Conversion Masters in IT (MIT)
AI as Representation and Search

(‘Intelligent’ Game Playing Algorithms)
Lecture 003

Sandro Spina

Heuristic Search

- AI problem solvers employ heuristics in the situations:
  - A problem may not have an exact solution. Eg Medical diagnosis. A given set of symptoms may have several possible causes. Doctors use heuristics to choose the most likely diagnosis.
  - A problem may have an exact solution BUT the computational cost of finding it may be prohibitive. In many problems (such as chess, GI, etc), state space grows exponentially with the depth of the search. Using a heuristic algorithm can (**its designer hopes**) defeat this combinatorial explosion and find an **acceptable** solution.
- We'll discuss the second scenario. Always keep in mind that heuristics are **fallible**. Game playing and theorem proving are two areas of AI which require heuristics.

An algorithm for heuristic search (Best-First Search … hill climbing for now)

- The simplest way to implement a heuristic is through a procedure called ***hill climbing***.

- Algorithm:
  - Expand current state and evaluate children.
  - If there exists a better child
    - The best child is becomes the current state
    - Repeat loop
  - If not … then the current state is the solution

- Problem of becoming stuck at a local maxima
- Thus if state is not a goal but just a local maxima the algorithm fails to find a solution !!!

An algorithm for heuristic search (Best-First Search … implementation)

- The evaluation function has to be sufficiently informative to avoid local maxima. However if this happens we need a method (priority queue) by which to recover from this local maxima.

- Like DFS and BFS, best-first search uses lists to maintain states:
  - OPEN keeps track of the current fringe of the search
  - CLOSED records states already visited

- An added step is necessary to ORDER the states on the *open* list according to some heuristic estimate of their 'closeness' to a goal.

- Hence, each iteration effectively considers the most 'promising' state on the *open* list.

The pseudo-code !!!

- Open := [Start]
- Closed := []
- While open != [] do
  - Remove leftmost state X from open
  - If X=Goal return path from Str to X else
    - Generate children of X and for each do
      - If child is not on open or closed
        - Assign heuristic value to child
        - Add the child to open
      - If child is already on open
        - Give state in open shorter path (if shorter found)
      - If child is already on closed
        - If child reached from shorter path then remove from closed and add child to open
    - Put X on closed
    - Re-order states in open by heuristic merit (leftmost)

Some notes on Best First Search

- By updating the ancestor history of nodes on open and closed when they are rediscovered, the algorithm is more likely to find a shorter path to a goal
- Always selects the most promising state on open for further expansion
- Since the heuristic might be erroneous, it does not abandon all the other states but maintains them on open
- IMP the open list allows backtracking from paths that fail to produce a goal

Example … 8-puzzle (part i)

