Enrique Belenguer

40406742@live.napier.ac.uk

Edinburgh Napier university – Artificial Intelligence (SET09122)

Contents

1. Research

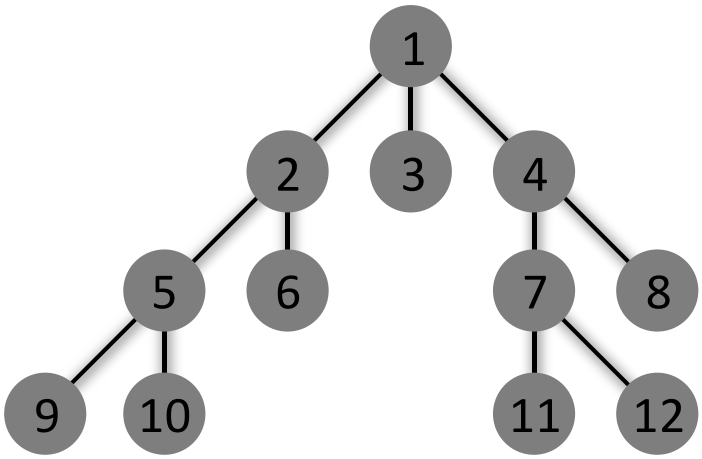
1.1. Breadth-First-Search

The Breadth-First Search is a simple strategy in which first the root node expands and afterwards all the successors of the root node are expanded, then their successors, and so on.

In general, initially all nodes are expanded to a depth in the search tree before expanding any node from the next level.

The Breadth-First-Search can be implemented using a queue type structure first enter first out, making sure that the first nodes visited will be the first ones expanded. The main disadvantage of the BFS is the memory requirements to store all the nodes that have not been expanded during the search.

Here there is an example.



[1]

Starting Node → 1

Once the starting node has been searched, 2, 3 and 4 are added to the queue.

If 2,3 and 4 are not the goal nodes, then 5,6,7 and 8 are explored in that order.

If 5,6,7 and 8 are not the goal nodes, then 9,10,11 and 12 are explored in that order.

1.2. A*

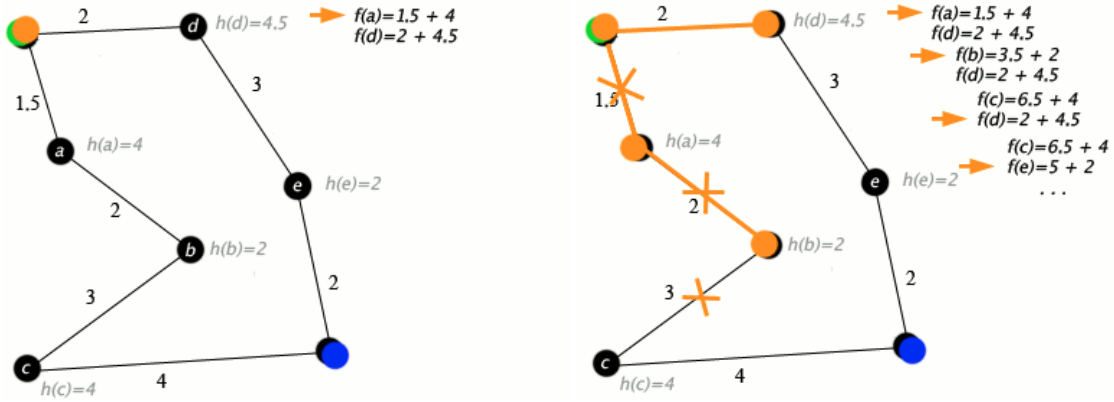
A* is probably the most widely known method of informed search algorithm. This method evaluates the nodes combining: *g(n)* (the cost to reach the node) and *h(n)* (the cost of going to the target node).

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

Since the g(n) gives us the cost of the path from the beginning node to the node n, and the *h(n)* the estimated cost of the cheapest way form n to the objective, we have: *f(n)* = estimated cheapest cost of the solution through n. So, if we try to find the cheapest solution, it is reasonable to first try the node with the lowest value of *g(n) + h(n)*. This is the most practical strategy.

This technique can be implemented with a priority queue (openList), using the heuristic function to obtain the ordering value.

To exemplify this strategy, I am going to use the following example.



[2]

1.Firstly, we add the first node, in our case the one with the green mark. to the openList, We calculate its heuristic function where: *g(n)=0* (It's the first node) and *h(n)=6*. Knowing g(n) and h(n) we could assume that *f(n)=6.*

2.We check if the actual Node is the node Objective (represented in our Diagram with a blue mark). In case it was, the search would finish, in this example it's not the objective so we procced to eliminate the node from the openList and add it to another list called closedList.

3.We add to the openList(now empty) the neighbours's nodes 'a' and 'd' if they are not already in the closedList. We must make sure that they are ordered by their f(n). In this case the *f(a) < f(d).* Afterward, we take the first node from the openList and we go back to the point two until find what we are looking for.

