Artificial Intelligence Coursework

Marcos Ramirez Aceves
40284094@live.napier.ac.uk
Edinburgh Napier University - Artifical Intelligence (SET09122)

1 Search Strategies Description

1.1 Bidirectional Search

By knowing the Start and End nodes of the map route, one of the methods that we could use is a Bidirectional search strategy. The basic idea of this search strategy is to perform a search from two places at the same time, one search goes forward from the Start node and the other search goes backward from the End node. The reason to do this, is that $b^{d/2} + b^{d/2}$ is less than b^d (b is the maximum branching factor of the search tree and d is the depth of the optimal solution).

The way in which this strategy is implemented is by checking if the frontiers of the two searches intersect each time a node is expanded, and if that is the case then a solution has been found.

Finally, one of the things that we need to be aware of in Bidirectional search is that the forward search is easy to apply, however the backward search requires a method for computing the predecessors of each state, and in some kind of cases this can suppose a problem.

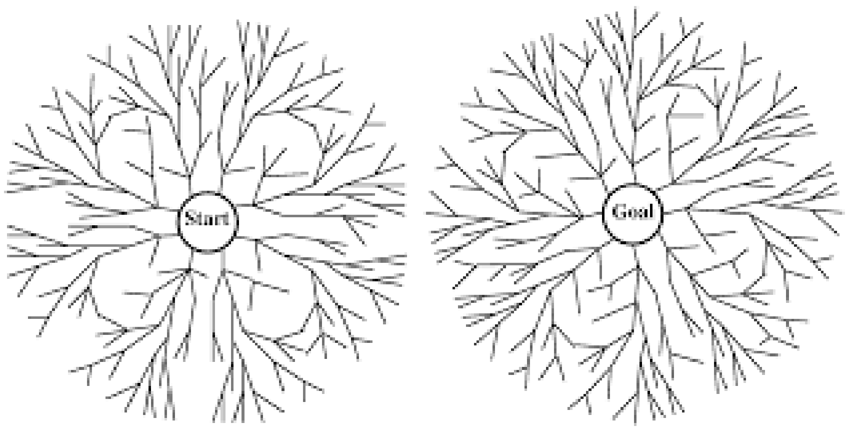

Figure 1: Bidirectional Search Diagram

1.2 A* Search

A* is the best form of the Best-First search and it's basic idea is to avoid expanding the more expensive paths. A* uses the following evaluation function : $f(n) = g(n) + h(n)$ where $h(n)$ is the estimated path cost from node n to the goal node, $g(n)$ is the path cost that we already have to get to the node n and $f(n)$ is the total estimated cost of the path to get to the goal node through the node n.

A* has the following ingredients or elements:

- OpenList (Frontier)
- ClosedList (Explored Set)
- ParentMap
- CurrentNode
- fScoreMap (Collection of all nodes $f(n)$)
- gScoreMap (Collection with the distances from the start node to each node)

About A* implementation, it is done in the following way: each time a node is expanded $f(n)$ is calculated for all the nodes in the frontier and the one with the smallest $f(n)$ is the one that is selected for the next expansion, this is done recursively and if at one point the $f(n)$ of all the nodes available is greater than a previous $f(n)$ of a node, it is necessary to go back to that previous node and continue the search through there, and again, this process is repeated until a solution is found or there is no more memory available to store the nodes.

