gradeup

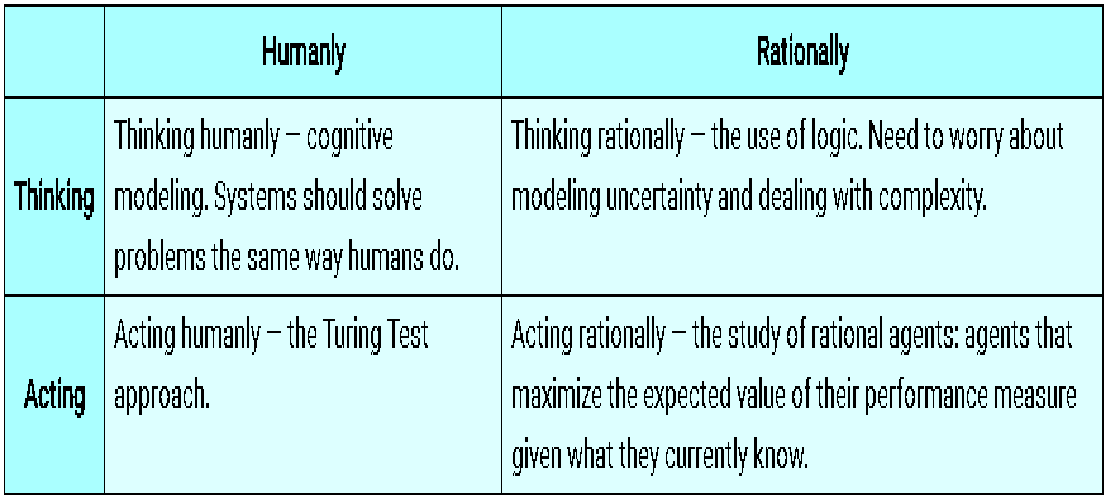
Approaches to AI

Approach to AI

Content:

Definition

There are hundreds of definitions of ***artificial intelligence***. Most contain a bias as to whether the writer of the definition sees AI as dealing with thinking versus acting, and whether he or she sees it as trying to model humans or capturing intelligence (rationality) abstractly.



	Humanly	Rationally
Thinking	Thinking humanly – cognitive modeling. Systems should solve problems the same way humans do.	Thinking rationally – the use of logic. Need to worry about modeling uncertainty and dealing with complexity.
Acting	Acting humanly – the Turing Test approach.	Acting rationally – the study of rational agents: agents that maximize the expected value of their performance measure given what they currently know.

Some definitions

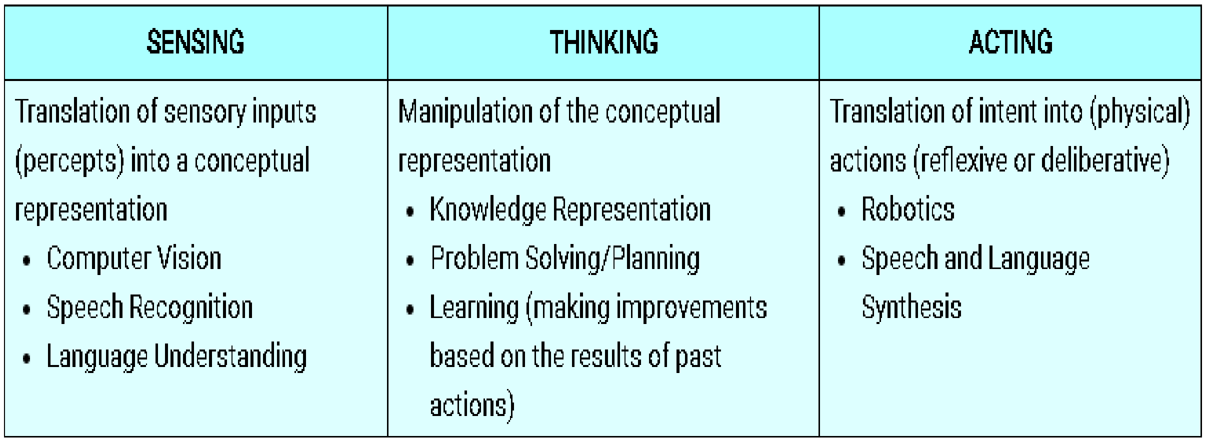
- The study of agents that receive percepts from the environment and perform actions. (Russell and Norvig)
- The science and engineering of making intelligent machines, especially intelligent computer programs (John McCarthy)
- The ability of a digital computer or computer-controlled robot to perform tasks commonly associated with intelligent beings (Encyclopædia Britannica)
- The study of ideas to bring into being machines that respond to stimulation consistent with traditional responses from humans, given the human capacity for contemplation, judgment and intention (Latanya Sweeney)
- The scientific understanding of the mechanisms underlying thought and intelligent behavior and their embodiment in machines (American Association for Artificial Intelligence)
- A branch of science which deals with helping machines find solutions to complex problems in a more human-like fashion (AI depot)

- A field of computer science that seeks to understand and implement computer-based technology that can simulate characteristics of human intelligence and human sensory capabilities (Raoul Smith)

AI researchers study the nature of intelligence. They don't (necessarily) try to build androids because

1. It was study of principles of aerodynamics, not the attempt to make mechanical birds, that enabled human flight.
2. We already know how to make new humans anyway (sorry, no hyperlinks here).