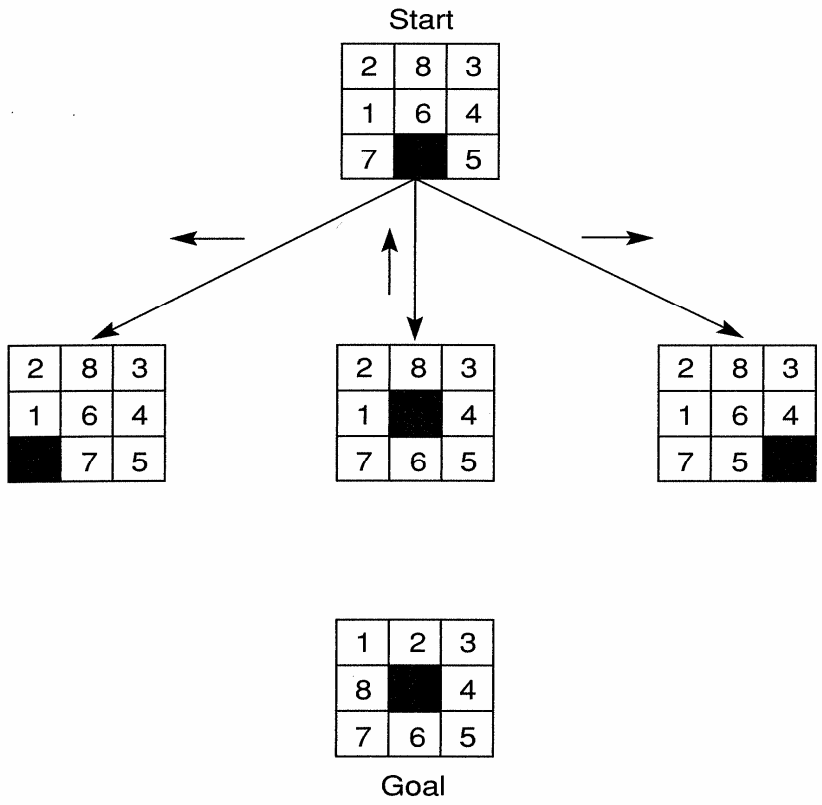


Figure 4.6 The start state, first set of moves, and goal state for an 8-puzzle instance.

The simplest heuristic counts the tiles out of place in each state when it is compared with the goal.

Or

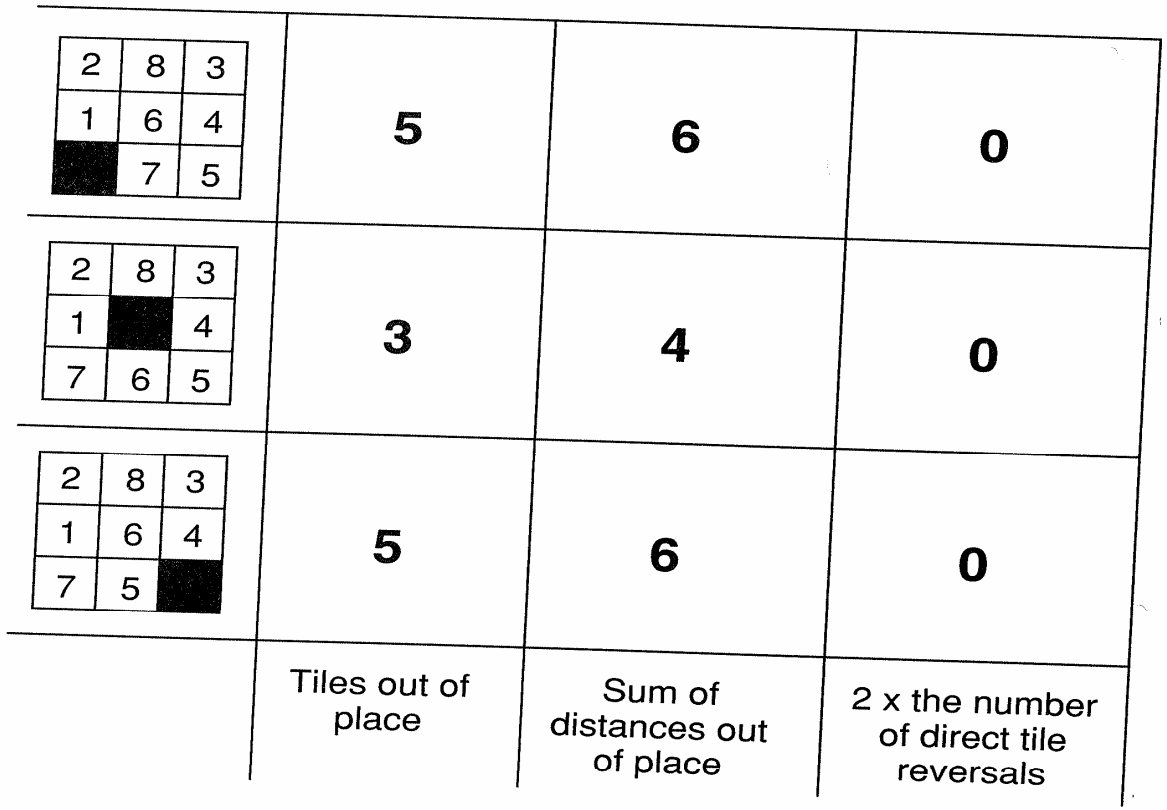
This + taking into account the distance the tiles are out of place.

