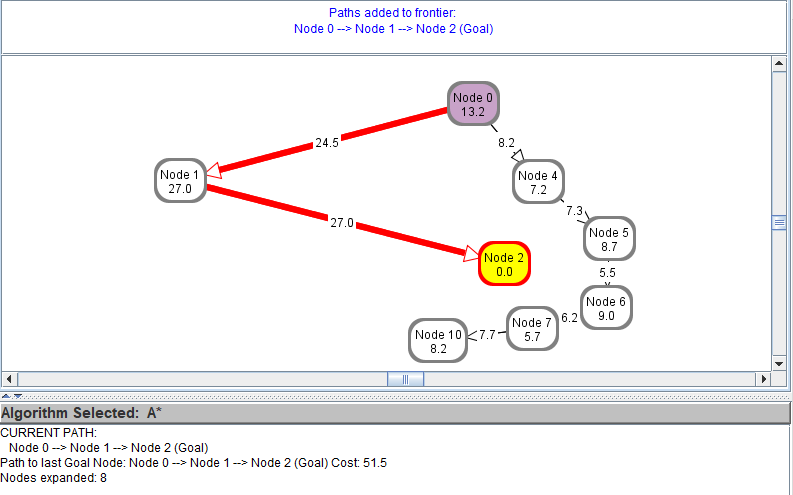
Empirical evidence

The example below shows a A\* search with the h(n) values of the nodes with exact distances to the goal. Optimal path is S-> N2 -> N5 -> G, with a cost of 50.4.



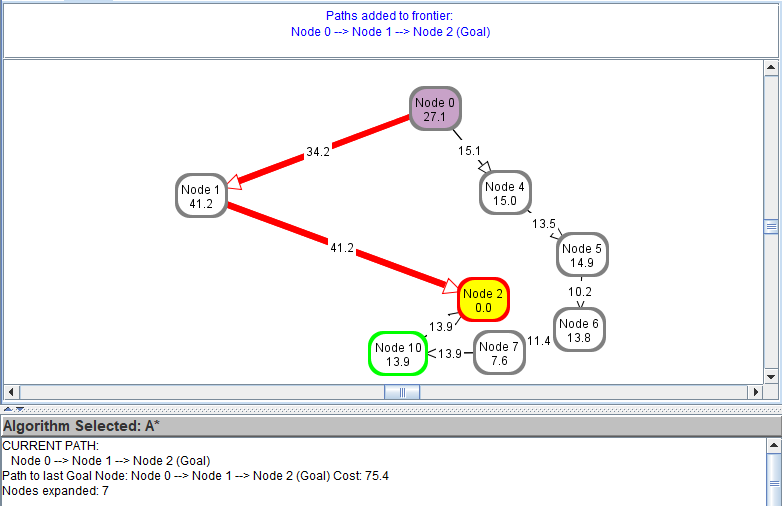
Here the optimal path S->N2->N5->G is found, with 4 expanded nodes. The search operates at max efficiency, expanding only nodes that lie along the optimal path. Note that this is because f(n) of the expanded nodes is always lower than the f(n) of N1, and so the algorithm will never expand N1.

Now let us consider another case where A\* search fails to be efficient even though h(n) = exact distance to goal.



In this graph, there is only one path to the goal, 0>1>2, and h(n) = exact distance to goal for each node. The most efficient number of nodes to be expanded is 3. Even so, all of the nodes are expanded before A\* search finds that path. This is because the graph is misleading, as the closer nodes to the goal do not have a path to a goal.

Even if the graph is misleading, however, the search will find the least cost path. Below, the least cost path is 0 > 1 > 2 with a path cost of 75.4. Although more nodes are expanded than needed, the search still remains optimal.



Conclusion

Setting h(n) to be the exact distance to the goal means that h(n) is always less than the actual path cost to the goal, and so A\* star search remains admissible. This means that A\* search is always optimal for these values of h(n). Also, in general, A\* search is efficient so as long as the graph is not misleading. This is more so in real-world cases, where nearer nodes are likely to have a path to the goal. A\* search will usually expand nodes along the optimal path in such cases. However, in certain exceptional misleading cases, A\* star will expand far more nodes than needed. Even then, however, A\* search will always find the least cost path, and so remains optimal.

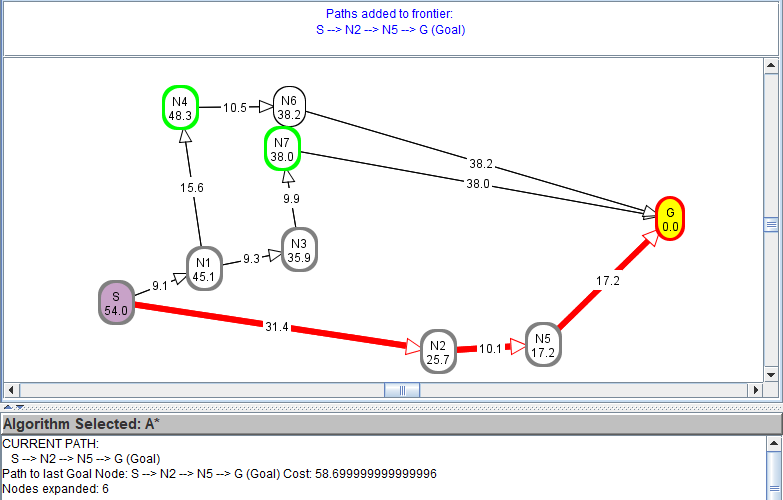
2c)