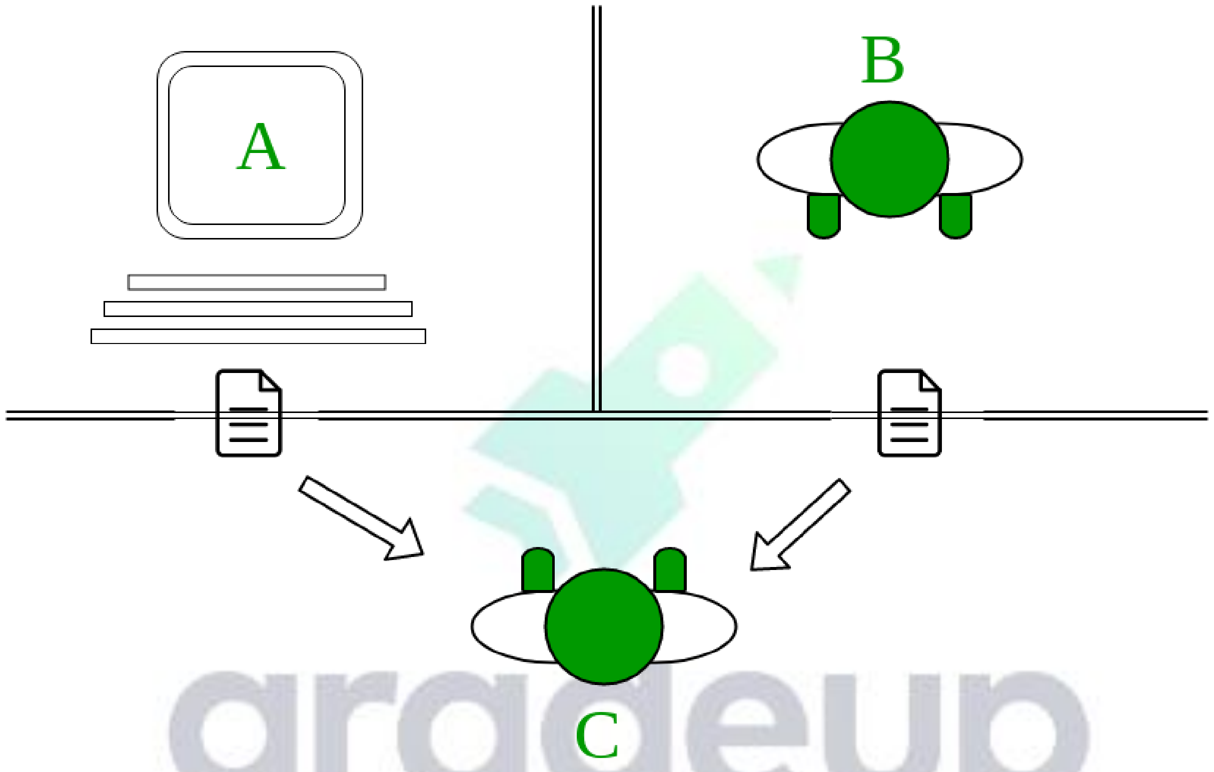
Intelligence involves sensing, thinking, and acting.



SENSING	THINKING	ACTING
Translation of sensory inputs (percepts) into a conceptual representation • Computer Vision • Speech Recognition • Language Understanding	Manipulation of the conceptual representation • Knowledge Representation • Problem Solving/Planning • Learning (making improvements based on the results of past actions)	Translation of intent into (physical) actions (reflexive or deliberative) • Robotics • Speech and Language Synthesis

from other two players. The interrogator job is to try which one is human and which one is computer by asking questions from both of them. To make the things harder computer is trying to make the interrogator guess wrongly. In other words computer would try to alike from human as Turing Test in Artificial Intelligence

Assume a game of three players having two humans and one computer, an interrogator(as human) is isolated much as possible.



The "standard interpretation" of the Turing Test, in which player C, the interrogator, is given the task of trying to determine which player - A or B - is a computer and which is a human. The interrogator is limited to using the responses to written questions to make the determination.

If interrogator wouldn't be able to distinguish the answers provided by both human and computer then the computer passes the test and machine(computer) is considered as intelligent as human. In other words, a computer would be considered intelligent if it's conversation couldn't be easily distinguished from a human's. The whole conversation would be limited to a text-only channel such as a computer keyboard and screen.

He also proposed that by the year 2000 a computer "would be able to play the imitation game so well that an average interrogator will not have more than a 70% chance of making the right identification (machine or human) after five minutes of questioning." No computer has come close to this standard.

What is an Intelligent Agent?

An agent which acts in a way that is expected to maximize to its performance measure, given the evidence provided by what it perceived and whatever built-in knowledge it has.

The performance measure defines the criterion of success for an agent.

