4 Heuristic Search

Additional references for the slides:

Russell and Norvig’s AI book (2003).

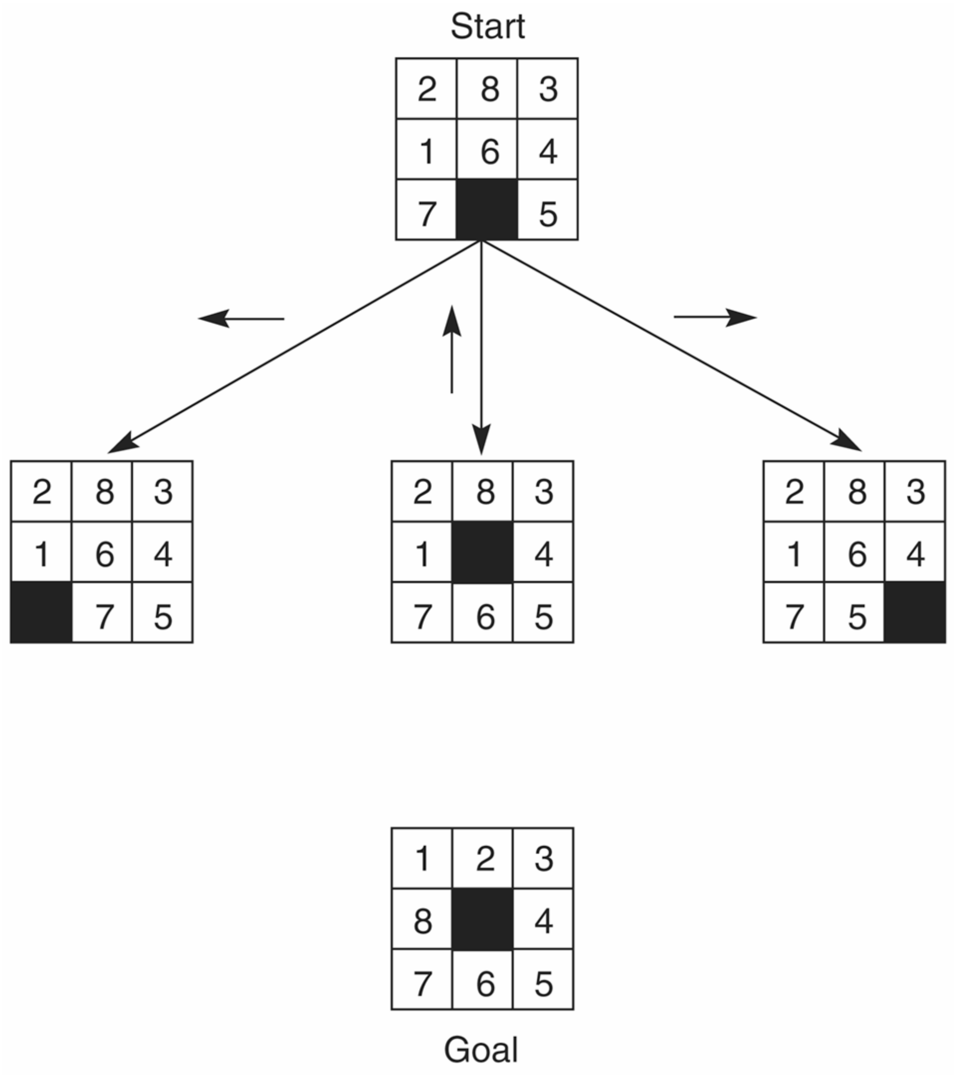
Robert Wilensky’s CS188 slides:
www.cs.berkeley.edu/~wilensky/cs188/lectures/index.html

Tim Huang’s slides for the game of Go.