Conjecture

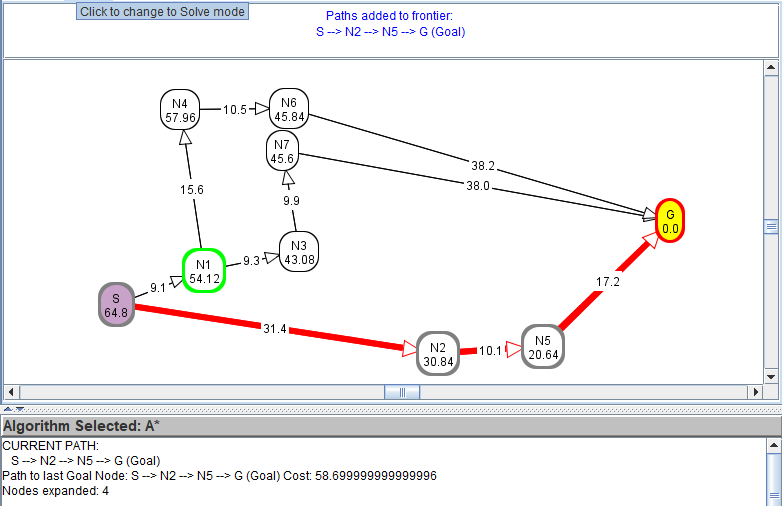
When h(n) is not an underestimate, then it is likely that the real path cost is less than h(n). The result of this is that A\* search fails the criteria of admissibility, and so would not be optimal, in the sense that it will not find the least cost path to the goal every time. In exchange, however, the search becomes more efficient – that is, it will generally expand less nodes to find the goal. This is implied by the conclusion of 2a), where reducing h(n) made the search less efficient.

Empirical evidence

First let us show that if h(n) is not an underestimate, then A\* search is generally more efficient in expanding nodes. For example, below is a graph with h(n) = exact distance to goal. Optimal path is S->N2->N5->G, and so the minimum number of nodes to expand is 4. However, A\* search expands 6 nodes.

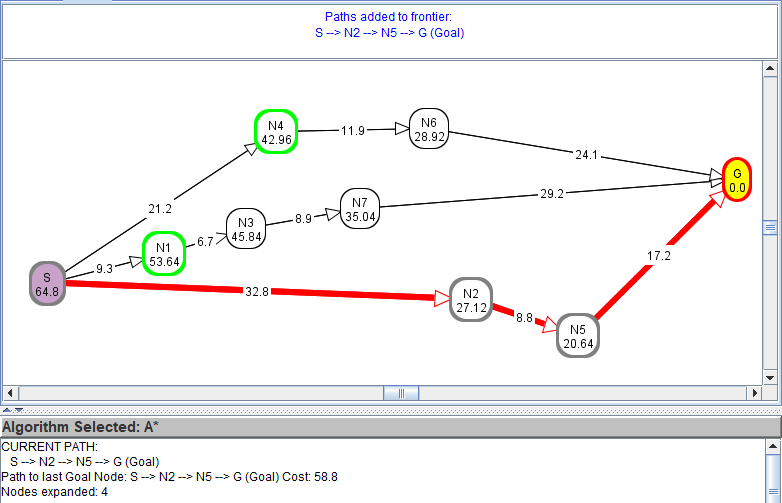


Let us increase h(n) by a factor of 1.2 and retry the A\* search.



The optimal path is returned, and the number of nodes expanded decreases from 6 to 4. Thus, it can be said that if h(n) is not an underestimate, the search is generally more efficient. Note that if the graph is misleading, like in 2(b), then it is possible that the efficiency may worsen instead.

Also, since h(n) is not an underestimate, it will not always give the least cost path. An example where the node heuristics is multiplied by 1.2:



The actual least path cost is S -> N1 -> N3 -> N7 -> G, with a cost of 54.1. Here the search returns the alternate path of S -> N2 -> N5 -> G with a higher cost of 58.8. Therefore, it can be said that the search is not optimal.

Conclusion

When h(n) is not an underestimate, h(n) becomes a non-admissible heuristic, and this will result in an overestimate of evaluation functions f(n). This can result in A\* search returning a non-least cost result, and this means that A\* search will not be optimal.

On the other hand, in general, search efficiency increases – that is, the number of nodes expanded usually decreases. This usually occurs when A\* search fails to operate at max efficiency using an underestimate of h(n). If the graph is misleading, however, the number of nodes expanded may increase instead.