Example … 8-puzzle (part ii)

- But both heuristics underestimate the difficulty of tile reversals. That is, if two tiles are next to each other and the goal requires their being in opposite locations, it takes (many) more than two moves to put them back in place.

- Therefore we add a third measure to include in the heuristic that takes into account adjacent tiles. For example we multiply by 2 for each direct tile reversal

Example … 8-puzzle (part iii)



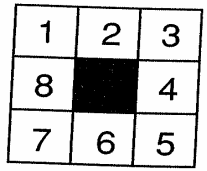
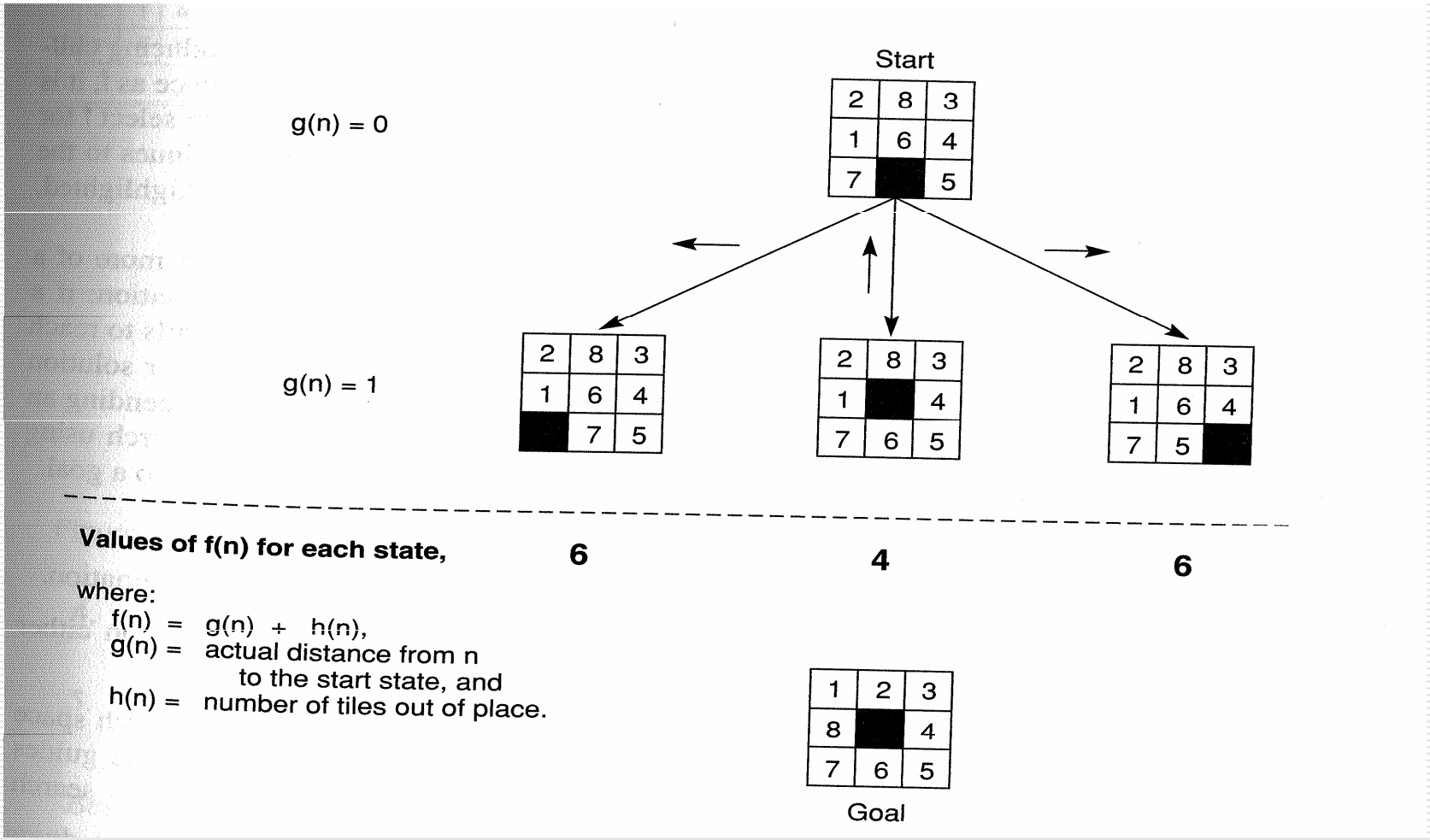