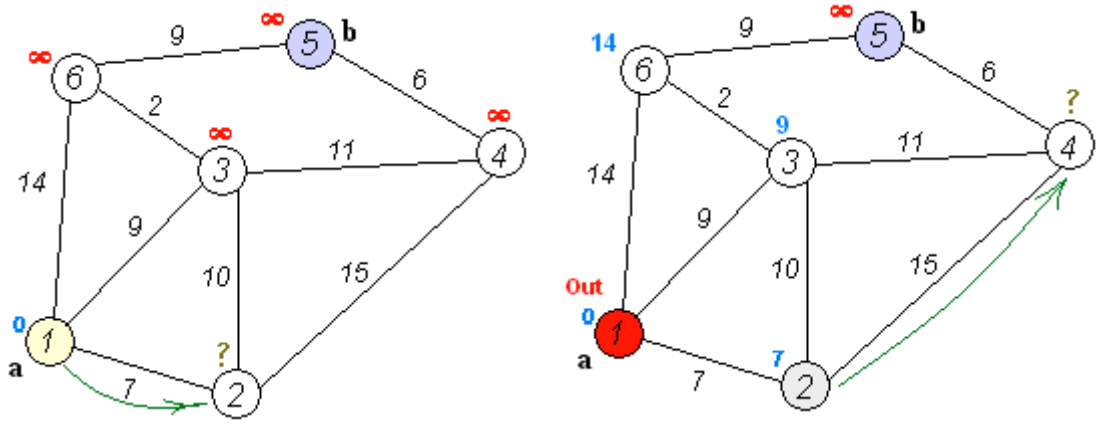
The A* strategy is very satisfactory, unfortunately, does not mean that A* is the answer to all our search needs. The difficulty is that, for most problems, the number of nodes to expand are exponential in solution length.

1.3. Dijkstra

The Dijkstra's algorithm is another well known method of informed search algorithm. It was invented by Edsger Dijkstra in 1959.

The underlying idea in this algorithm is to explore all the shortest paths that start from the vertex origin and that lead to all the other vertices; when the shortest path is obtained from the vertex origin, to the rest of the vertices that make up the graph, the algorithm stops. The algorithm is a specialization of the uniform cost search, and as such, it does not work in graphs with edges of negative cost (when choosing the node with less distance, they can be excluded from the search nodes that in future iterations lower the general cost of the road when going through an edge with negative cost).

Having a weighted directed graph of N non-isolated nodes, let x be the initial node, a vector D of size N will save the distances from x to the rest of the nodes at the end of the algorithm.



[3]

1. Initialize all distances in *D* with a relative infinite value since they are unknown at the beginning, except for the one of *x* that must be placed in 0 because the distance from *x* to *x* would be 0.

2.If *a=x* (we take *a* as the current node).

3. Remember all the adjacent nodes of a, except the marked nodes, we will call these vi.

4. If the distance from *x* to *vi* saved in *D* is greater than the distance from *x* to *a* added to the distance from a to *vi*; this is replaced with the second name, which is:

if $(D_i > D_a + d\ (a, vi))$ then $D_i = D_a + d\ (a, vi)$

5. We mark node *a* as complete.

6. We take as the next current node the one with the lowest value in *D* (it can be done by storing the values in a ordered list) and we return to step 3 while there are no unmarked nodes.

Once the algorithm is finished, *D* will be completely full.

We can estimate the computational complexity of the Dijkstra algorithm (in terms of sums and comparisons). The algorithm performs at the most n-1 iterations, since in each iteration a vertex is added to the distinguished set.

2. Conclusion

The search strategies give us an idea of how AI researchers propose different ways to solve problems. These techniques are basic to the AI and that is why they should be known by all those who are related to programming solutions by computer.

Each technique is different, all of them has advantages and disadvantages, and they should be used depending of the problem that you have to solve.

The BFS has two mainly advantages, the first is that it will find the solution if this one exit and secondly that is easy to code but the mainly disadvantage is that it requires a lot of memory which is not good if you are trying to solve a big problem.

The A* always find a good solution, with an optimal cost it does it pretty fast, but the main advantage is that get exponentially big, and it always need a good Heuristic to work well.

The main advantage of the Dijkstra algorithm is that most of the time it doesn't have to investigate all the edges, which save a lot of recourses, but the main disadvantage is that cannot handle negative edges.

In most of the strategies contemplated in the chapter, must be a strategy to avoid repeated state, this will save storage space and travel time. As Rusell-Norvig says in his book, "Algorithms that forget their history are doomed to repeat it". And that is why I decided to use the A* method.

For our problem, the best solution was using the A*, the problem was big, but not big enough to cause us any problem in terms of memory and also the use of the Heuristic would help to avoid expanding useless nodes, it would expand just nodes if it seems promising.

3. References

1. https://es.wikipedia.org/wiki/B%C3%BAsqueda_en_anchura

2. https://en.wikipedia.org/wiki/A*_search_algorithm

3. https://en.wikipedia.org/wiki/Dijkstra%27s_algorithm

4. Russell, Stuart , Norvig, Peter. Artificial Intelligence: A Modern Approach, Third Edition.