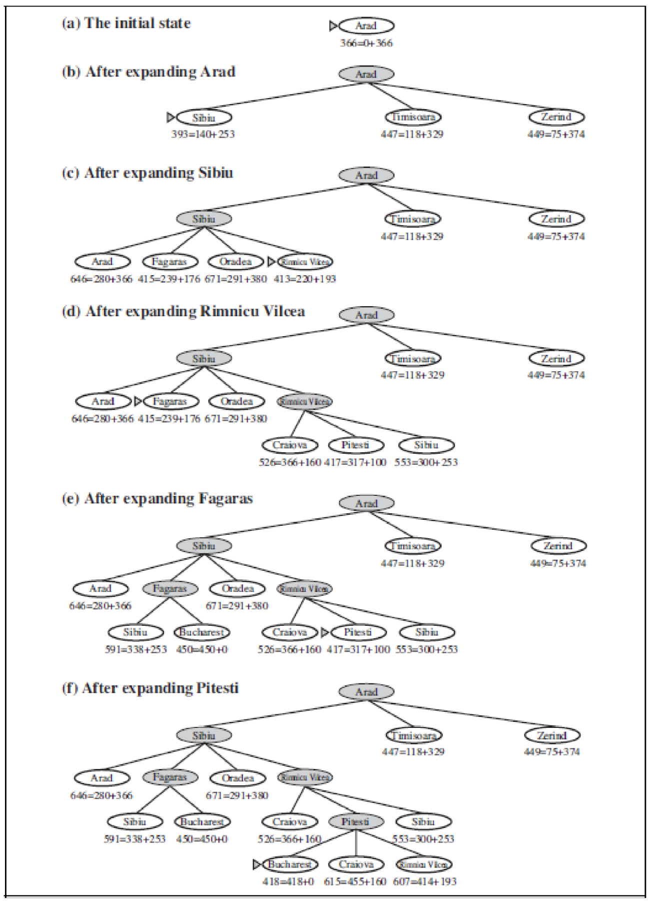

Figure 2: A* Search Diagram

1.3 SMA* Search

Simplified Memory-bounded A* (SMA*) is based on A* with the main advantage that uses bounded memory instead of exponential memory. Besides that, the rest of the characteristics are inherited from A*.

SMA* works in the same way as A*, selecting the node of expansion based on the evaluation function $f(n) = g(n) + h(n)$ until memory is full. When memory is full, SMA* needs to remove an old node in order to add a new one to the search tree. The node that is selected to be removed is the worst node, the one with the highest $f(n)$. In addition, SMA* needs to remember the $f(n)$ of the best forgotten child with the parent node so when all the explored paths seem to be worse than the forgotten one, the path is re-generated.

An important scenario that we need to consider about SMA* is what would happen if all the leaf nodes have the same $f(n)$ value, as that situation can cause the selection of the same node for deletion and expansion, which is a big problem. Therefore, to solve that issue, SMA* selects the newest best node for expansion and the oldest worst node for deletion.