Figure 4.8 Three heuristics applied to states in the 8-puzzle.

Example … 8-puzzle (part iv)

- We can (not necessarily) combine these heuristics. Sum of distances out of place seems the most promising,
- Moreover since we want the shortest path to a solution we keep track of the number of tile movements we've done to reach a particular state,
- F(n) = g(n) + h(n)

Example … 8-puzzle (part v)



Example … 8-puzzle (part vi)

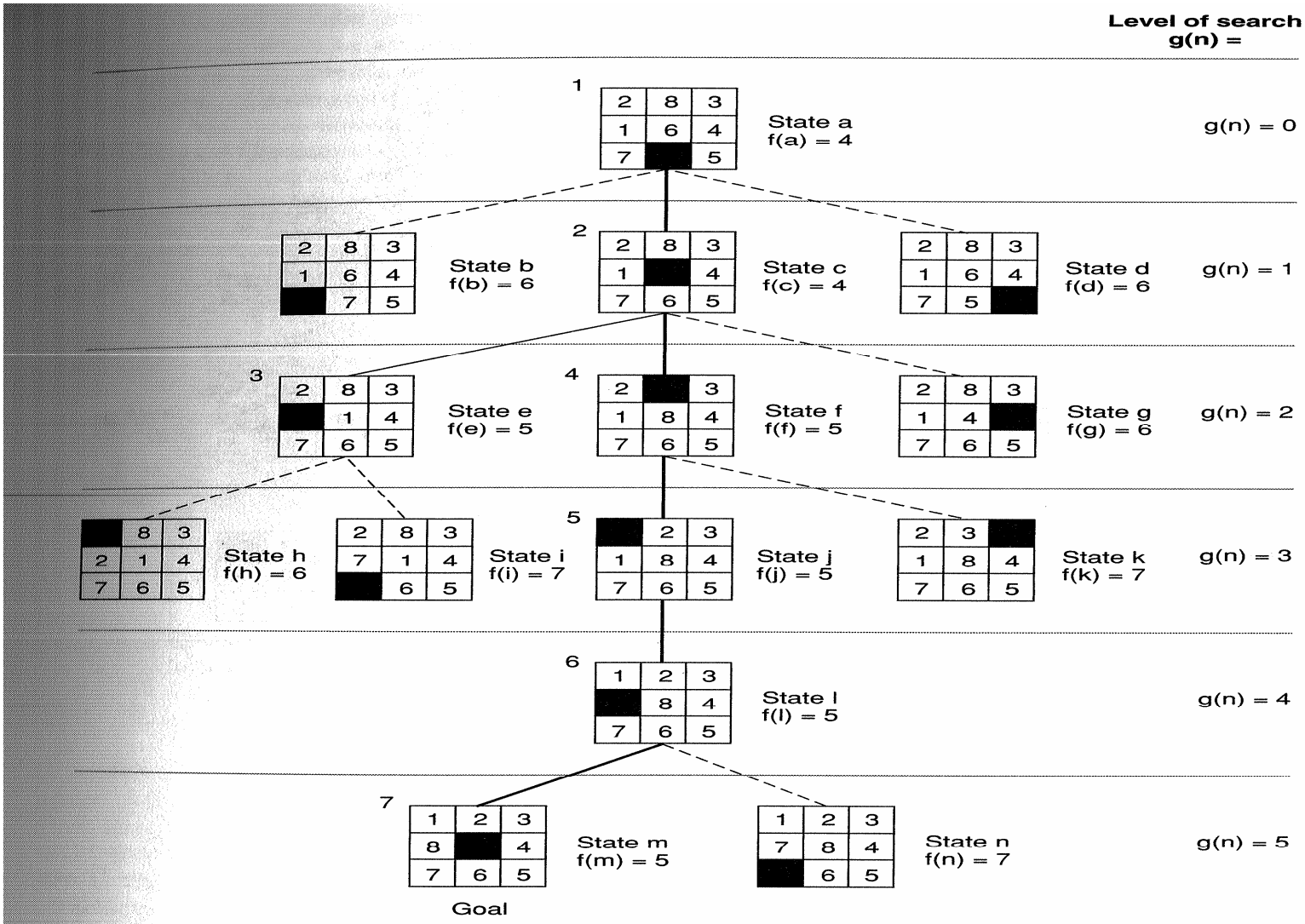


Figure 4.10 State space generated in heuristic search of the 8-puzzle graph.

Example … 8-puzzle (part vii)

1. open = [a4];
   closed = [ ]
2. open = [c4, b6, d6];
   closed = [a4]
3. open = [e5, f5, b6, d6, g6];
   closed = [a4, c4]
4. open = [f5, h6, b6, d6, g6, l7];
   closed = [a4, c4, e5]
5. open = [j5, h6, b6, d6, g6, k7, l7];
   closed = [a4, c4, e5, f5]
6. open = [l5, h6, b6, d6, g6, k7, l7];
   closed = [a4, c4, e5, f5, j5]
7. open = [m5, h6, b6, d6, g6, n7, k7, l7];
   closed = [a4, c4, e5, f5, j5, l5]
8. success, m = goal!

- Implementation using the best-first Search algorithm. These are the open And close lists.
- Note how OPEN is sorted according to the heuristic used.
- Hence with this algorithm one can change the heuristic and see what happens.!!

Two Player Games – MiniMax Algorithm !!

- Two-person games are more complicated than simple puzzles because of the existence of a 'hostile' and essentially unpredictable opponent !!
- Here we describe the MiniMax algorithm (with alpha-beta pruning) and implement two games, namely, a variant of *nim* and *tic-tac-toe*.
- *Nim* game state space can be exhaustively searched.

Two Player Games – Nim

- To play this game, a number of tokens are placed on a table between the two opponents; at each move, the player must divide a pile of tokens into two nonempty piles of different sizes.
- Thus, 6 tokens may be divided into piles of 5/1, 4/2 but not 3/3.
- The player who can no longer make a move loses the game.
- Next slide illustrates the search space with 7 tokens

