Brief Algorithms Explanation

This document provides explanations for all the algorithms of the file, alongside with their complexity for both space and time.

○ A* Algorithm

Description: A* is an informed search algorithm that finds the shortest path between two nodes in a graph. It uses a heuristic function to estimate the cost of reaching the goal from a given node. It combines the actual cost from the start node to the current node (g(n)) with the estimated cost from the current node to the goal node (h(n)) to calculate the total cost (f(n) = g(n) + h(n)). It uses a priority queue to select the node with the lowest f(n) value.
Steps:

1. Initialize the open list with the starting node.
2. While the open list is not empty:
   2.1. Select the node with the lowest f(n) value from the open list.
   2.2. If the node is the goal node, return the path.
   2.3. Generate the successors of the node.
   2.4. For each successor:
      2.4.1. Calculate g(n), h(n), and f(n).
      2.4.2. If the successor is not in the open or closed list, add it to the open list.
      2.4.3 If the successor is already in the open or closed list, and the current path to that successor is better than the previous path, update the successor.
      2.4.5. Move the current node to the closed list.

Complexity:
Time Complexity: $O(b^d)$ in the worst case, where b is the branching factor and d is the depth of the solution. The actual complexity depends heavily on the heuristic function.
Space Complexity: $O(b^d)$ in the worst case.

○ Alpha - Beta Pruning

Description: Alpha-Beta pruning is a search algorithm that reduces the number of nodes evaluated by the minimax algorithm in a game tree. It eliminates branches that cannot possibly affect the final decision. It maintains two values, alpha and beta, representing the minimum score that the maximizing player is assured of and the maximum score that the minimizing player is assured of.
Steps:

Perform a minimax search, but prune branches when alpha >= beta.

Complexity:
Time Complexity: $O(b^{(d/2)})$ in the best case, where b is the branching factor and d is the depth. $O(b^d)$ in the worst case.
Space Complexity: O(d)

- ## Beam Search

Description: Beam search is a heuristic search algorithm that explores a graph by expanding the most promising nodes in a limited set. It maintains a beam, which is a list of the best nodes found so far at each level. The beam width limits the number of nodes that are kept in the beam.
Steps:

1. Initialize the beam with the starting node.
2. For each level:
   2.1. Generate the successors of the nodes in the beam.
   2.2. Select the best successors (based on a heuristic function) to form the new beam.

Complexity:
Time Complexity: O(b\w\d) where b is branching factor, w is beam width, and d is depth.
Space Complexity: O(w\d)

- ## Best-First Search

Description: Best - first search is a search algorithm that explores a graph by expanding the most promising node according to a heuristic function. It uses a priority queue to select the node with the lowest heuristic value.
Steps:

1. Initialize the open list with the starting node.
2. While the open list is not empty:
   2.1. Select the node with the lowest heuristic value from the open list.
   2.2. If the node is the goal node, return the path.
   2.3. Generate the successors of the node.
   2.4. Add the successors to the open list.

Complexity:
Time Complexity: $O(b^d)$ in the worst case. The actual complexity depends on the heuristic.
Space Complexity: $O(b^d)$ in the worst case.

- ## Bidirectional Search

Description: Bidirectional search is a search algorithm that searches forward from the starting node and backward from the goal node simultaneously. It stops when the two searches meet.
Steps:

1. Perform a forward search from the starting node.
2. Perform a backward search from the goal node.
3. Stop when the two searches meet.

Complexity:
Time Complexity: $O(b^{(d/2)})$ where b is the branching factor and d is the depth of the solution.
Space Complexity: $O(b^{(d/2)})$.

- ## Branch and Bound

Description: Branch and bound is a search algorithm that systematically enumerates all candidate solutions and discards large subsets of fruitless candidates, by using upper and lower bound estimations of the quantity being optimized.
Steps:

1. Perform a depth-first search.
2. Maintain a bound on the best solution found so far.
3. Prune branches that cannot lead to a better solution.

Complexity:
Time Complexity: Exponential in the worst case.
Space Complexity: Depends on the implementation.

- ## Brute Force Search (British Museum)

Description: Brute force search is a search algorithm that systematically enumerates all possible solutions until the correct solution is found.
Steps:

1. Generate all possible solutions.
2. Check each solution until the correct solution is found.

Complexity:
Time Complexity: $O(b^d)$ in the worst case, where b is the branching factor and d is the depth of the solution.
Space Complexity: Depends on problem.

- ## Hill Climbing

Description: Hill climbing is a local search algorithm that repeatedly moves to the neighbor with the best heuristic value. It stops when it reaches a local maximum.
Steps:

1. Start with an initial solution.
2. Repeat:
   2.1. Generate the neighbors of the current solution.
   2.2. Move to the neighbor with the best heuristic value.
   2.3. Until no neighbor is better.

Complexity:
Time Complexity: Depends on the landscape of the search space.
Space Complexity: O(1).

- ## Min-Max Algorithm

Description: The minimax algorithm is a decision-making algorithm used in game theory for two-player zero-sum games. It recursively explores the game tree to determine the optimal move for the maximizing player, assuming that the minimizing player will make the best possible moves.

Steps:

1. Recursively explore the game tree.
2. At each level, select the move that maximizes or minimizes the score.

Complexity:

Time Complexity: $O(b^d)$, where b is the branching factor and d is the depth of the tree.
Space Complexity: O(d).