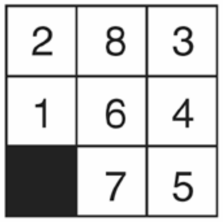
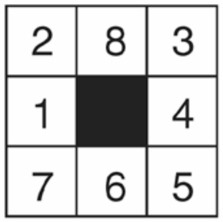
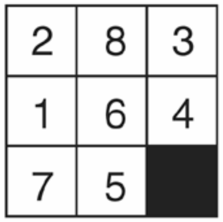
Chapter Objectives

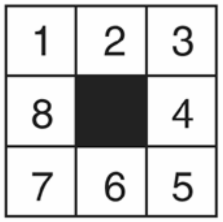
- Learn the basics of heuristic search in a state space.
- Learn the basic properties of heuristics: admissability, monotonicity, informedness.
- Learn the basics of searching for two-person games: minimax algorithm and alpha-beta procedure.
- The agent model: Has a problem, searches for a solution, has some “heuristics” to speed up the search.

An 8-puzzle instance



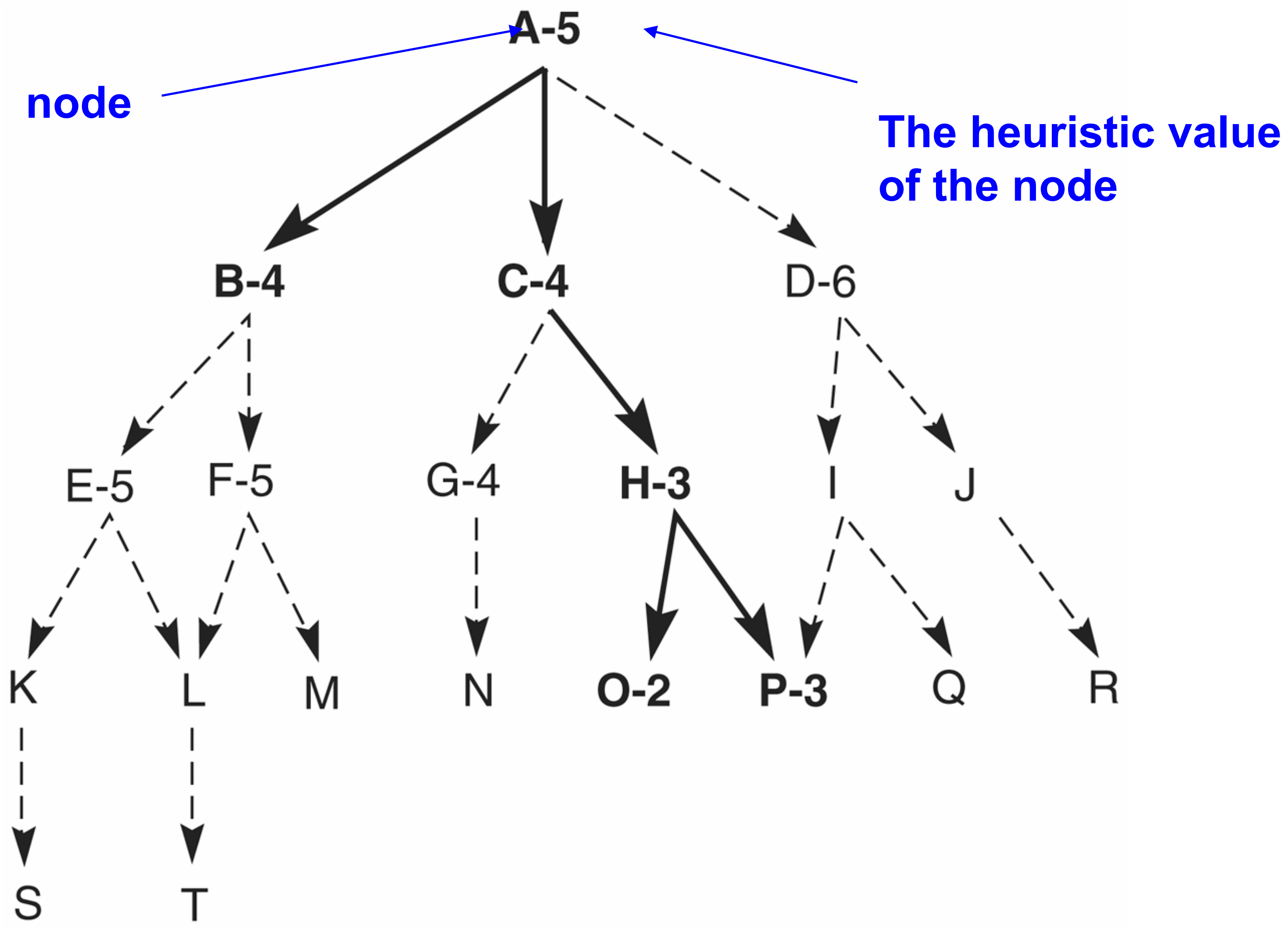
Three heuristics applied to states

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



Goal

Heuristic search of a hypothetical state space (Fig. 4.4)



Take the DFS algorithm

```
Function depth_first_search;

begin
  open := [Start];
  closed := [ ];
  while open ≠ [ ] do
    begin
      remove leftmost state from open, call it X;
      if X is a goal then return SUCCESS
        else begin
          generate children of X;
          put X on closed;
          discard remaining children of X if already on open or closed
          put remaining children on left end of open
        end
      end;
  return FAIL
end.
```

Add the children to OPEN with respect to their heuristic value

```
Function best_first_search;

begin
  open := [Start];
  closed := [ ];
  while open ≠ [ ] do
    begin
      remove leftmost state from open, call it X;
      if X is a goal then return SUCCESS
        else begin
          generate children of X;
          assign each child their heuristic value;
          put X on closed;
          (discard remaining children of X if already on open or closed)
          put remaining children on open
          sort open by heuristic merit (best leftmost)
        end
      end;
  return FAIL
end.
```