State Space Representation

we study the concept of state space and different searches that can be used to explore the search space in order to find a solution. Before an AI problem can be proved it must be represented as a state space. It is searched to find a solution to the problem. It is essentially consists of a set of nodes representing each state of the problem, arcs between nodes representing the legal moves from one state to another, an initial state and a goal state. Every state space takes the form of a tree or a graph.

Factors that control which search algorithm or technique will be used include the type of the problem and the how the problem can represented. Search techniques that will be studied in the course include:

- Depth First Search
- Depth First Search with Iterative Deepening
- Breadth First Search

- Best First Search
- Hill Climbing
- Branch and Bound Techniques
- A* Algorithm

Classic AI Problems

Three of it which will be referred to in this section is the Traveling Salesman problem and the Towers of Hanoi problem and the 8 puzzle.

Traveling Salesman

A salesman has a list of cities, every of which he must visit exactly once. There are direct roads between every pair of cities on the list. Find the route that the salesman should follow for the shortest trip that starts and finishes at any one of the cities.

Towers of Hanoi

In a monastery in the deepest Tibet there are three crystal columns and 64 golden rings. The rings are different sizes and rest over the columns. At the beginning of time all the rings rested on the leftmost column, and since than the monks have toiled ceaselessly moving the rings one by one between the columns. In moving rings a larger ring must not be placed on a smaller ring. Only one ring at a time can be moved from one column to the next. A simplified version of this problem which will consider includes only 2 or 3 rings instead of 64.

8-Puzzle

1	2	3
8	■	4
7	6	5

The 8-Puzzle involves moving the tiles on the board above into a particular arrangement. The black square on the board represents a space. The player can move a tile into the space, freeing that position for another tile to be moved into and so on.

For example, given the initial state above we want the tiles to be moved so that the following goal state may be achieved.

1	8	3
2	6	4
7	■	5

Problem Representation

Factors that need to be taken into consideration when developing a state space representation. Factors that must be addressed are:

• What is the goal to be achieved?

• What are the legal moves or actions?

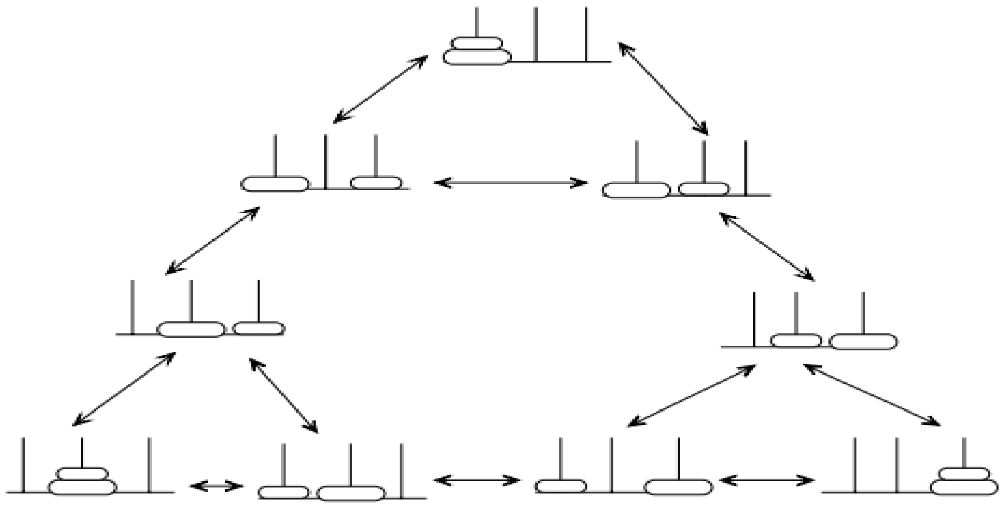
• What knowledge needs to be represented in the state description?

• Type of problem - There are basically three types of problems. Many problems need a representation, e.g. crossword puzzles. Another problems need yes or no response showing whether a solution can be found or not. Finally, the last type problem are those that need a solution path as an output, e.g. mathematical theorems, Towers of Hanoi. In these cases we know the goal state and we need to know how to achieve this state

• Good solution vs. Good enough solution - For many problems a good enough solution is sufficient. For example, theorem proving , eight squares. However, some problems need a good or optimal solution, e.g. the traveling salesman problem.

