GPGC

COURSE TITLE: ARTIFICIAL INTELLIGENCE

TOPIC: SEARCH TECHNIQUES

ASSIGNMENT NO.: 01

SUBMITTED TO: MA'AM AMINA

SUBMITTED BY: WAQAR ALI

ROLL NO.: (117104)

DATE OF SUBMISSION: 18th NOV 2014

GOVT. POSTGRADUATE COLLEGE NO.2

MANDIAN, ABBOTTABAD.

Part- I

MIN-MAX/MINIMAX SEARCH

Minimax is a decision rule used in decision theory, game theory, statistics and philosophy for minimizing the possible loss for a worst case (maximum loss) scenario. Originally formulated for two-player zero-sum game theory, covering both the cases where players take alternate moves and those where they make simultaneous moves, it has also been extended to more complex games and to general decision making in the presence of uncertainty.

The minimax search is especially known for its usefulness in calculating the best move in two player games where all the information is available, such as chess or tic tac toe. It consists of navigating through a tree which captures all the possible moves in the game, where each move is represented in terms of loss and gain for one of the players. It follows that this can only be used to make decisions in zero-sum games, where one player's loss is the other player's gain. Theoretically, this search algorithm is based on Neumann's minimax theorem which states that in these types of games there is always a set of strategies which leads to both players gaining the same value and that seeing as this is the best possible value one can expect to gain, one should employ this set of strategies (Kulenovic, 2008).

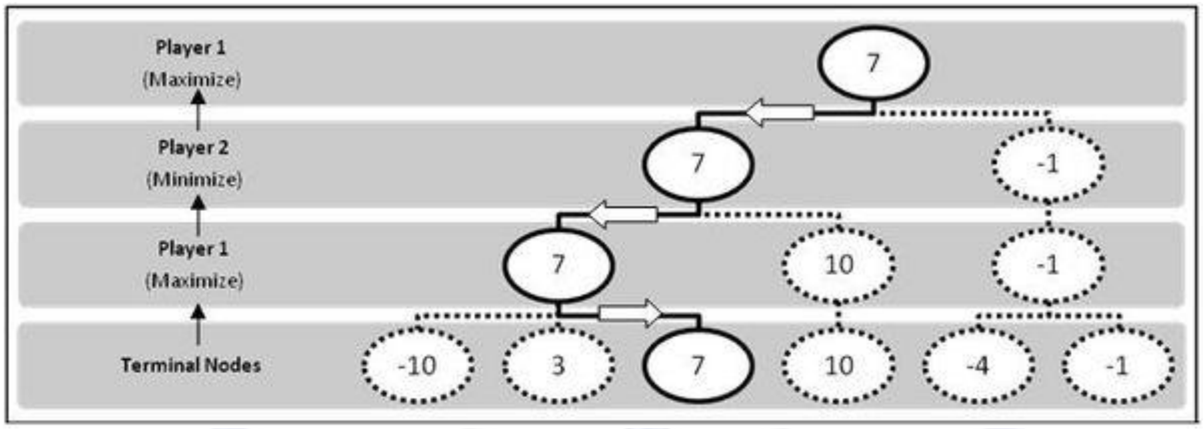
History

Jaap van den Herik's thesis (1983) contains a detailed account of the known publications on that topic. It concludes that although John von Neumann is usually associated with that concept (1928), primacy probably belongs to Émile Borel. Further there is a conceivable claim that the first to credit should go to Charles Babbage. The original minimax as defined by Von Neumann is based on exact values from game-terminal positions, whereas the minimax search suggested by Norbert Wiener is based on heuristic evaluations from positions a few moves distant, and far from the end of the game.

Description

The first step is to give each node a value in terms of utility for player 1. These values are derived from the values of the terminal nodes, which each represent an end of the game. Each level of the tree alternates between player 1's move, who is trying to maximize her score, and player 2's move, who is trying to minimize player 1's score in order to undermine her success. If the level above the terminal nodes represents player 2's possible moves, then the value of the nodes will be the lowest value among those of their children, the minimum. On the other hand, if this level represents the possible moves for player 1, the value of its nodes will be the highest value among those of their children. Finally, once all the nodes have been assigned values, it is decision time for player 1 who just picks the move which will lead to the highest value.