new

will be handled differently

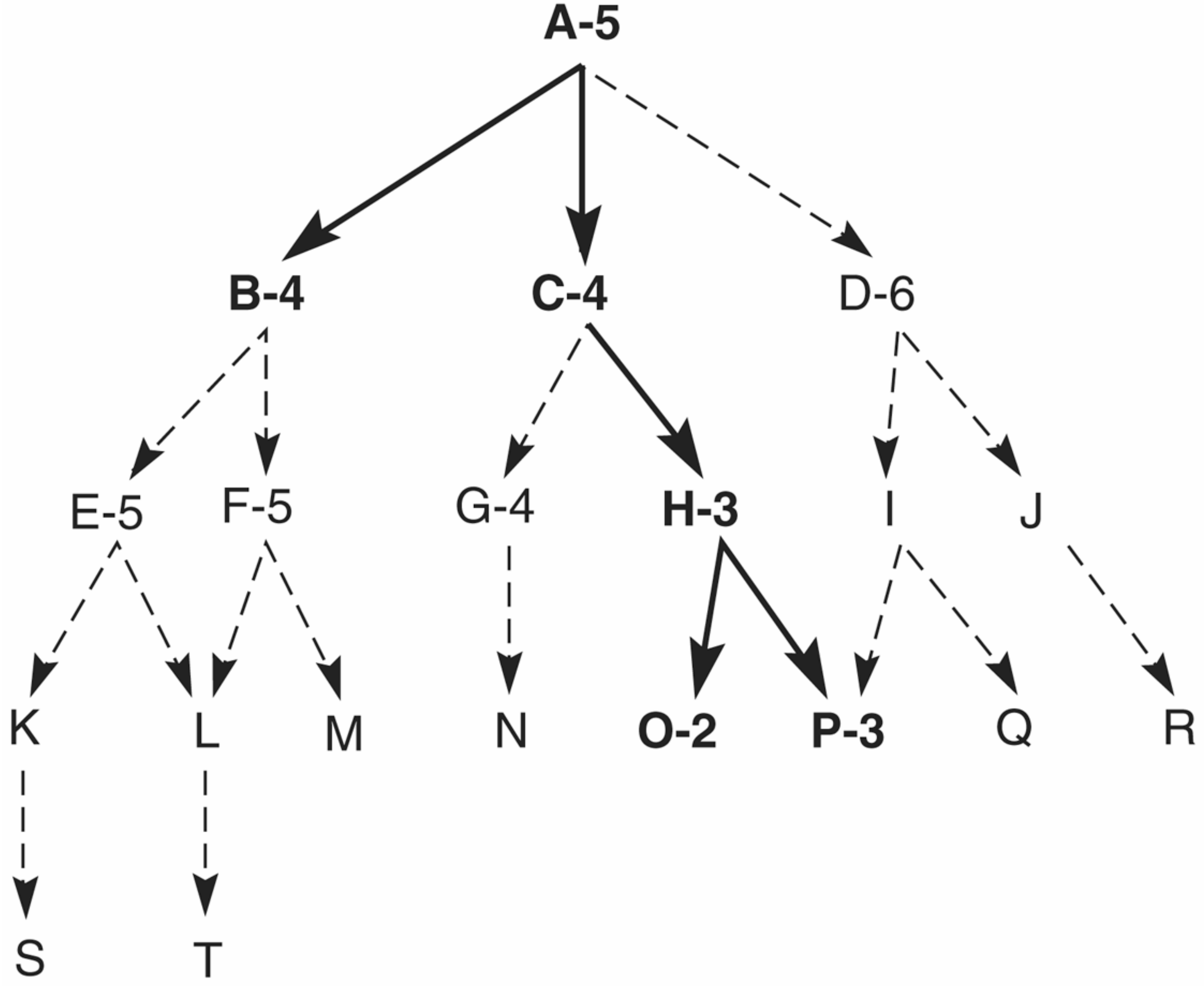
Now handle those nodes already on OPEN or CLOSED

```
...
        generate children of X;
         for each child of X do
         case
             the child is not on open or closed:
               begin
                 assign the child a heuristic value;
                 add the child to open
               end;
             the child is already on open:
               if the child was reached by a shorter path then
                  give the state on open the shorter path
             the child is already on closed:
               if the child was reached by a shorter path then
                begin
                   remove the child from closed;
                   add the child to open
                 end;
           end;
         put X on closed;
         re-order states on open by heuristic merit (best leftmost)
     end;
 ...
```

The full algorithm

```
Function best_first_search;
begin
 open := [Start];        closed := [ ];
 while open ≠ [ ] do
   begin
     remove leftmost state from open, call it X;
     if X is a goal then return SUCCESS
       else begin
         generate children of X;
         for each child of X do
         case
             the child is not on open or closed:
               begin
                 assign the child a heuristic value;
                 add the child to open
               end;
             the child is already on open:
               if the child was reached by a shorter path
               then give the state on open the shorter path
             the child is already on closed:
               if the child was reached by a shorter path then
                  begin
                     remove the child from closed;
                     add the child to open
                   end;
           end;
         put X on closed;
         re-order states on open by heuristic merit (best leftmost)
     end;
 return FAIL
end.
```