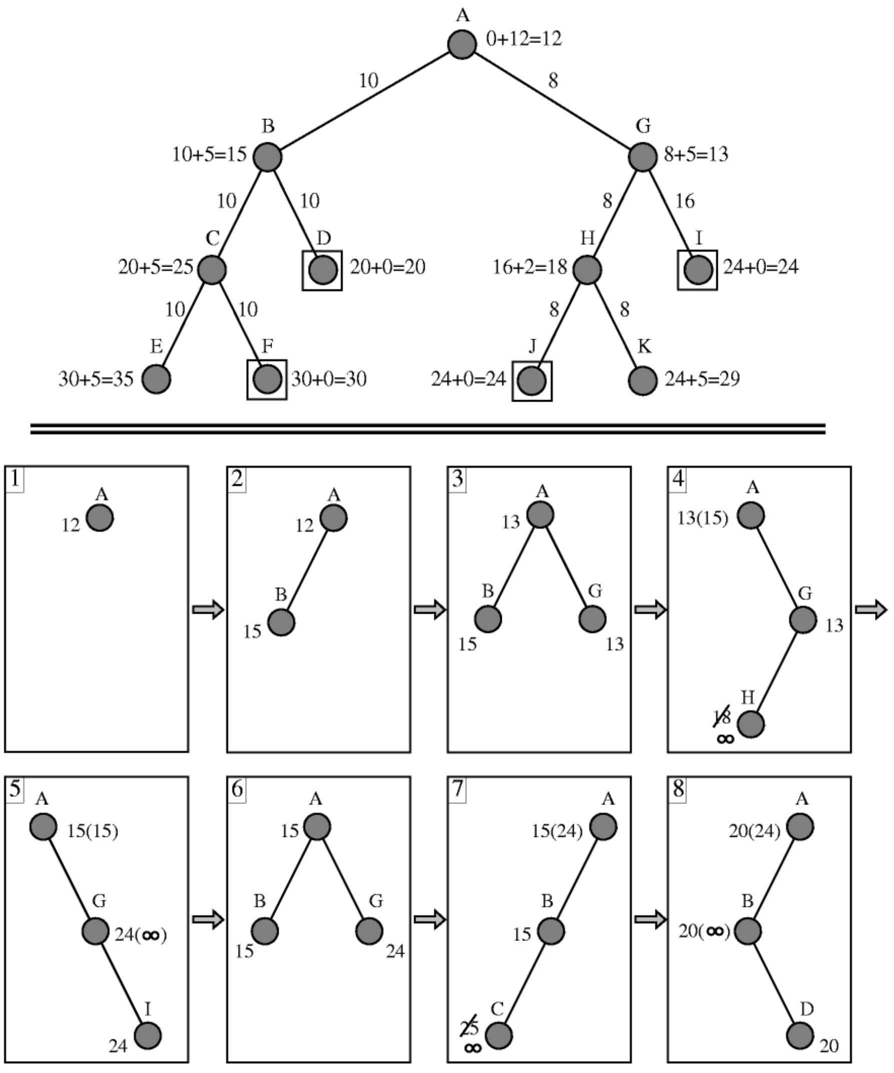

Figure 3: SMA* Search Diagram

2 Search Strategies Evaluation

2.1 Bidirectional Search

In general the Bidirectional search strategy provides completeness and optimality if Breadth-First search is implemented at both searchs, the time complexity is $b^{d/2}$ and space complexity is $b^{d/2}$ as well. Therefore we can see that the space complexity is the biggest issue that we can have.

When it comes to the caves problem, the main issue that we can have with Bidirectional search is that there are routes which only go in one direction, for the forward search this might not be a big issue but when it comes to the backward search things can get a bit more complex.

2.2 A* Search

In general A* search provides completeness and optimality. For completeness it is necessary that the number of nodes with a path cost less or equal to C^* (path cost of optimal solution) is finite. While for optimality, in the tree version, $h(n)$ needs to be an admissible heuristic (does not overestimates the path cost to get to the goal node). When it comes to time complexity, it is exponential to the path length and for the space complexity, all nodes are stored in memory, which makes space the main issue of this approach.

When it comes to the caves problem, on the one hand the number of caves is going to be finite so we have completeness and in the other hand, a heuristic that can be used is h_{SLD} (Straight Line Distance) and as a straight line is the shortest path between two points, the path cost can not be overestimated. About the space complexity, there might be memory issues if the number of caves is too big.

2.3 SMA* Search

In general SMA* is complete if there is any reachable solution (the depth of the deepest node is less than the memory size), and it is optimal if any optimal solution is reachable. The only issue that SMA* might have is that with very complex problems it can be forced to switch all the time between many possible solution paths while only a small set of them can fit in memory, and this will affect the time.

When it comes to the caves problem, we can come to the same evaluation as for A* with the difference that now if the number of caves gets too big the memory is not going to be a big issue.

References

[1] Stuart Russell, Peter Norvig. *Artificial Intelligence A Modern Approach Third Edition.* Chapter 3.4.6, Pages 90-91. Chapter 3.5.2, Pages 93-99. Chapter 3.5.3, Pages 101-102.

[2] Informed Search Methods: http://www.massey.ac.nz/~mjjohnso/notes/59302/l04.html

[3] A* Search Algorithm: https://en.wikipedia.org/wiki/A*_search_algorithm