Artificial Intelligence

Informed Search

- Incorporate additional measure of a potential of a specific state to reach the goal.
- A potential of a state to reach a goal is measured through a heuristic function \( h(n) \), thus always \( h(goal) = 0 \).
- An evaluation function is denoted \( f(n) \).

Properties of Heuristic functions

- The 2 most important properties:
  - relatively cheap to compute
  - relatively accurate estimator of the cost to reach a goal. Usually a "good" heuristic is if \( \frac{1}{2} \text{opt}(n) < h(n) < \text{opt}(n) \)
- Examples:
  - Navigating in a network of roads from one location to another. **Heuristic function**: Airline distance.
  - Sliding-tile puzzles. **Heuristic function**: Manhattan distance - number of horizontal and vertical grid units each tile is displaced from its goal position

Best First Search Algorithms

- Principle: Expand node \( n \) with the best evaluation function value \( f(n) \).
- Implement via a priority queue
- Algorithms differ with definition of \( f \):
  - Greedy Search: \( f(n) = h(n) \)
  - A*:
    - \( f(n) = g(n) + h(n) \)
  - IDA*: iterative deepening version of A*
  - Etc'.
Best-FS Algorithm Pseudo code

1. Start with $open = \text{[initial-state]}$.
2. While $open$ is not empty do
   1. Pick the best node on $open$.
   2. If it is the goal node then return with success. Otherwise find its successors.
   3. Assign the successor nodes a score using the evaluation function and add the scored nodes to $open$.

General Framework using Closed-list (Graph-Search)

```plaintext
GraphSearch(Graph graph, Node start, Vector goals)
1. $O \leftarrow \text{make_data_structure}(\text{start})$ // open list
2. $C \leftarrow \text{make_hash_table}$ // closed list
3. While $O$ not empty loop
   1. $n \leftarrow O.\text{remove_front}()$
   2. If goal ($n$) return $n$
   3. If $n$ is found on $C$ continue
   4. Otherwise
   5. $O \leftarrow \text{successors} (n)$
   6. $C \leftarrow n$
4. Return null //no goal found
```

Greedy Search Attributes

- Completeness: No. Inaccurate heuristics can cause loops (unless using a closed list), or entering an infinite path.
- Optimality: No. Inaccurate heuristics can lead to a non-optimal solution.
- Time & Memory complexity: $O(b^m)$.

A* Algorithm (1)

- Combines greedy $h(n)$ and uniform cost $g(n)$ approaches.
- Evaluation function: $f(n) = g(n) + h(n)$.
- Completeness:
  - In a finite graph: Yes
  - In an infinite graph: if all edge costs are finite and have a minimum positive value, and all heuristic values are finite and non-negative.
- Optimality:
  - In tree-search: if $h(n)$ is admissible
  - In graph-search: if it also consistent
Heuristic Function $h(n)$

- **Admissible/Underestimate:** $h(n)$ never overestimate the actual cost from $n$ to goal.

- **Consistent/monotonic** (desirable):
  
  $$h(m) - h(n) \leq w(n,m)$$
  
  where $m$ is parent of $n$. This ensures $f(n) \geq f(m)$.

A* Algorithm (2)

- **optimally efficient:** A* expands the minimal number of nodes possible with any given (consistent) heuristic.

- Time and space complexity:
  - **Worst case:** Cost function $f(n) = g(n)$
    
    $O(b^{c/e})$
  
  - **Best case:** Cost function $f(n) = g(n) + h^*(n)$
    
    $O(bd)$

A* Pseudo code

```plaintext
A-Star(Graph graph, Node start, Node goal, HeuristicFunction h)
1. O <-- make_priority_queue(startNode) // open list
2. C <-- make_hash_table // closed list
3. While O not empty loop
   1. n <-- O.remove_front() //O is sorted by $f(n) = g(n) + h(n)$ values
   2. If goal (n) return n
   3. If n is found on C --> continue
   4. //otherwise
   5. S <-- successors (n)
   6. For each node s in S
      1. Set s.g = n.g + w(n,s)
      2. Set s.parent = n //for path extraction
      3. Set s.h = h(s) //to calculate f
      4. O <-- s
   7. C <-- n
4. Return null //no goal found
```