Two Player Games – Nim Search Space

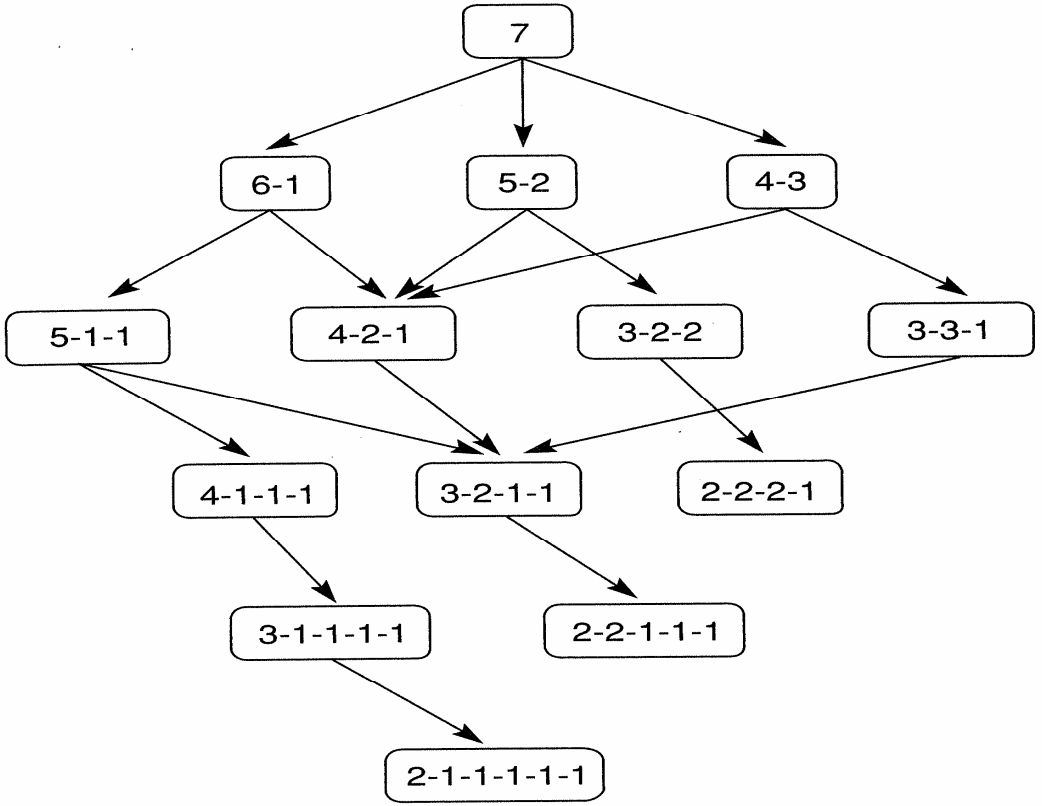


Figure 4.13 State space for a variant of nim. Each state partitions the seven matches into one or more piles.

Two Player Games – Generic MiniMax

- We need to account for the actions of the opponent.
- Assuming opponent uses same knowledge of the state space and applies that knowledge in a consistent way, we can predict the opponent's behaviour.
- MiniMax searches the game space under this assumption.
- Opponents are referred to as MIN and MAX
- MIN attempts to MINimize MAX's score while MAX tries to win by MAXimizing his/her advantage.

Two Player Games – MiniMax Algorithm

- Label each level in the search space according to whose move it is at that point in the game, MIN or MAX,
- Each leaf node is given a value of 1 or 0, depending on whether it is a win for MAX or for MIN.
- Minimax propagates these values up the graph through successive parent nodes according to these rules:
  - If parent state is a MAX node, give it the maximum value among its children
  - If parent state is a MIN node, give it the minimum value of its children