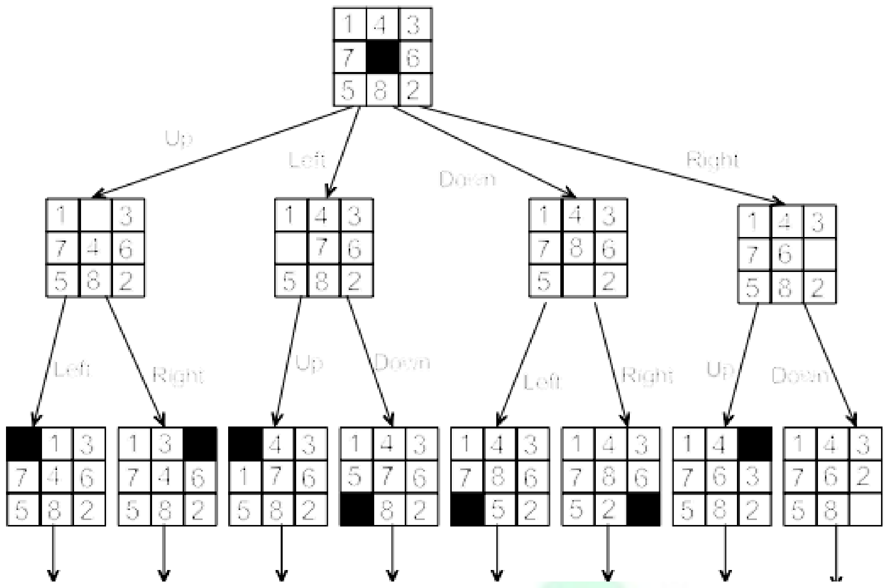
Towers Hanoi

A possible state space representation of the Towers Hanoi problem using a graph is indicated in Figure 3.1.



The legal moves in this state space involve moving one ring from one pole to another, moving one ring at a time, and ensuring that a larger ring is not placed on a smaller ring.

8-Puzzle



Although a player moves the tiles around the board to change the configuration of tiles. However, we will define the legal moves in terms of moving the space. The space can be moved up, down, left and right.

Summary

- A state space is a group of descriptions or states.
- Every search problem consists of: < One or more initial states. < A set of legal actions. Actions are represented by operators or moves applied to each state. For example, the operators in a state space representation of the 8-puzzle problem are left, right, up and down. < One or more goal states.
- The number of operators are problem dependant and specific to a particular state space representation. The more operators the larger the branching factor of the state space. Thus, the number of operators

should kept to a minimum, e.g. 8-puzzle: operations are defined in terms of moving the space instead of the tiles.

- Why generate the state space at run-time, and not just have it built in advance?
- A search algorithm is registered to a state space representation to find a solution path. Every search algorithm applies a particular search strategy

Graphs versus Trees

If it states in the solution space can be frequent more than once a directed graph is used to represent the solution space. In a graph more than one move sequence can be used to get from one state to another. Moves in a graph can be incomplete. In a graph there is more than one path to a goal whereas in a tree a path to a goal is more clearly distinguishable. A goal state may need to appear more than once in a tree. Search algorithms for graphs have to cater for possible loops and cycles in the graph. Trees may be more "efficient" for representing such problems as loops and cycles do not have to be catered for. The entire tree or graph will not be generated.

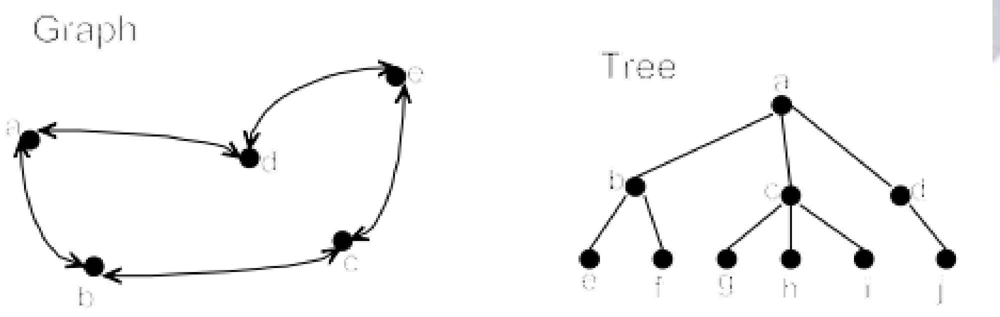


Figure 4.1: Graph vs. Tree

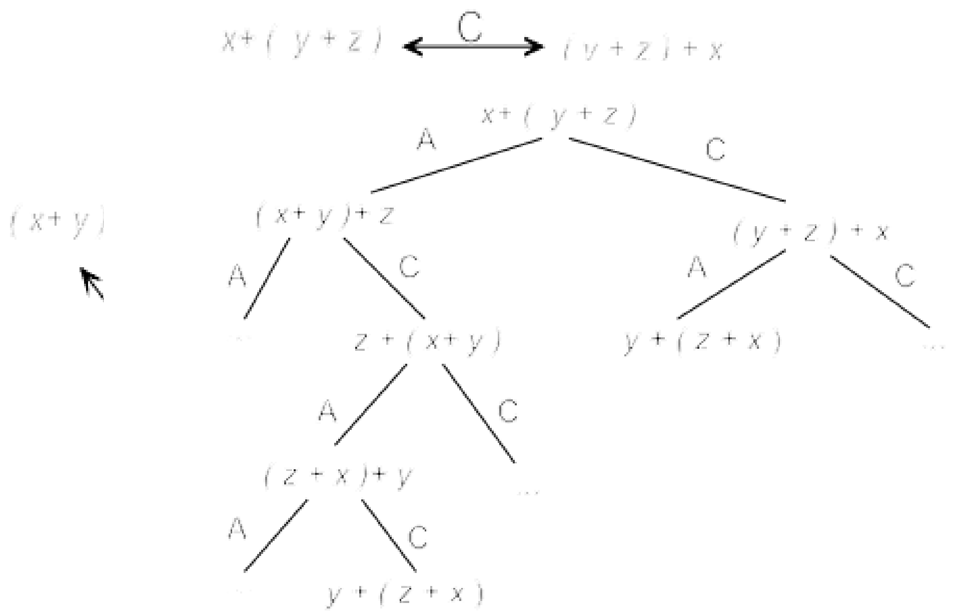
Notice that there is more than one path connecting two particular nodes in a graph whereas this is not so in a tree

