Artificial Intelligence

Heuristic Search

Andrea Torsello
Search in AI

Blind search strategies

- Breadth First
- Depth First
- Iterative Deepening

Analyzed

- Completeness
- Optimality
- Time Complexity
- Space Complexity

Optimality evaluated only in terms of number of steps
Uniform Cost Search

Uniform Cost is a **blind search** algorithm that is **optimal** according to any specified **path length** function.

Can be obtained from the generic search stab function by selecting from $L$ the node with the *minimal path length* from the root.

- Can be implemented with a heap or any efficient priority queue implementation.

Let $L$ be a list of **visited** but not **expanded** nodes:

1) Initialize $L$ with the initial state.
2) If $L$ is empty, **FAIL**, else extract from $L$ node $n$ with minimum path length from the root.
3) If $n$ is a goal node, **SUCCEED** and return the path from the initial state to $n$.
4) Remove $n$ from $L$ and *insert* all the children of $n$.
5) Goto 2.)
Time/Space complexity

Uniform cost is guided by path length rather than depth, so its time complexity cannot easily be characterized in terms of $b$ and $d$

Rather, let $C^*$ be the path length of the optimal solution and $\varepsilon$ the minimum cost of any action.

Then the worst-case time and space complexity is

$$O(b^{1+[C^*/\varepsilon]})$$

Which can be much greater than $b^d$

- Uniform cost can explore large trees of small steps before exploring paths with large and potentially useful steps.

When all steps costs are equal, uniform cost is similar to breadth first.
Heuristics

The assumption behind **blind search** is that we have no way of telling whether a particular search direction is likely to lead us to the goal or not.

The key idea behind **informed** or **heuristic search** algorithms is to exploit a task-specific measure of goodness to try to either reach the goal more quickly or find a more desirable goal state.

Heuristic: From the Greek for “find”, “discover”.

Heuristics are criteria, methods, or principles for deciding which among several alternative courses of action promises to be the most effective in order to achieve some goal”.

Judea Pearl “Heuristics” 1984

Problem knowledge comes in the form of a **heuristic function** $h'(n)$

- $h'(n)$ assigns to each node $n$ an estimate of the optimal path length from $n$ to a goal node.
- We assume that $h'$ is non-negative and that $h'(n)=0$ iff $n$ is a goal node.
Heuristics for the game of 8

$h'_1(n)$ = number of misplaced tiles

$h'_2(n)$ = sum of Manhattan distance* of each tile

Example:

$h'_1(n) = 7$

$h'_2(n) = 4 + 0 + 2 + 3 + 1 + 3 + 1 + 3 = 17$

* Given two planar points of coordinate $P_1 = (x_1, y_1)$ and $P_2 = (x_2, y_2)$, the Manhattan distance is defined as

$$D(P_1, P_2) = |x_2 - x_1| + |y_2 - y_1|$$
$h'(n) = \text{straight-line distance between } n \text{ and goal node}$
Best First (Greedy) Search

The idea behind best first search is to always explore the most promising path first.

Can be obtained from the generic search stab function by selecting from \( L \) the node with the minimal value of the expected distance to goal (value of heuristic function).

Let \( L \) be a list of visited but not expanded nodes

1) Initialize \( L \) with the initial state
2) If \( L \) is empty, FAIL, else extract from \( L \) node \( n \) with minimum value of \( h'(n) \)
3) If \( n \) is a goal node, SUCCEED and return the path from the initial state to \( n \)
4) Remove \( n \) from \( L \) and insert all the children of \( n \)
5) Goto 2)
Best First Search Example
Best First Search Example
Best First Search Example

Diagram:
- **Arad**
  - **Sibiu**
    - **Arad** 366
    - **Fagaras** 176
    - **Oradea** 380
    - **Himnicu Vilcea** 193
  - **Timisoara** 320
  - **Zarind** 374
Best First Search Example
Analysis of Best First

Complete? No – can get stuck in loops, e.g., Iasi -> Neamt -> Iasi -> Neamt

Time? $O(bm)$, but a good heuristic can give dramatic improvement

Space? $O(bm)$ -- keeps all nodes in memory

Optimal? No

Example of non optimality

- The number in parentheses represent the order of expansion
- The other the value of the heuristic function $h'$
**A***