Two Player Games – MiniMax Nim

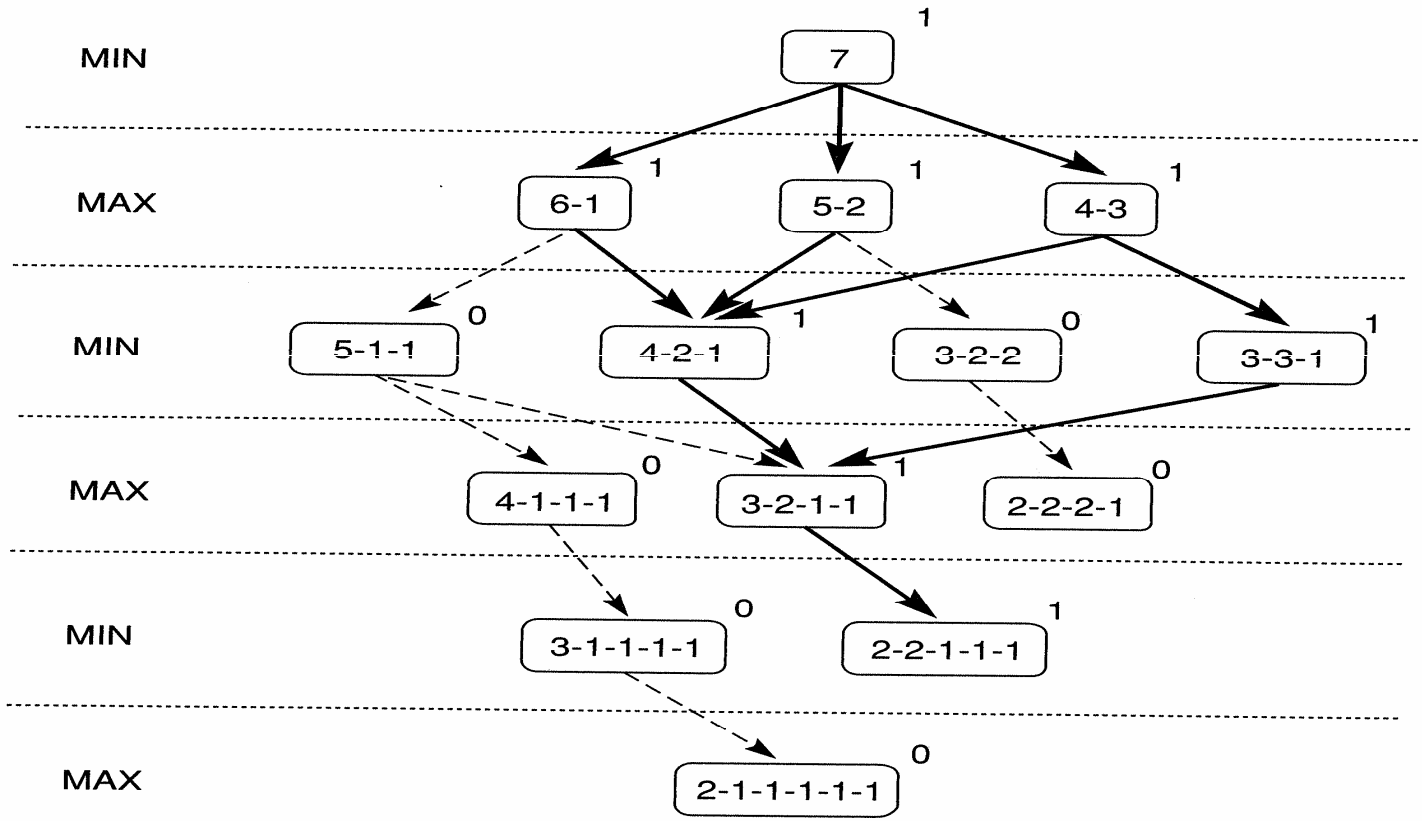
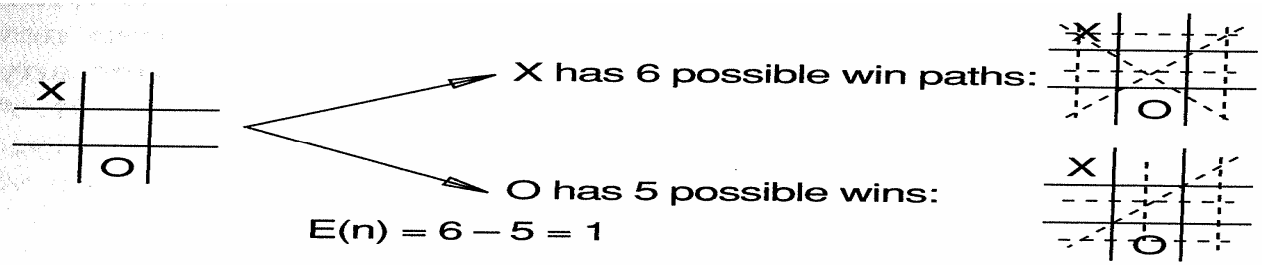


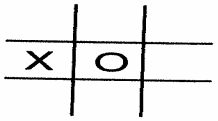
Figure 4.14 Exhaustive minimax for the game of nim. Bold lines indicate forced win for MAX. Each node is marked with its derived value (0 or 1) under minimax.