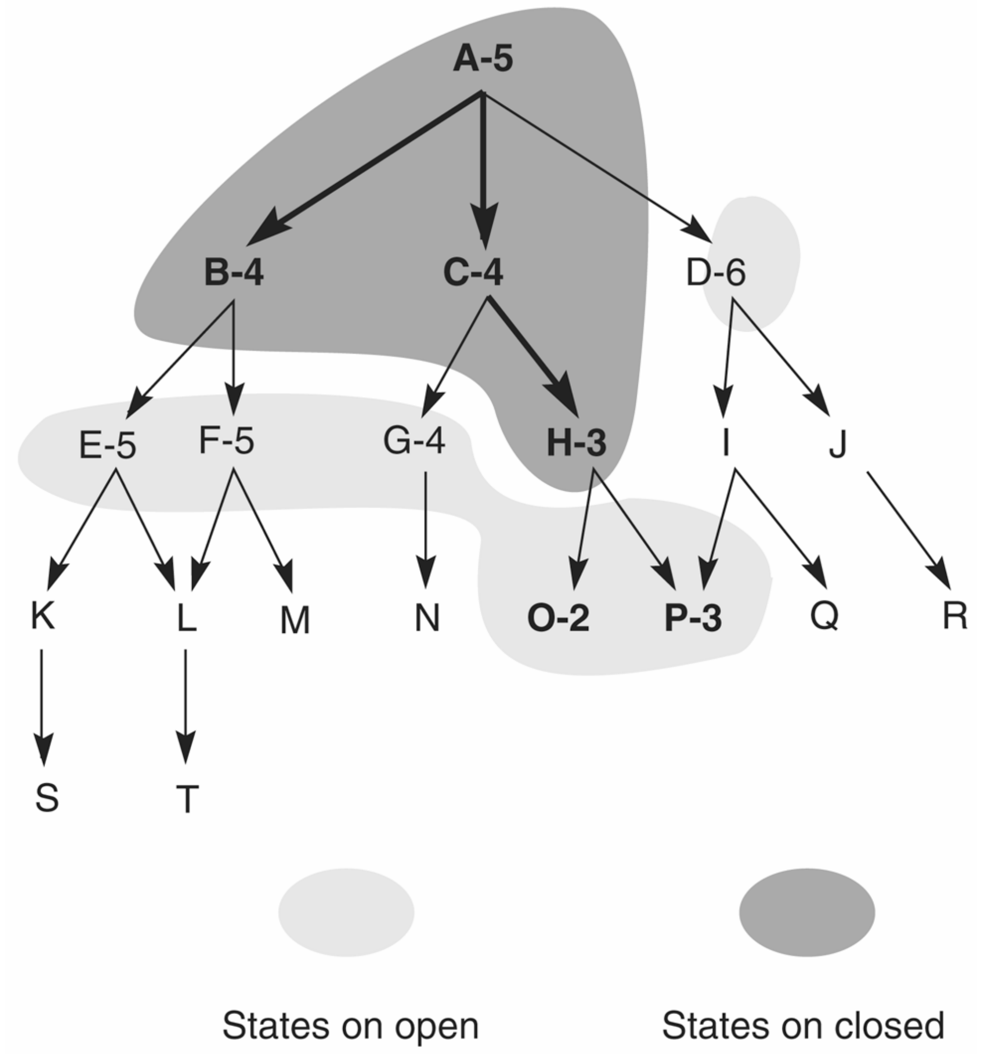
Heuristic search of a hypothetical state space



A trace of the execution of best_first_search for Fig. 4.4

1. **open = [A5]; closed = [ ]**
2. **evaluate A5; open = [B4,C4,D6]; closed = [A5]**
3. **evaluate B4; open = [C4,E5,F5,D6]; closed = [B4,A5]**
4. **evaluate C4; open = [H3,G4,E5,F5,D6]; closed = [C4,B4,A5]**
5. **evaluate H3; open = [O2,P3,G4,E5,F5,D6]; closed = [H3,C4,B4,A5]**
6. **evaluate O2; open = [P3,G4,E5,F5,D6]; closed = [O2,H3,C4,B4,A5]**
7. **evaluate P3; the solution is found!**

Heuristic search of a hypothetical state space with open and closed highlighted



What is in a “heuristic?”

