**Lecture 4: Heuristic Search**

**Introduction**

* AI problem solvers employ heuristics in two situations:
	1. A problem may not have an exact solution because of the ambiguities in the problem statement or available data.
	2. A problem may have an exact solution, but the computational cost is high.
* A heuristic is an informed estimate of the next step to be taken in solving a problem.
* A heuristic is based on the experience and intuition. In other words, it is based on the knowledge of the present and current situation.
* It sometimes leads to suboptimal solution or it fails to find a solution at all.

**Examples:** Consider the game of tic-tac-toe.

1. The complexity of this game is 9! = 9 ∗ 8 ∗ 7 ∗ 6 ∗ 5 ∗ 4 ∗ 3 ∗ 2 ∗ 1
2. By using symmetry reduction heuristic, many configurations are actually equivalent.
3. By applying a heuristic function which is Move to the board where X has most win line. (if there are two or more boards have the same number of win lines chose the first one found)





Figure 1: The “most wins” heuristic applied to the first children in *tic*−*tac*−*toe*.

4

Figure 2: Heuristically reduced state space for *tic*−*tac*−*toe*.

1. **The Best first Search Algorithm**

 The goal of the best-first search algorithm is to find the goal state by looking at as few states as possible. The more informed heuristic, the fewer states are processed in finding the goal

* + This algorithm expands current node in the search space and evaluates heuristically its children.
	+ The best child is selected for further expansion, neither its sibling nor its parents are retained.
	+ The search halts, when we reach state which has better heuristic than any of its children.
	+ This strategy can mainly fail because of two main factors:-
		1. It does not keep history, therefore it may go on an infinite loop.
		2. It can stuck at local maxima, when it reaches state with better evaluation than all of its children. Then the algorithm halts, if such state is local maximum.

1. **Implement Heuristic Evaluation Function**

 Here we want to evaluate performance of several heuristics to solve the 8-puzzle game.



1. Count the tiles out of place: In each state we compare the position of all tiles with respect to the goal board (State with fewest tiles out of place is closer to the desired goal).
2. Distance Summation: Sum all the distance by which the tiles are out of +place (State with the shortest distance is closer to the desired goal).
3. Count reversal Tiles: If two tiles are next to each other, and the goal requires their position to be swapped. The heuristic takes this into account by evaluating the expression (2 ∗ *number of direct tiles reversal*)



* Heuristics might be wrong:

The search could continue down a wrong path

* Solution:

Maintain depth count, i.e., give preference to shorter paths

Modified evaluation function f:

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

* f(n) estimate of total cost along path through n
* g(n) actual cost of path from start to node n
* h(n) estimate of cost to reach goal from node n

Example:-



* Best first search: *ABDEF* with cost = 13 = (2 + 4 + 3 + 4)





Games in general are ideal fields to explore and design behavior of heuristic because of many reasons:-

1. Search space is very large, therefore, we need heuristic techniques.
2. Games are complex, so It has many heuristic evaluations.
3. Games do not involve with a complex representation, so It does not focus on knowledge representation problem.
4. We can easily apply heuristic techniques because all nodes have a common representation.
5. **Hill-Climbing**

 Hill-climbing is similar to the best first search. while the best first search considers states globally, hill-climbing considers only local states. The application of the hill-climbing algorithm to a tree that has been generated prior to the search is illustrated in the Figure above:



**Example 1:** Suppose that each of the letters represent a state in a state space representation. Legal moves:

A to B3 and C2

B3 to D2 and E3

C2 to F2 and G4

D2 to H1 and I99

F2 to J99

G4 to K99 and L3

Start state: A when Goal states: H, L

The result of applying the hill-climbing algorithm to this problem is illustrated in the table below.

|  |  |
| --- | --- |
| **OPEN** | **CLOSED** |
| [A] | - |
| [C2, B3] | [A] |
| [F2, G4, B3] | [A, C2] |
| [J99, G4, B3] | [A, C2, F2] |
| [G4, B3] | [A, C2, F2, J99] |
| [L3,K99, B3] | [A, C2, F2,J99, G4] |

**4. A\* Algorithm**

A\* Algorithm extends the path that minimizes the following function-

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

Here,

* ‘n’ is the last node on the path
* g(n) is the cost of the path from start node to node ‘n’
* h(n) is a heuristic function that estimates cost of the cheapest path from node ‘n’ to the goal node

**Algorithm-**

* The implementation of A\* Algorithm involves maintaining two lists- OPEN and CLOSED.
* OPEN contains those nodes that have been evaluated by the heuristic function but have not been expanded into successors yet.
* CLOSED contains those nodes that have already been visited.

The algorithm is as follows-

**Step-01:**

* Define a list OPEN.
* Initially, OPEN consists solely of a single node, the start node S.

**Step-02:**

If the list is empty, return failure and exit.

**Step-03:**

* Remove node n with the smallest value of f(n) from OPEN and move it to list CLOSED.
* If node n is a goal state, return success and exit.

**Step-04:**

Expand node n.

**Step-05:**

* If any successor to n is the goal node, return success and the solution by tracing the path from goal node to S.
* Otherwise, go to Step-06.

**Step-06:**

For each successor node,

* Apply the evaluation function f to the node.
* If the node has not been in either list, add it to OPEN.

**Step-07:**

Go back to Step-02

**Example:**

Consider the following graph-



The numbers written on edges represent the distance between the nodes.

The numbers written on nodes represent the heuristic value.

Find the most cost-effective path to reach from start state A to final state J using A\* Algorithm.

**Solution-**

 **A** is the start state and **J** is the Goal state**.**

|  |  |
| --- | --- |
| open | Close |
| [ A ] | [ ] |
| [ F9 B12 ] | [ A ] |
| [ G9 H13 B14 ] | [ A F9 ] |
| [ I8 H13 B14 ] | [ A F9 G9 ] |
| [ **J10** E15 H13 B14 ] | [ A F9 G9 I8] |

**Step-01:**

 We start with node A.

* Node B and Node F can be reached from node A.

A\* Algorithm calculates f(B) and f(F).

* f(B) = 6 + 8 = 14
* f(F) = 3 + 6 = 9

 Since f(F) < f(B), so it decides to go to node F.

 **Path- A → F**

**Step-02:**

Node G and Node H can be reached from node F.

A\* Algorithm calculates f(G) and f(H).

* f(G) = (3+1) + 5 = 9
* f(H) = (3+7) + 3 = 13

Since f(G) < f(H), so it decides to go to node G.

**Path- A → F → G**

 **Step-03:**

 Node I can be reached from node G.

A\* Algorithm calculates f(I).

f(I) = (3+1+3) + 1 = 8

It decides to go to node I.

**Path- A → F → G → I**

**Step-04:**

Node E, Node H and Node J can be reached from node I.

A\* Algorithm calculates f(E), f(H) and f(J).

* f(E) = (3+1+3+5) + 3 = 15
* f(H) = (3+1+3+2) + 3 = 12
* f(J) = (3+1+3+3) + 0 = 10

 Since f(J) is least, so it decides to go to node J.

 **Path- A → F → G → I → J**

This is the required shortest path from node A to node J.



Example:



Start State is A and F is the Goal.

|  |  |
| --- | --- |
| Open | Close |
| [ A ] | [ ] |
| [ B10 C11] | [ A] |
| [ C11 D12 ] | [ A B] |
| [ G8 D12 ] | [ A B C ] |
| [ F7 D12 ] | [ A B C G ] |

**Important Note-**

 It is important to note that-

* A\* Algorithm is one of the best path finding algorithms.
* But it does not produce the shortest path always.
* This is because it heavily depends on heuristics.