A* algorithm mixes the optimality of uniform cost with the heuristic search of best first

A* realizes a best first search with evaluation function

\[ f(n) = g(n) + h'(n) \]

with

- **g(n)** is the path length from the root to **n**
- **h'(n)** is the heuristic prediction of the cost from **n** to the goal

Let **L** be a list of **visited** but not **expanded** nodes

1) Initialize **L** with the initial state

2) If **L** is empty, **FAIL**, else **extract** from **L** node **n** with minimum value of **f(n)**=\(g(n)+h'(n)\)

3) If **n** is a goal node, **SUCCEED** and return the path from the initial state to **n**

4) Remove **n** from **L** and **insert** all the children of **n**

5) Goto 2)
A* search example

366 = 0 + 366
A* search example

- Sibiu: 393 = 140 + 253
- Timisoara: 447 = 118 + 329
- Zerind: 449 = 75 + 374
A* search example
A* search example
A* search example

A* algorithm is a search algorithm that finds the path from a given start node to a given goal node in a graph. It uses a combination of the cost of reaching the node from the start and an estimated cost to reach the goal. This diagram shows the A* search example with the costs and estimated costs for each node.
A* search example
Consistent heuristics

A heuristic is consistent if for every node $n$, every successor $n'$ of $n$ generated by any action $a$, with cost $c(n,a,n')$

$$h'(n) \leq c(n,a,n') + h'(n')$$

If $h'$ is consistent, we have

$$f(n') = g(n') + h'(n')$$
$$= g(n) + c(n,a,n') + h'(n')$$
$$\geq g(n) + h'(n)$$
$$= f(n)$$

i.e., $f(n)$ is non-decreasing along any path

Corollary: $h'(n) \leq h(n)$, where $h(n)$ is the real (unknown) distance from the goal
Optimality of A*

**Theorem:** If function $h'$ is consistent, then A* is optimal.

Proof: Let $s$ be an optimal goal node and $x$ a non-optimal goal. Let $n_0 n_1 \ldots n_k = s$ be the path from the root to $s$. For all $i=0 \ldots k$, we have $f(n_i) < f(x)$.

In fact:

$$f(n_i) = g(n_i) + h'(n_i) \leq g(n_i) + h(n_i) = g(s)$$

$$< g(x) = g(x) + h'(x) = f(x)$$

But then $s$ must be expanded before $x$. 
Optimality of A*

A* expands nodes in order of increasing f value

Gradually adds "f-contours" of nodes

Contour \( i \) has all nodes with \( f = f_i \), where \( f_i < f_{i+1} \)
<table>
<thead>
<tr>
<th>Properties</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Complete?</td>
<td>Yes (unless there are infinitely many nodes with ( f \leq f(G) ))</td>
</tr>
<tr>
<td>Time?</td>
<td>Exponential</td>
</tr>
<tr>
<td>Space?</td>
<td>Exponential (Keeps all nodes in memory)</td>
</tr>
<tr>
<td>Optimal?</td>
<td>Yes</td>
</tr>
</tbody>
</table>

A* is an Heuristic counterpart of breadth first. We can improve the space requirements in a way similar to iterative deepening.
Iterative Deepening A* (IDA*)

Let L be the list of visited but not expanded node, and $C$ the maximum depth

1) Let $C=0$

2) Initialize $L$ to the initial state (only)

3) If $L$ is empty increase $C$ and goto 2), else extract a node $n$ from the front of $L$

4) If $n$ is a goal node, SUCCEED and return the path from the initial state to $n$

5) Remove $n$ from $L$. If the level is smaller than $C$, insert at the front of $L$ all the children $n'$ of $n$ with $f(n') \leq C$

6) Goto 3)
IDA* Example

(1, 2, 4, 9) 0+2=2

(3, 5, 10) 1+1=2

(6, 11) 2+1=3

(12) 3+1=4

4+1=5

5+0=5

(7, 13) 1+2=3

(8, 14) 2+1=3

(15) 3+1=4

(16) 4+0=4