Following figure illustrates this process.



Pseudocode

This pseudocode is adapted from a version presented by Russell and Norvig (1995).

```
function minimaxDecision (game)
    for each possible move in game
        moveValue = minimaxValue(state, game)

    return move with highest value

function minimaxValue (state, game)
    if terminal node
        return utility

    if player 1's turn
        return highest value of children

    if player 2's turn
        return lowest value of children
```

Complications, Shortcomings and Future Directions

One shortcoming of this search is that it assumes that the opponent will act in such a way that he will effectively minimize your gains and maximize his own. However, this will not hold if your opponent is irrational, a beginner or a human who can be mistaken. In some cases, it may be best to gamble and hope that your opponent will not notice a subtle countermove.

Moreover, representing all the possible moves for simple games like tic tac toe is not so difficult but with games like chess, it becomes computationally expensive to represent them all. Therefore, programmers often rely on further algorithms to narrow the search, such as alpha-beta pruning. This consists of the stopping the calculation of the utility of one node when it is determined that there is no way its value could be higher than that of an adjacent node.

Another way of cutting down the search space is to only go to a certain depth, treat the moves at that level as makeshift terminal nodes and determine their values using heuristics. Still, this is problematic in cases where long term planning is necessary. However, this approach can be made more viable by looking for stable stages within a game and using these as intermediate terminal nodes. Despite these techniques, there is a debate as to whether using a minimax search is actually effective in such complex cases.

Finally, this only discusses games in which the next state is solely determined by the moves of the players involved. The final frontier of the minimax search are games which involve chance, such as backgammon and poker.

Part- II

Branch and Bound Search

Branch and bound (BB or B&B) is a search design paradigm for discrete and combinatorial optimization problems. A branch-and-bound search consists of a systematic enumeration of candidate solutions by means of state space search: the set of candidate solutions is thought of as forming a rooted tree with the full set at the root. The search explores branches of this tree, which represent subsets of the solution set. Before enumerating the candidate solutions of a branch, the branch is checked against upper and lower estimated bounds on the optimal solution, and is discarded if it cannot produce a better solution than the best one found so far by the algorithm.

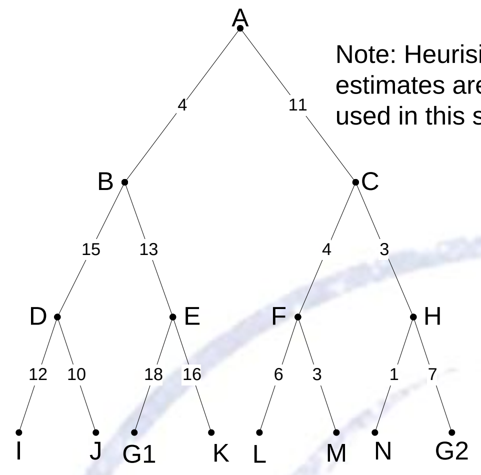
History

The method was first proposed by A. H. Land and A. G. Doig in 1960 for discrete programming, and has become the most commonly used tool for solving NP-hard optimization problems. The name "branch and bound" first occurred in the work of Little et al. on the traveling salesman problem.

Procedure

To conduct a branch-and-bound search,

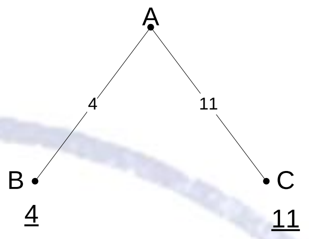
- Form a one-element queue consisting of a zero-length path that contains only the root node.
- Until the first path is the queue terminates at the goal node or the queue is empty,
    - Remove the first path from the queue; create new paths by extending the first path to all the neighbors of the terminal node.
    - Reject all new paths with loops.
    - *Add the remaining new paths, if any, to the queue.*
    - *Sort the entire queue by path length with least-cost paths in front.*
- If the goal node is found, announce success; otherwise announce failure.