Minimaxing to a fixed ply depth

- With more complex problems (games) we cannot expand the state space graph out to the leaf nodes.
- Since the leaves of this subgraph are not final states of the game, it is not possible to give them values that reflect a win or a loss (1 or 0)
- We have to use a heuristic function !!
- Therefore we are actually computing the best state that can be reached in n moves (according to a particular heuristic)
- These values are propagated back to the root in a similar fashion to what we've seen with *nim*

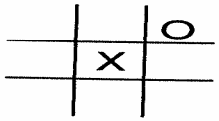
Minimaxing to a fixed ply depth – Tic Tac Toe





X has 4 possible win paths;
O has 6 possible wins

E(n) = 4 − 6 = −2



X has 5 possible win paths;
O has 4 possible wins

E(n) = 5 − 4 = 1

Heuristic is E(n) = M(n) − O(n)
where M(n) is the total of My possible winning lines
O(n) is total of Opponent's possible winning lines
E(n) is the total Evaluation for state n

Figure 4.16 Heuristic measuring conflict applied to states of tic-tac-toe.

Minimaxing to a fixed ply depth – Tic Tac Toe

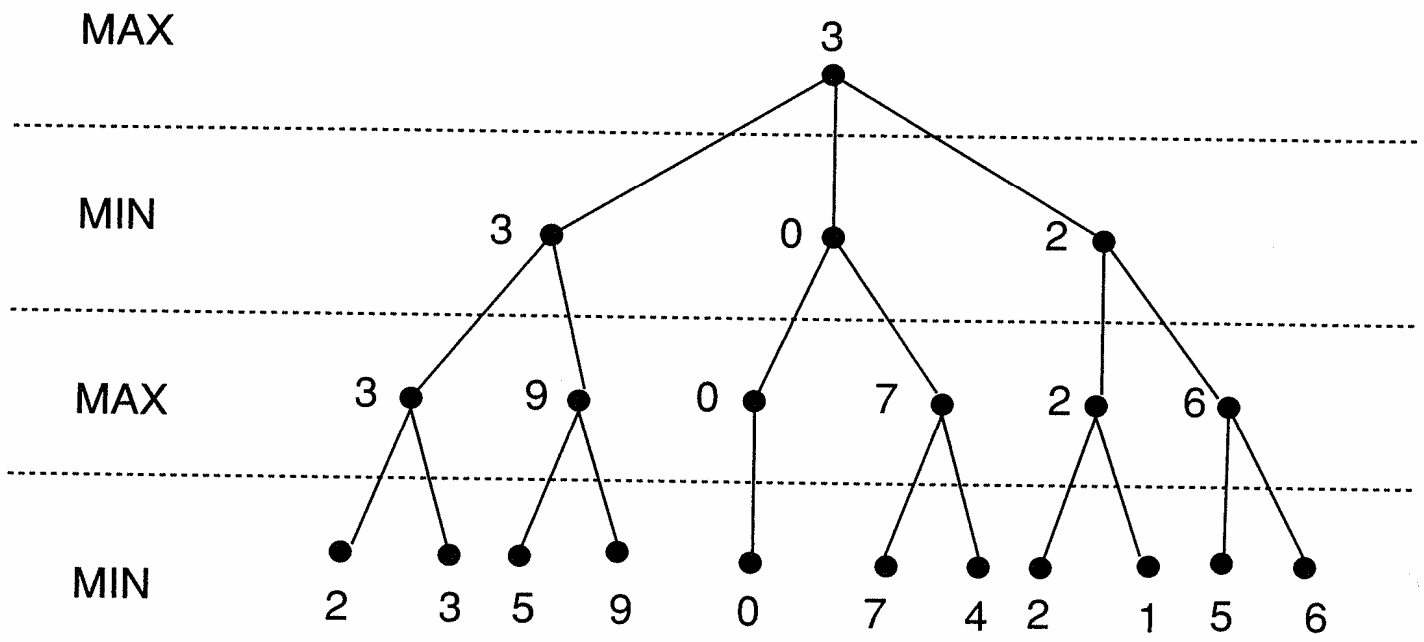


Figure 4.15 Minimax to a hypothetical state space. Leaf states show heuristic values; internal states show backed-up values.