Artificial Intelligence

Informed Search

Paper Code: CS-401

Anupam Pattanayak[1]

Assistant Professor,

Department of Computer Science,

Raja N. L. Khan Women's College (Autonomous),

Midnapore, West Bengal

April 14, 2020

[1]anupam.pk@gmail.com

Contents

1

Informed Search

In classes we have seen *uninformed search* and A^* search algorithm as an *informed search* technique. In class, we have seen an example demonstrating how A^* works. In this study material, we will discuss remaining portion of informed search.

This study material has been prepared by consulting multiple books and video lectures. These references include the book by Russel[1], book by Coppin[2], and NPTEL Course lectures on *An Introduction to Artificial Intelligence* by Prof. Mausam[3] in SWAYAM platform of MHRD, Govt. of India.

1.1 Background

Let us briefly revisit whatever have been covered so far. Many computational problems can be mapped as AI search problems. In AI search problem, there are huge number of states and actions. There is a *start state* and one or more *goal state(s)*. Our target is to find a path from start state to goal state. The difference between traditional algorithmic search and AI search is that, the complete search space is not given to us in AI. This is because the search space can be really so huge that representing those states in memory will not be feasible. So, in AI, the complete search state space is provided in terms of *expansion function*.

The algorithms that we have discussed as uninformed search techniques - Depth First Search (DFS), Breadth First Search (BFS), and Iterative Deepening Search (IDS) are somewhat *good* but they are not satisfactory. *These algorithms search blindly at all directions*. We require a guidance in search-

[1]Artificial Intelligence A Modern Approach by Russel and Norvig, PHI.
[2]Artificial Intelligence Illuminated by Ben Coppin, Jones and Bartlett Publishers
[3]https://swayam.gov.in/nd1_noc20_cs42/preview

ing that helps us to reach good state more efficiently. This guided search is called *informed search* or *heuristic search*.

1.2 Informed Search

Here, some kind of intuition is applied that guides search to reach the goal. It makes a guess in every step. This guess may not help always to reach the goal. There can be obstacles in the path to goal. In that case, we need to replan the search. So, the idea of *informed search (or heuristic search)* is to be *smart* regarding search paths to try. Here, we want to capture the notion that, *intuitively, this node is closer to a goal state than remaining nodes considering the present state*. We have to specify this formally. A node is chosen for next expansion on the basis of an *evaluation function* that estimates cost to reach a goal state. In the following, we describe the algorithm 1 depicting *general tree search paradigm of informed search*

Algorithm 1 General Tree Search Paradigm of Innformed Search

function *TreeSearch*(RootNode)
Input: Search Tree
Output: solution on success, failure otherwise

```
1.  frontier ← Successors(RootNode)
2.  while(notempty(frontier))
3.    node ← RemoveFirst(frontier)    /* smallest f value */
4.    state ← state(node)
5.    if(GoalTest(state))
        return solution(node)
6.    frontier ← InsertAll(Successors(RootNode))
7.  return failure
  end TreeSearch
```

Similarly, the algorithm of general graph search paradigm of informed search can be given. But we are skipping that. If you are interested, refer

the book by Russel and Norvig[4].

1.2.1 Evaluation Function and Heuristic Functions

We use three functions: *f(n)*, *g(n)*, and *h(n)*.
Evaluation function, $f(n) = g(n) + h(n)$ where n is the current node.
Purpose of these functions are as follows:
$g(n)$ = cost incurred so far to reach n,
$h(n)$ = estimated cost from n to goal,
$f(n)$ = estimated total cost of path through n to goal.
$f()$ is the *evaluation function* and $h()$ = is the *heuristic function*.

1.2.2 Admissible Heuristic and Constituent Heuristic

A heuristic function $h(n)$ is said to be *admissible* if $h(n) \leq h^*(n)\ \forall n$ where $h^*(n)$ is the *true cost* to reach goal state from n. An admissible heuristic function never over-estimates the cost of reaching goal. That is, admissible function is *optimistic*. There is a theorem that says the following.

If $h(n)$ is admissible, then A^* using Tree-Search is *optimal*.

A relevant question you may ask is, *is heuristic function always need to be less than thee optimal*? The answer is *no*. It depends on whether we are minimizing or maximizing. If we want minimum path cost then it has to be $\leq$. Otherwise, if we are interested in maximizing profit, then it has to be $\geq$.

Let us now see what is meant by *consistent* heuristic. A heuristic function $h(n)$ is said to be *consistent* if $h(n) \leq c(n, a, n') + h(n')\ \forall n$ and $\forall$ successor n' due to legal action a.

There is a theorem that says the following.

If $h(n)$ is consistent, then A^* using Graph-Search is *optimal*. The *consistent* condition is more stricter than *admissible*. Every consistent heuristic is also admissible, but not the vice versa.

[4]Artificial Intelligence A Modern Approach by Russel and Norvig, PHI

1.3 Properties of A^* Search

Properties of A^* search algorithm is given in the tabular format in table 1.1.