Example:

Prove x + (y +z) = y + (z+x) given

L+(M+N) = (L+M) + N ....(A)

M+N = N+M.......................(C)



Heuristic Search

DFS and BFS need too much memory to generate an entire state space - in these cases heuristic search is used. It help us to reduce the size of the search space. An evaluation function is applied to every goal to assess how promising it is in leading to the goal. Examples of heuristic searches:

- Best first search
- A∗ algorithm
- Hill climbing

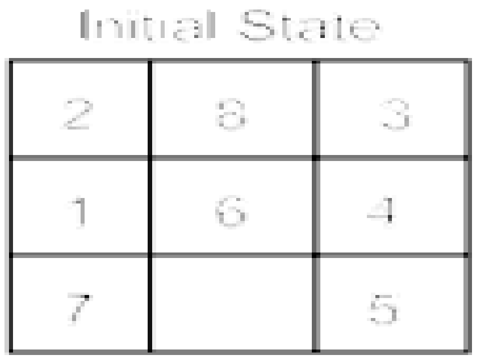
It incorporate the use of domain-specific knowledge in the method of choosing which node to visit next in the search process. Search methods that include the use of domain knowledge in the form of heuristics are described as “weak” search methods. The knowledge used is “weak” in that it usually helps but does not always help to find a solution.

Calculating heuristics

These are rules of thumb that may find a solution but are not guaranteed to. It functions have been defined as evaluation functions that estimate the cost from a node to the goal node. The incorporation of domain knowledge into the search process by means of heuristics is meant to speed up the search process. These functions are not guaranteed to be completely accurate. If they were there would be no need for the search process. These values are greater than and equal to zero for all nodes. These values are seen as an approximate cost of finding a solution. The value of zero shows the state is a goal state.

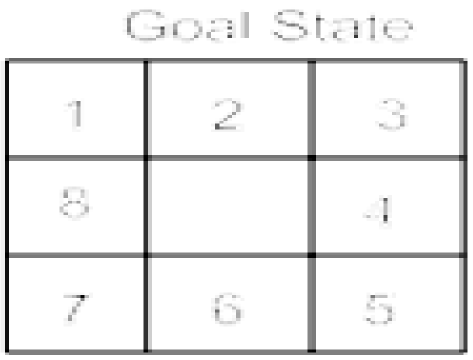
It never overestimates the cost to the goal is referred to as an admissible heuristic. Not all heuristics are necessarily admissible. A heuristic value of infinity shows the state is a “deadend” and is not going to lead anywhere. A best heuristic must not take long to compute. These are often defined on a simplified or relaxed version of the problem, e.g. the number of tiles that are out of place. It function h_1 is better than some heuristic function h_2 if fewer nodes are expanded during the search when h_1 is used than when h_2 is used.

Example: The 8-puzzle problem



Initial State

2	8	3
1	6	4
7		5



Goal State

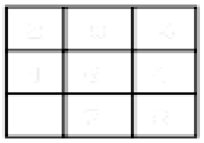
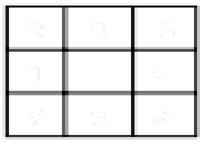
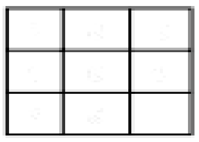
1	2	3
8		4
7	6	5

It serve as a heuristic function for the 8-Puzzle problem:

• Number of tiles out of place - count the number of tiles out of place in each state compared to the goal .

• Sum all the distance that the tiles are out of place.

• Tile reversals - multiple the number of tile reversals by 2.

State	Tiles out of place	Sum of distances out of place	2 x the number of direct tile reversals
	5	6	0
	3	4	0
	5	6	0

Game Playing

One of the earliest areas in artificial intelligence is game playing. We study search algorithms that are able to look ahead in order to decide which shift made in a two-person zero-sum game, i.e. a game in which there can be at most one winner and two players. Firstly, we will look at games for which the state space is small enough for the entire space to be generated. We will then look at examples for which the entire state space cannot be generated. In the latter case heuristics will be used to guide the search.

The Game Nim

The game begins with a pile of seven tokens. Two players can play the game at a time, and there is one winner and one loser. Each player has a turn to divide the stack of tokens. A stack can only be separated into two stacks of different size. The game tree for the game is illustrated below.