A* Application Example

- Game: *Tales of Trolls and Treasures*

- Yellow dots are nodes in the search graph.
Duplicate Pruning

• Do not enter the father of the current state
  – With or without using closed-list

• Using a closed-list, check the closed list before entering new nodes to the open list
  – Note: in A*, h must be consistent!
  – Do not remove the original check

• Using a stack, check the current branch and stack status before entering new nodes

IDA* Algorithm

• Each iteration is a depth-first search that keeps track of the cost evaluation \( f = g + h \) of each node generated.

• If a node is generated whose cost exceeds the threshold for that iteration, its path is cut off.

• The cost threshold is initialized to the heuristic of the initial state.

IDA* Attributes

• The cost threshold increases in each iteration to the total cost of the lowest-cost node that was pruned during the previous iteration.

• The algorithm terminates when a goal state is reached whose total cost does not exceed the current threshold.

• Completeness and Optimality: Like A*

• Space complexity: \( \mathcal{O}(c) \)

• Time complexity*: \( \mathcal{O}(b^{c/2}) \)

IDA* Pseudo code

• **IDAStar-Main** (Node root)
  1. Set bound = \( f(root) \);
  2. WHILE (bound<infinity)
     1. Set bound= IDAStar(root, bound)

• **IDAStar** (node n, Double bound)
  1. if n is a goal, Exit algorithm and return goal
  2. if n has no children, return infinity
  3. \( fn = infinity \)
  4. for each child c of n, Set f=\( f(c) \)
     1. IF (f<= bound) \( fn=min(fn, \text{IDAStar}(c,bound)) \)
     2. Else \( fn=min(fn,f) \)
  5. Return fn
Exercise

- Q: What are the advantages of IDA* over:
  - A*
  - DFS (no closed list)
  - Uniform-Cost (closed list)?

<table>
<thead>
<tr>
<th>Alg.</th>
<th>Endless branch</th>
<th>Informed pruning</th>
<th>Space</th>
<th>Optimality</th>
</tr>
</thead>
<tbody>
<tr>
<td>A*</td>
<td>V</td>
<td></td>
<td>V</td>
<td></td>
</tr>
<tr>
<td>DFS</td>
<td>V</td>
<td>V</td>
<td>V</td>
<td></td>
</tr>
<tr>
<td>UC</td>
<td>V</td>
<td>V</td>
<td>V</td>
<td></td>
</tr>
</tbody>
</table>

Exercise – Cont.

- Q: When IDA* is not preferable?
- A:
  - A space graph with dense node duplications
  - When all the node costs are different, if the asymptotic complexity of A* is $O(N)$ - IDA*'s complexity can get in the worst case to $O(N^2)$.

- Q: What algorithm we'll get if we implement Greedy search on a uniform cost graph using
  - $h(n) = g(n)$?
  - $h(n) = -g(n)$?
- A:
  - $h(n) = g(n) \rightarrow$ BFS
  - $h(n) = -g(n) \rightarrow$ DFS

Exercise – True/False.

<table>
<thead>
<tr>
<th>Sentence</th>
<th>True/False</th>
</tr>
</thead>
<tbody>
<tr>
<td>DFS is not optimal</td>
<td>True, see DFS slides for example</td>
</tr>
<tr>
<td>Forward Search is always more preferable than Backwards Search</td>
<td>False, For example if there are more start nodes than goal nodes, or it is more natural to go backwards (expert systems).</td>
</tr>
<tr>
<td>ID alg. is always equal or slower than BrFS (assuming nodes expansion order is deterministic)</td>
<td>True. The last iteration expands nodes as BrFS.</td>
</tr>
<tr>
<td>IDS alg. is the exact implementation of BrFS</td>
<td>False. Its space complexity is $bd$ instead of $b^d$.</td>
</tr>
</tbody>
</table>