Table 1.1: Properties of A^* Search

Property	Description
Complete	Yes, unless there exists infinitely many nodes with $f \leq f(G)$
Optimal	Yes, depending upon goodness of heuristic property
Time	Exponential. In worst case, all nodes are visited
Space	Keeps all generated nodes in memory

A^* is optimally efficient for any given consistent heuristic. In other words, no other optimal search algorithm will expand fewer nodes than A^*. Complexity of A^* often makes it impractical for adoption in many large scale applications.

1.4 Iterative Deepening A^* (IDA^*) Search

It tries to overcome the disadvantage of A^* algorithm. A^* algorithm expands all nodes with lower f value before it takes up higher f value. IDA^* follows the idea similar to ID search: exhaust all nodes upto a certain bound before expanding to next level of bound. This bound can be cost, cost made so far, depth etc. Overview of the IDA^* algorithm is given in algorithm 2.

Algorithm 2 IDA^* algorithm overview

```
while (solution not found)
  do DFS but prune when cost(f) > current bound
   increase bound
```

It says that just do DFS as long as our f increases upto a certian bound. When it reaches the specified bound then stop and backtrack. Then increase the bound and continue.

IDA^* is not *systematic*. But, IDA^* never expands to a node where cost $>$ optimal cost. For real problems, often IDA^* search algorithm is used instead of A^* search algorithm.

1.5 Depth First Branch & Bound Search

As the name sugggests, it also follows DFS. It uses two mechanisms:

I. BRANCH: This is the mechanism for branching when searching in the solution space. It suggests which of the children is better at present and go there.

II. BOUND: This is the mechanism to generate a bound so that many branches can be terminated. A branch is terminated if it exceeds a certain bound.

Now, let us look at an example. Consider the following weighted graph in figure 1.1. The weights represent cost of edges.

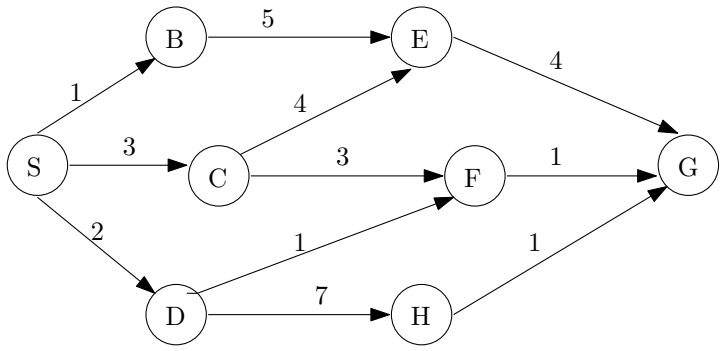

Figure 1.1: A Graph

The search tree corresponding to above graph is shown in figure 1.2.

Let us now explore this search tree by DFS using branching strategy. Starting node is S, and goal state is G. Initially, we do not know cost of the solution. We can assume our cost of solution is ∞. So, initially we set *upper bound* $= \infty$.

Then using DFS, we will traverse to node B from S, as $S - B$ cost is smaller than both $S - C$ and $S - D$. That is, node $S - B$ is visited prior to node C and node D from S. Similarly, from node B there is only one edge to E, and so is from E - node G. SO obtain a path to goal $S - B - E - G$. Cost of this path is $1 + 5 + 4 = 10$. As soon as we find a path to goal, we

A

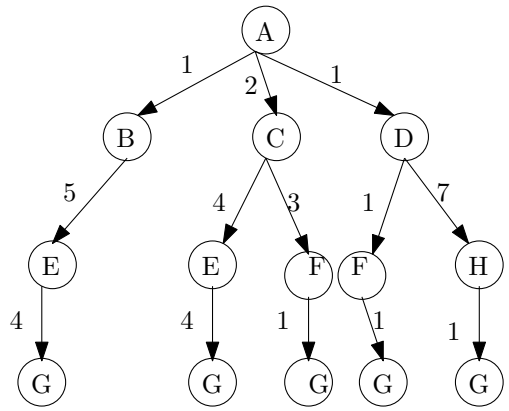

Figure 1.2: Search Tree

update upper bound to be 10 from ∞. THis may be optimal, but we do not know that yet. We now backtrack to E, but there is no other path from E. So, we again backtrack to B. From B also, we do not have no other path. So, we again backtrack to node S. From S, next node selected is D, and the path to goal state reached is $S - D - F - G$, which has cost $2 + 1 + 1 = 4$. This is smaller than present upper bound. So, we update upper bound to 4. Next we backtrack to node D and see the alternate path so far $S - D - H$ which already incurs more cost than upper bound, so we discard this path. Similarly we will discard the path $S - C - F$ whose cost is $6 >$ upper bound, and so also the path $S - C - E$ with cost $7 >$ upper bound. So, the optimal path to goal state is $S - D - F - G$.

Depth first branch and bound is systematic, but it expands to sub-optimal nodes also. If the given search graph has infinte depth then IDA^* is better than DFS branch and bound technique. Otherwise, if we have better estimate of upper bound then DFS Branch & Bound works better. Also, if there are many paths to goal state then DFS Branch & Bound does very well.