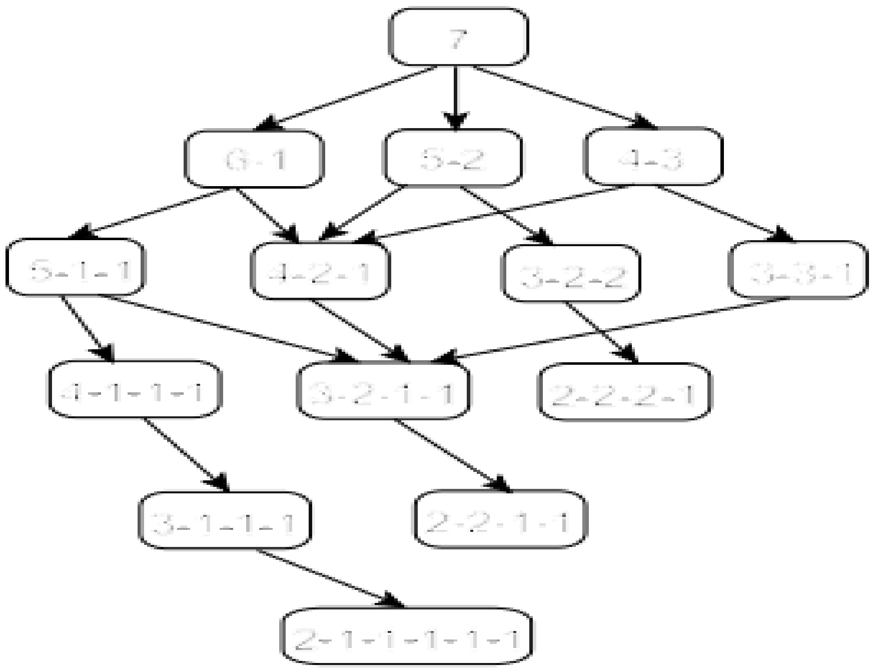
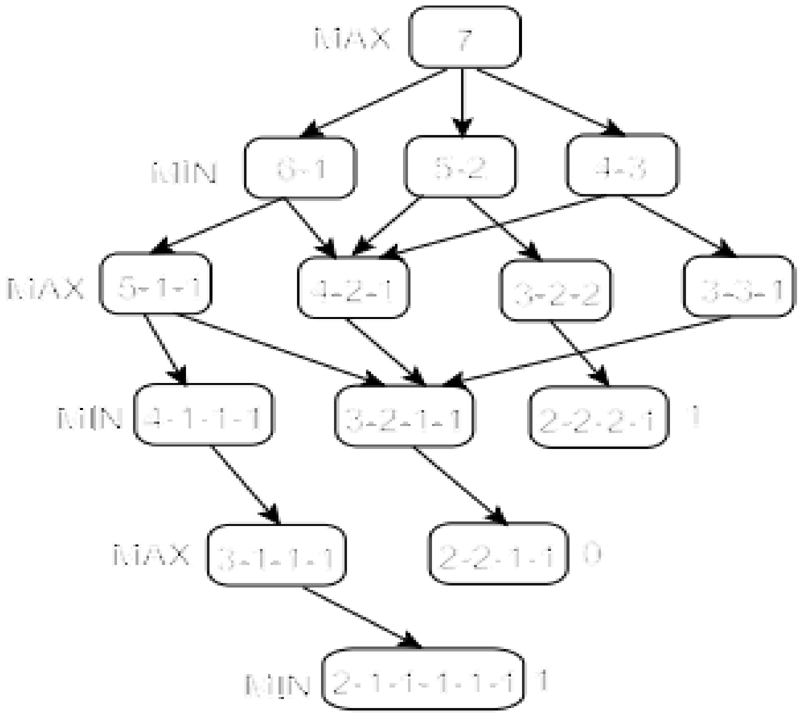
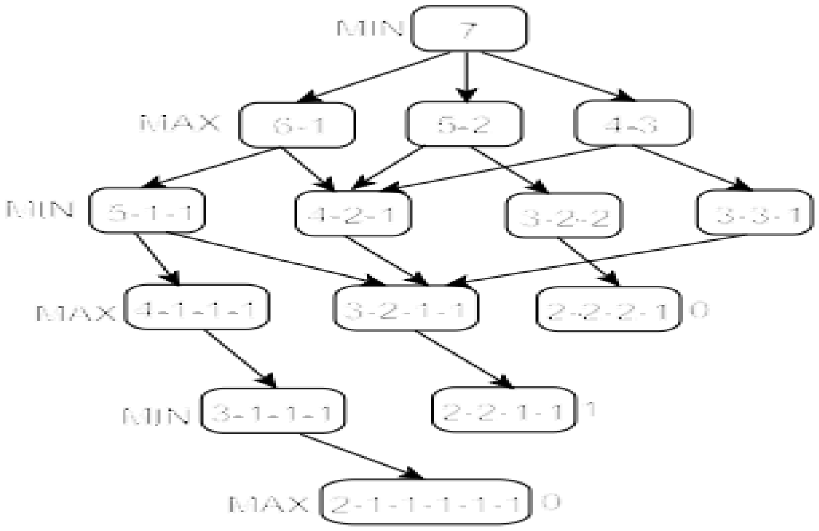


Figure 15.1: NIM state space

If the game tree is small enough we can generate the entire tree and look ahead to decide which move to make. In a game where there are two players and only one can win, we want to maximize the score of one player MAX and minimize the score of the other MIN. The game tree with MAX playing first is depicted in Figure . The MAX and MIN labels indicate which player's turn it is to divide the stack of tokens. The 0 at a leaf node indicates a loss for MAX and a win for MIN and the 1 represents a win for MAX and a loss for MIN. Figure depicts the game space if MIN plays first.