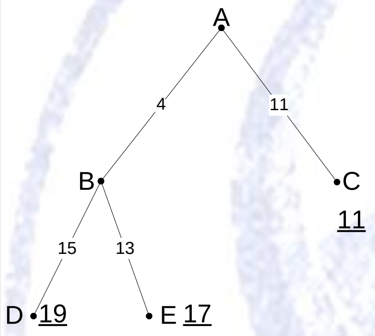
Note: Heurisitic estimates are not used in this search!

● Start with root node A.

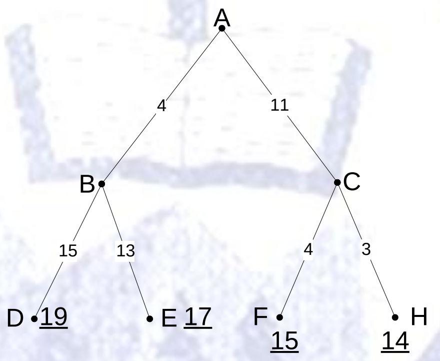
Paths from root are generated.



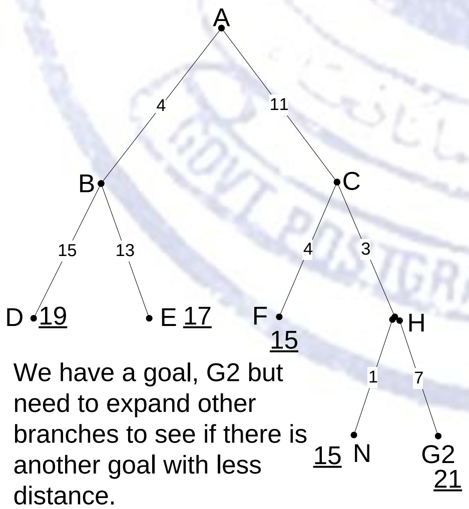
Since B has the least cost, we expand it.



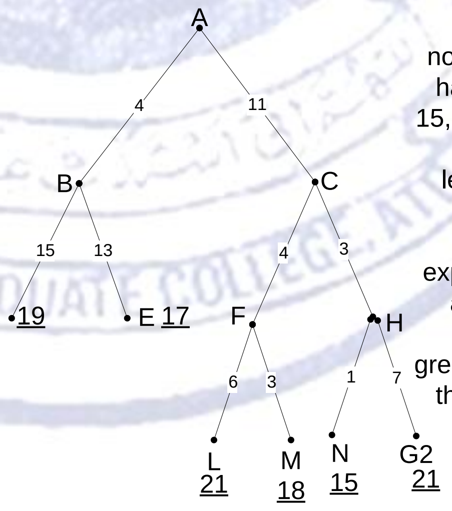
Of our 3 choices, C has the least cost so we'll expand it.



Node H has the least cost thus far, so we expand it.

