Grokking Artificial intelligence Algorithms
by Rishal Hurbans

Section 3 Intelligent Search

Defining

A heuristic is a set of rules used to evaluate a state. Heuristics are best used when an optimal solution is not very clear, rather its closer to an educated guess used to guide.

A* Search

_____This search algorithm uses heuristics to evaluate the next stage and find the optimal solution. Using the formula f(n)=g(n)+h(n) where h(n) represents the heuristic function and g(n) represents the cost from the start to the n. ***NOTE in Assignment one, I altered the A* formula a bit to find the f(n) for each edge and find the lowest value to pick the fastest route to the goal node instead of g(n) being the cost from start to n g(n) represented the weight and h(n) represented the straight line value given**** A* computes more effectively and unlike BFS and DFS doesn't check every possibility.

Adversarial Search

These kinds of problems expect one to counteract the actions of an opponent such as in a game of tic-tac-toe. Within these types of problems we have min-max search which aims to build a tree of possible outcomes based on moves that each player could make and favor paths that are advantageous to the agent while avoiding paths that are favorable to the opponent. This type of search simulates different possibilities and then scores them based on a heuristic based on the move performed. Min-max search attempts to discover as many states in the future as possible; but due to memory and computation limitations, discovering the entire game tree may not be realistic, so it searches to a specified depth. Since games like connect four can cause the tree of possible outcomes to explode in the number of possibilities, it is possible to use alpha-beta pruning to short circuit paths that are not desirable. The algorithm works by storing the best score for maximizing in alpha and the best for minimizing in beta. Alpha and beta start off as the worst score for each player, infinity.

Important Figures:

A* Function

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

g(n): cost of the path from the start node to node n

h(n): the cost from the heuristic function for node n

f(n): the cost of the path from the start node to node n plus the cost from the heuristic function for node n