Max Play First



Min Plays First

The Minimax Search Algorithm

The players in the game are referred to as MAX and MIN. MAX represents the person who is trying to win the game and hence maximize his or her score. It represents the opponent who is trying to minimize MAX's score. The technique assumes that the opponent has and uses the same knowledge of the state space as MAX. Every level in the search space contains a label MAX or MIN showing whose move it is at that particular point in the game. This algorithm is used to look ahead and decide which move to make first. If the game space is small enough the entire space can be generated and leaf nodes can be allocated a win (1) or loss (0) value. These values can then be propagated back up the tree to decide which node to use. In propagating the values back up the tree a MAX node is given the maximum value of all its children and MIN node is given the minimum values of all its children.

Algorithm 1: Minimax Algorithm

Repeat

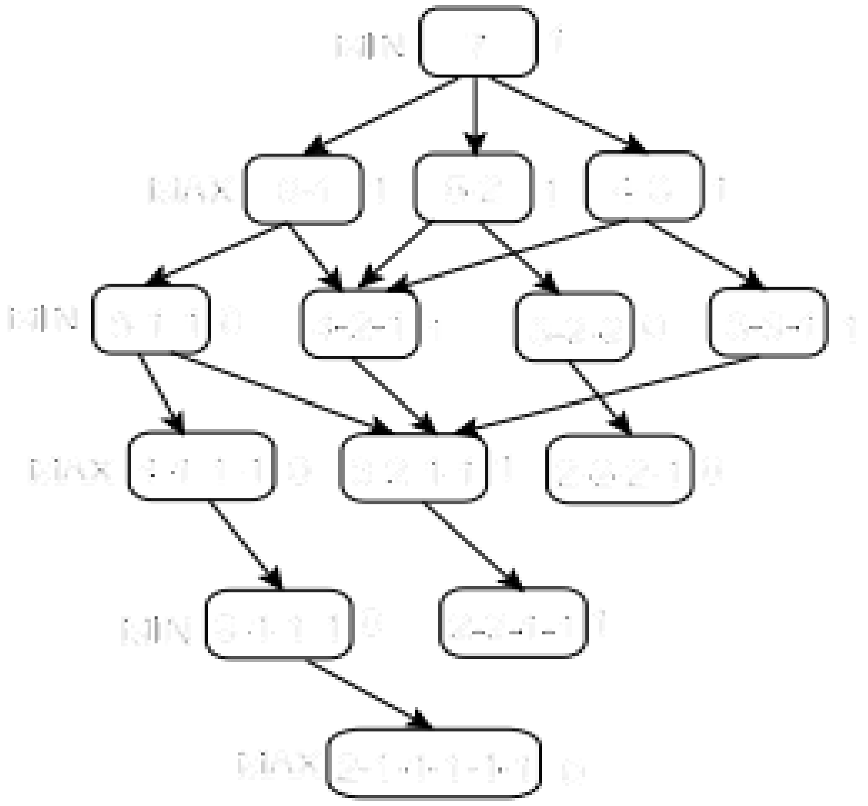
1. If the limit of search has been reached, compute the static value of the current position relative to the appropriate player. Report the result.

2. Otherwise, if the level is a minimizing level, use the minimax on the children of the current position. Report the minimum value of the results.

3. Otherwise, if the level is a maximizing level, use the minimax on the children of the current position. Report the maximum of the results.

Until the entire tree is traversed

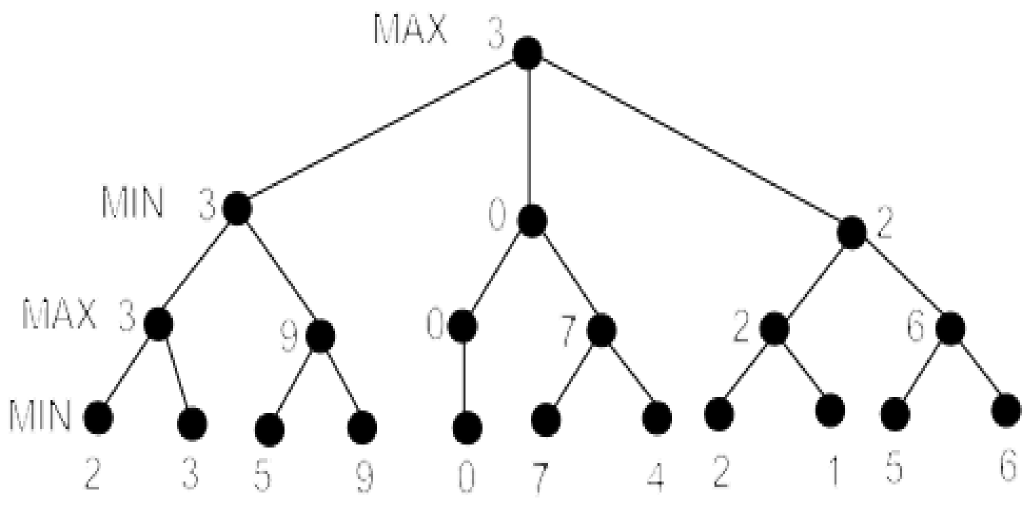
illustrates the minimax algorithm applied to the NIM game tree



Minimax applied to NIM

The value propagated to the top of the tree is a 1. Thus, no matter which move MIN makes, MAX still has an option that can lead to a win.