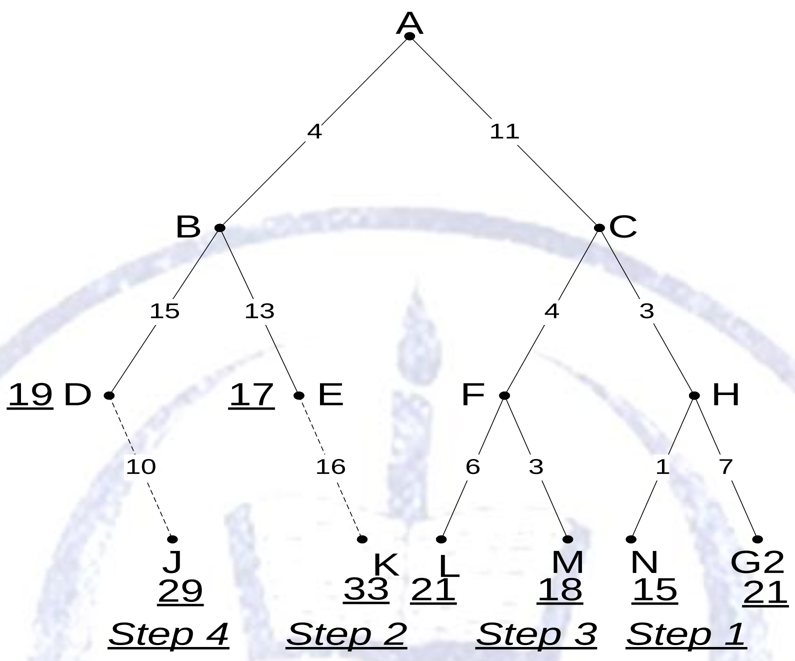


We have a goal, G2 but need to expand other branches to see if there is another goal with less distance.



Note: Both nodes F and N have a cost of 15, we chose to expand the leftmost node first. We continue expanding until all remaining paths are greater than 21, the cost of G2

B & B Example



All partial paths must be extended until their costs ≥ shortest path to goal.

- **Step 1:** The path to node N cannot be extended.
- **Step 2:** The next shortest path, A -> B -> E is extended its cost now exceeds 21.
- **Step 3:** The path to node M, much like to node N, cannot be extended.
- **Step 4:** The last partial path with a cost ≤ 21 is extended. Its cost, 29, now exceeds the start -> goal path.

The shortest path to a goal is A -> C -> H -> G2 with a cost of 21.

Part- III

Problem Solving in AI

Early AI researchers developed algorithms that imitated the step-by-step reasoning that humans use when they solve puzzles or make logical deductions. By the late 1980s and 1990s, AI research had also developed highly successful methods for dealing with uncertain or incomplete information, employing concepts from probability and economics.

For difficult problems, most of these algorithms can require enormous computational resources – most experience a "combinatorial explosion": the amount of memory or computer time required becomes astronomical when the problem goes beyond a certain size. The search for more efficient problem-solving algorithms is a high priority for AI research.

Human beings solve most of their problems using fast, intuitive judgments rather than the conscious, step-by-step deduction that early AI research was able to model. AI has made some progress at imitating this kind of "sub-symbolic" problem solving: embodied agent approaches emphasize the importance of sensorimotor skills to higher reasoning; neural net research attempts to simulate the structures inside the brain that give rise to this skill; statistical approaches to AI mimic the probabilistic nature of the human ability to guess.

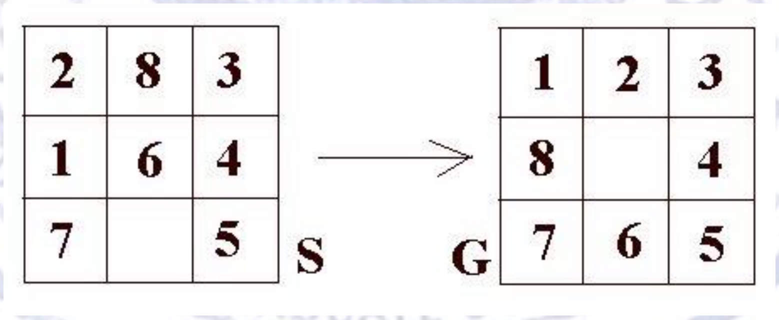
A well-defined problem can be described by:

- **Initial state**
- **Operator or successor function** - for any state x returns s(x), the set of states reachable from x with one action
- **State space** - all states reachable from initial by any sequence of actions
- **Path** - sequence through state space
- **Path cost** - function that assigns a cost to a path. Cost of a path is the sum of costs of individual actions along the path
- **Goal test** - test to determine if at goal state

Example Problems

The real art of problem solving is in deciding the description of the states and the operators.

- The 8-puzzle



State: location of blank

Operator: blank moves left, right, up and down

Goal Test: match G

Path Cost: each step costs 1 so cost is length of path