Figure 3.4 The function for the A* search algorithm

A* flow

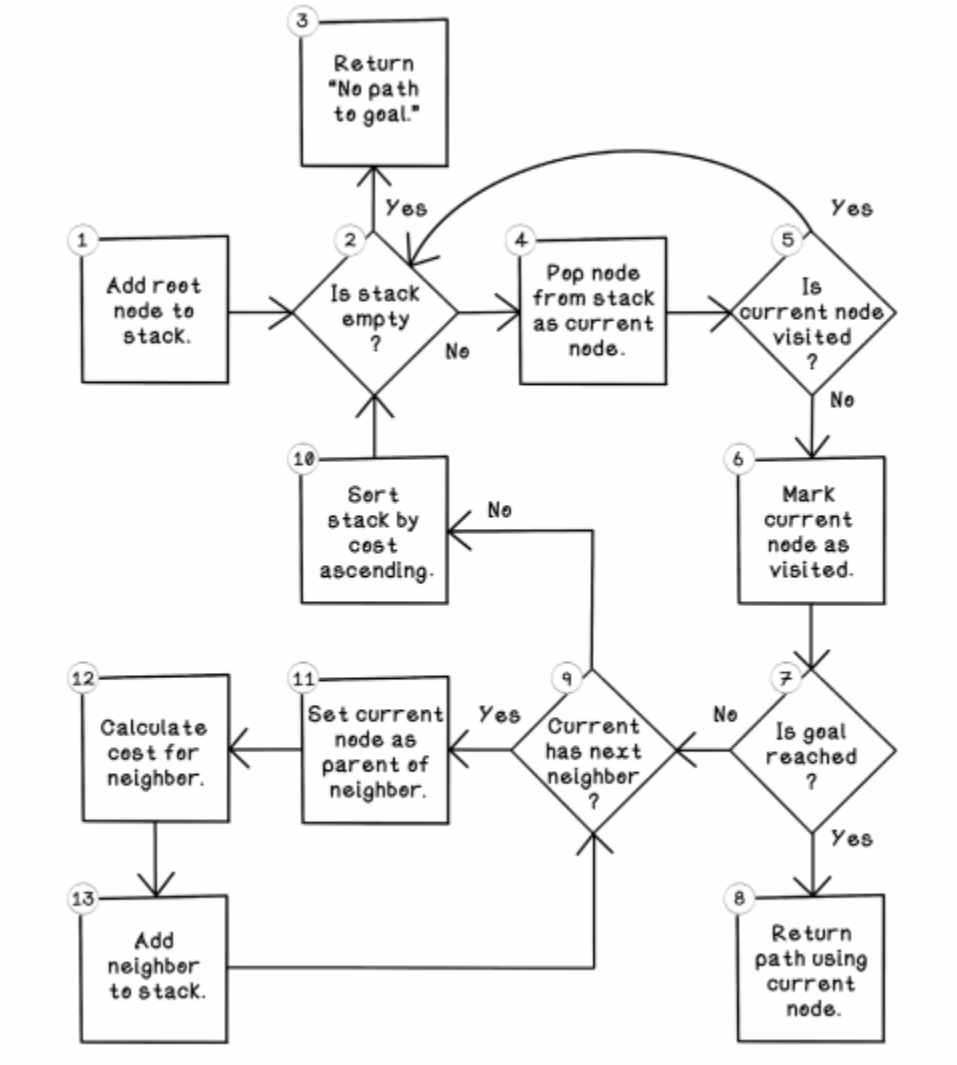

Figure 3.7 Flow for the A* search algorithm

Min-Max Algorithm

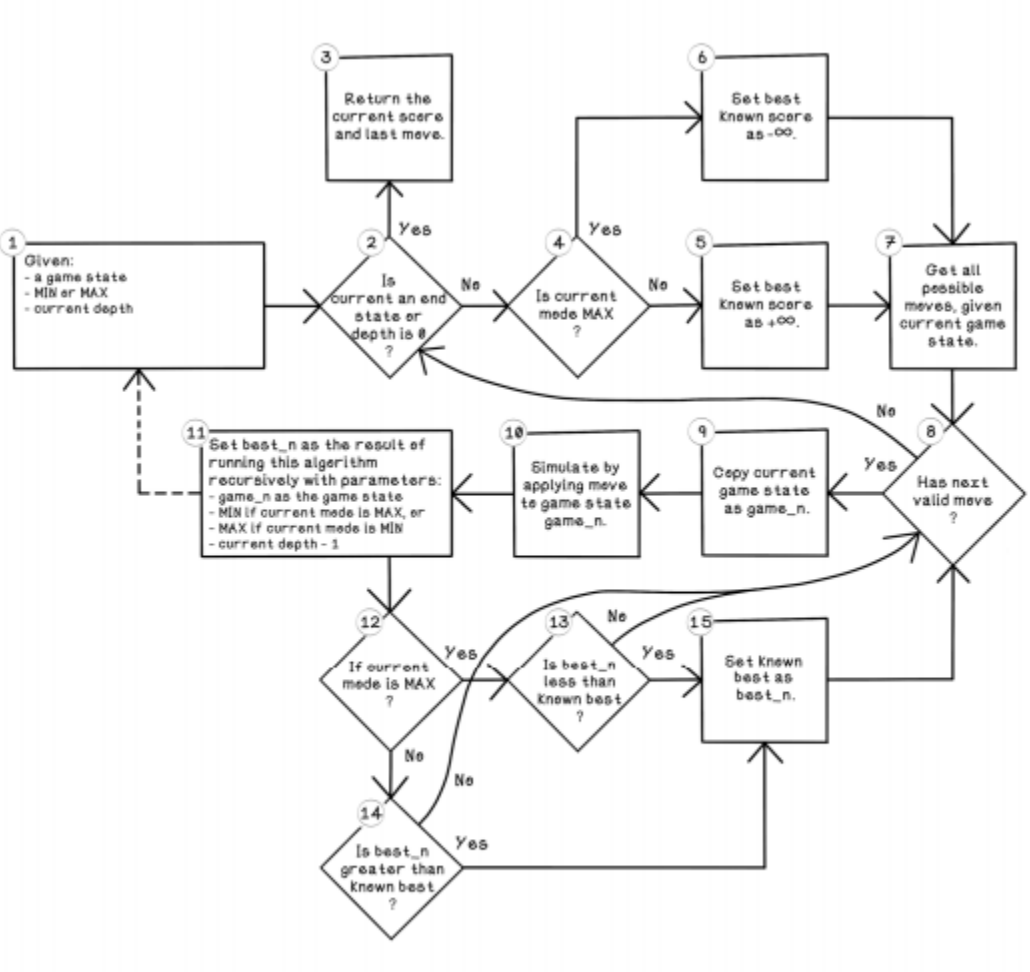

Figure 3.15 Flow for the min-max search algorithm

Alpha-Beta Pruning pseudocode

```
minmax_ab_pruning(state, depth, min_or_max, last_move, alpha, beta):
  let current score equal state.get_score
  if current_score is not equal to 0 or state.is_full or depth is equal to 0:
    return new Move(last_move, current_score)
  let best_score equal to min_or_max multiplied by -∞
  let best_move = -1
  for each possible choice (0 to 4 in a 5x4 board) as move:
    let neighbor equal to a copy of state
    execute current move on neighbor
    let best_neighbor equal
      minmax(neighbor, depth -1, min_or_max * -1, move, alpha, beta)
    if (best_neighbor.score is greater than best_score and min_or_max is MAX)
    or (best_neighbor.score is less than best_score and min_or_max is MIN):
      let best_score = best_neighbor.score
      let best_move = best_neighbor.move
      if best_score >= alpha:
        alpha = best_score
      if best_score <= beta:
        beta = best_score
    if alpha >= beta:
      break
  return new Move(best_move, best_score)
```