If the game space is large the game tree can be generated to a particular depth or ply. However, win and loss values cannot be assigned to leaf nodes at the cut-off depth as these are not final nodes. A heuristic value is used instead. The heuristic values are an estimate of how promising the node is in leading to a win for MAX. The depth to which a tree is searched is dependant on the time and computer resource limitations. One of the problems associated with generating a state space to a certain depth is the horizon effect. This refers to the problem of a look-ahead to a certain depth not detecting a promising path and instead leading you to a worse situation. Figure 14.5 illustrates the minimax algorithm applied to a game tree that has been generated to a particular depth. The integer values at the cutoff depth have been calculated using a heuristic function.



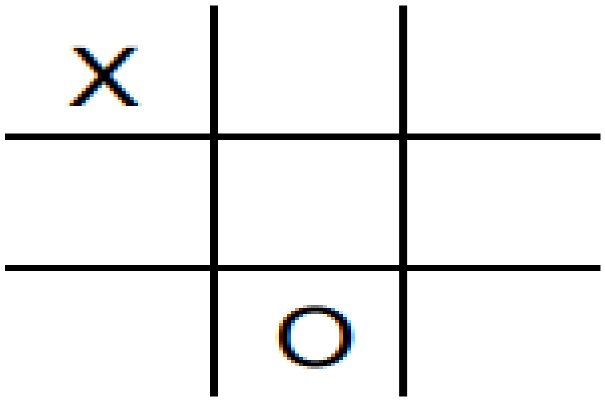
The function used to calculate the heuristic values at the cutoff depth is problem dependant and differs from one game to the next. For example, a heuristic function that can be used for the game tic-tac-toe : h(n) = x(n) - o(n) where x(n) is the total of MAX's possible winning lines (we assume MAX is playing x); o(n) is the total of the opponent's, i.e. MIN's winning lines and h(n) is the total evaluation for a state n. The following examples illustrate the calculation of h(n):

Example:

X has 6 possible winning paths

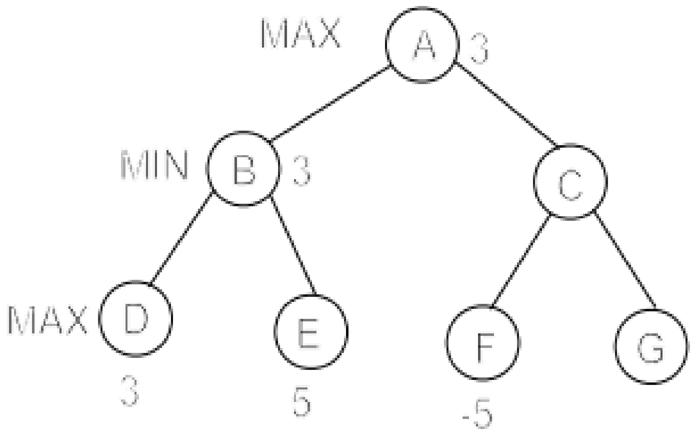
O has 5 possible winning paths

h(n) = 6 - 5 = 1



Alpha-Beta Pruning

The main disadvantage of the minimax algorithm is that all the nodes in the game tree cutoff to a certain depth are examined. Alpha-beta pruning helps reduce the



number of nodes explored.

The children of B have static values of 3 and 5 and B is at a MIN level, thus the value return to B is 3. If we look at F it has a value of -5. Thus, the value returned to C will be at most -5. However, A is at a MAX level thus the value at A will be at least three which is greater than -5. Thus, no matter what the value of G is, A will have a value of 3. Therefore, we do not need to explore and evaluate it G a s the its value will have not effect on the value returned to A. This is called an alpha cut. Similarly, we can get a beta-cut if the value of a subtree will always be bigger than the value returned by its sibling to a node at a minimizing level.

Minimax Algorithm with Alpha-Beta Pruning

```
Set alpha to -infinity and set beta to infinity
If the node is a leaf node return the value
If the node is a min node then
  For each of the children apply the minimax algorithm with alpha-beta pruning.
  If the value returned by a child is less then beta set beta to this value
  If at any stage beta is less than or equal to alpha do not examine any more
  children
  Return the value of beta
If the node is a max node
  For each of the children apply the minimax algorithm with alpha-beta pruning.
  If the value returned by a child is greater then alpha set alpha to this value
```

If at any stage alpha is greater than or equal to beta do not examine any more children

Return the value of alpha