Artificial Intelligence for Robotics
- Homework 6 -

Kiran Vasudev, Patrick Nagel

Due date: 16.05.2016

Contents

1 Search Strategies

1.1 Uniform Cost Search

The uniform cost search is an **optimal** algorithm with any step-cost function. It stores and orders its nodes in a queue related to their path-costs. The node with the lowest path cost will be expanded. The algorithm then tests the node, which is selected, if it is the goal. Not to apply the goal test before the node with the lowest path-cost is selected avoids to choose a suboptimal solution. It makes sure that all nodes are generated and considered first. The uniform cost search also provides a test for the case that a better path to the goal is found. The search focuses on the total costs of the nodes, which leads to the **possibility of infinite loops** if zero cost paths are given.

The complexity of this algorithm does not depend on the depth of the search. Instead it is characterized by the path costs. In the worst case of the complexity is $O(b^{1+\frac{C*}{\epsilon}})$ **(worst case)**. The C* represents the costs of an optimal solution and ε the costs which are guaranteed at every node (can be also higher than that). In the case (that all costs are equal the complexity is $O(b^{d+1})$ **(best case)**. The algorithm is then quite similar to breadth-first search just with the exception that the uniform cost search does not stop after finding the goal, but after checking all nodes (in case a lower cost solution exists). In this case the algorithm expands nodes unnecessarily since all costs are the same and the optimal path was alread found.

1.2 Depth-First Search

The Depth First Search Algorithm is an algorithm that is similar to Breadth First Search. In Breadth-First Search, the algorithm uses a queue which follows the principle of First In First Out (**FIFO**). In Depth First Search, we use the data structure of a stack that follows the principle of Last In First Out(**LIFO**).

The algorithm starts at the root node and then traverses to the child at the left and goes right down to the deepest node in the state space or until goal state is found. If the goal is found, the goal is returned, otherwise the algorithm backtracks and visits the unvisited nodes.

Depth First Search is optimal when we use a state space or a graph and its size is finite. If the size is infinite, then the algorithm is not optimal as it may end in infinite loops or it may return a path that is not the optimal solution.

The biggest advantage of Depth First Search is the space complexity. The Algorithm works in two ways. First, the visited elements are pushed onto the stack and once the maximum depth of that node is reached, the elements are popped. This method is called **backtracking**.

Complexities:

- Space Complexity:

$$O(bm)$$

- Time Complexity:

$$O(b^m)$$

where b is the branching factor and m is the max depth of the state space.

1.3 Limited Depth-First Search

The limited depth-first search is an extension of the depht-first search. It adds a depth limit to **avoid failure in infinite state spaces**. In the limit depth the children nodes are not longer generated, because it is the end of the search.

The limited depth-first search contains a few scenarios, which can lead to different results and characteristics: Setting up a limit can result in not finding the goal at all. This case occurs if the goal node is beyond the limit. If the limit is selected higher than the actually depth, the search is also nonoptimal. In cases of having more information of the states, the limit can be chosen more efficient. Such a limit is called diameter.

In general is the time complexity of the algorithm $O(b^l)$ and the space complexity $O(b{\cdot}l)$ (l = limit). In case that the limit is infite, there is no difference to the depth-first search any longer.

1.4 Iterative-deepening Search

The Algorithm **Iterative Deepening Search** is used when the search space is very large and the depth is unknown. IDDFS is a combination of both BFS and DFS. This algorithm uses a counter that is used to iterate through every level of the state space. This counter helps remove traversing the maximum depth of the state space like DFS and uses BFS to traverse every level. If the goal state is not found in a particular level, then the counter

is incremented and the algorithm follows. In every iteration, the algorithm traverses the entire state space starting from the root node. These states are generated multiple times but this generation isn't too costly.

Complexities:

- Space Complexity:
$$O(bd)$$
- Time Complexity:
$$O(b^d)$$

where b is the branching factor and d is the depth of least cost solution.

1.5 Informed Search

The expression informed search is the generic term for search algorithms which use problem-specific knowledge to find solutions more efficient. This types of searches are also called heurisitic searches.

Informed searches use their knowledge to build an evaluation function. The function evaluates if it is worth to expand the node based on the costs, which have to be paid to reach the node and at the end the goal. The approach of choosing the most suitable way first is called best-first search. The evaluation function of lots of algorithms, which fit into the definition of a best-first search, contain a part which is called heuristic function. This function itself gives the estimated cost of the cheapest path from a node n to the goal node.

1.6 Greedy Search

A Greedy Search algorithm is a search algorithm that uses a heuristic that helps the algorithm make an optimal choice at every iteration/stage of the algorithm. This type of algorithm does not produce an optimal solution for all problems, but they may provide optimal solutions over a reasonable amount of time. A greedy algorithm makes its decision based on the most optimal choice at that very instance but does not look ahead to see if we reach the desired goal in its best path.

1.7 A*

The A* search is the most famous search using the best-first approach. It was found 1968 by Peter Hart, Nils J. Nilsson and Bertram Raphael and

contains as many other **best-first** search a heuristic function ($h(n)$). Together with a function, which gives the costs to reach a node ($g(n)$), it builds the evaluation function of the A* search: $f(n) = g(n) + h(n)$. In words the evaluation function $f(n)$ means the estimated cost of the chepeast solution through n.

The algorithm finds always the best solution if such a solution is given at all. Thus it is **optimal**. Furthermore the search is **complete** unless there are infinity nodes with $f(n) <= f(G)$. The complexity of the search is in the best case $O(d)$ and in worst case it is O of the entire state space. The A* search can be compared to the uniformed-cost-search but instead of having only the g-function, it is g + h.

1.8 Iterative A*

Iterative A* algorithm is a search algorithm that helps to find the shortest distance between a one node to another in a state space. This algorithm uses Depth First Search and is a variant of the Iterative Depth First Search Algorithm, the only difference being that the iterative A* algorithm uses a heuristic function to evaluate the cost to get to the goal using the normal A* algorithm. As it uses the concepts of DFS, the memory requirements for this are very low, and also the algorithm does not traverse the entire depth of the tree, rather it traverses the depth where the cost to the goal is the cheapest.

2 What is a heuristic function?

A heuristic function is one that estimates the cost of the path from a node to the closest goal state. If the node is the goal state, the cost is 0. The heuristic function is denoted by **h(n)**.

3 Comparison of the two heuristics. (Task 3 (b))

The misplaced tiles method takes into account only the fact that the tile is misplaced and not the distance of this misplaced tile to the desired goal. Therefore, a tile closer to the goal is treated very similarly to a tile that is very far from the goal.

On the other hand, the Manhattan Distance method takes into consideration how far a misplaced tile is from its desired goal. This may lead to better outcomes.

4 Comment if the heuristics are consistent or inconsistent. (Task 3 (c))

Heuristics are inconsistent if the heuristic chosen overshoots the estimated cost to the desired goal.

Similarly, heuristics are consistent if the heuristic chosen is at most the estimated cost between a node and the goal state.