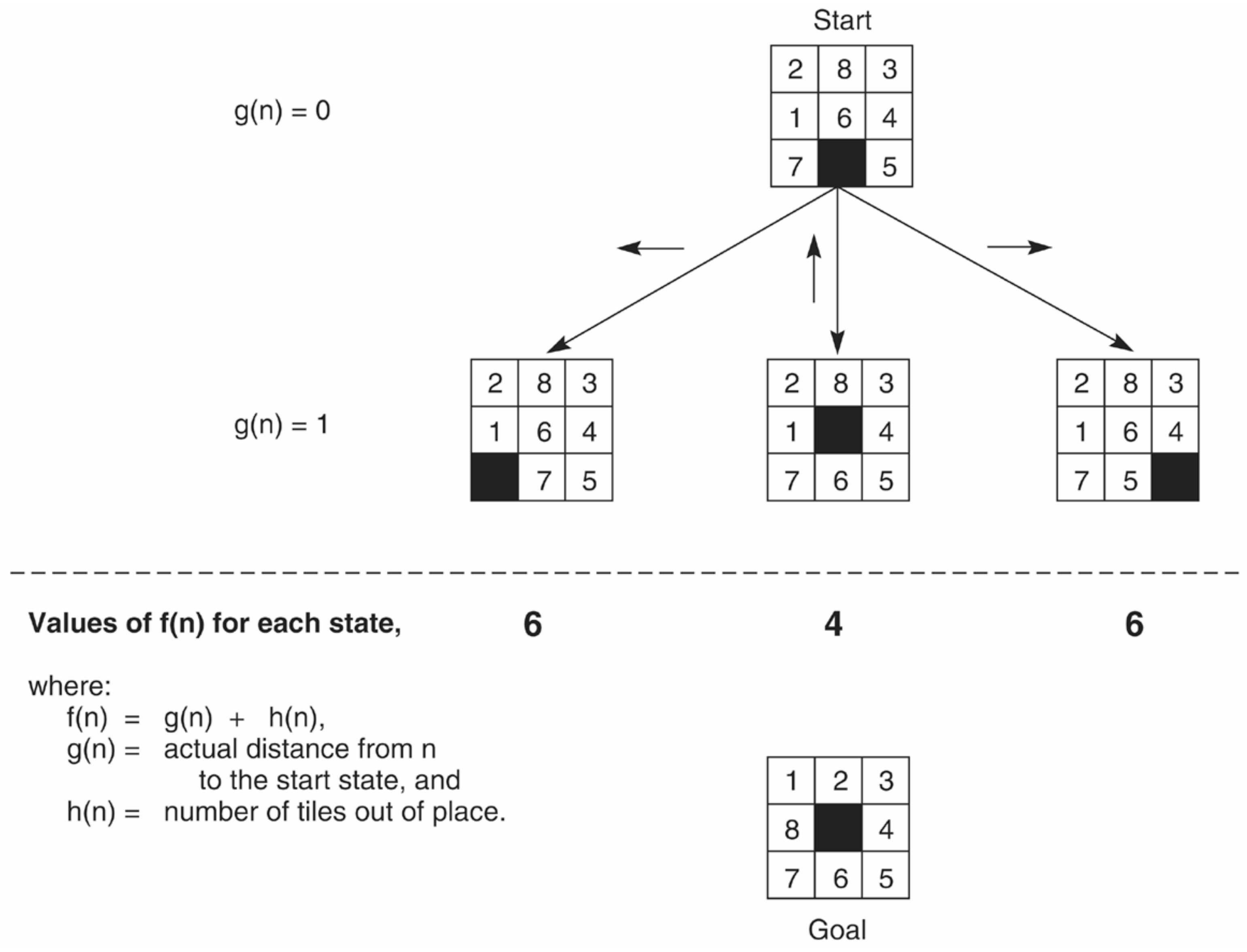
$$f(n) = g(n) + h(n)$$

The heuristic value of node n

The estimated cost of achieving the goal (from node n to the goal)

The actual cost of node n (from the root to n)

The heuristic f applied to states in the 8-puzzle



Values of f(n) for each state, **6** **4** **6**

where:

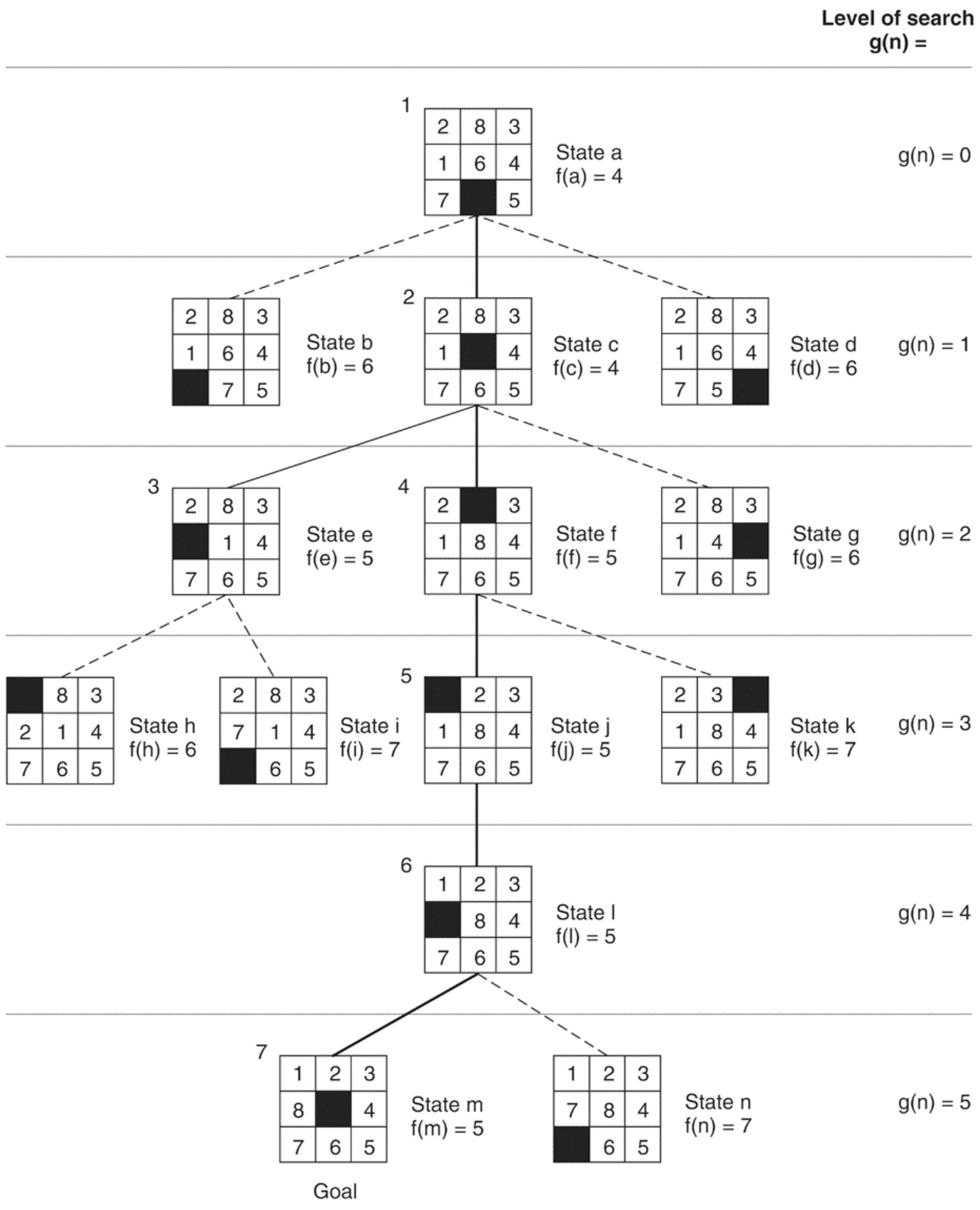
f(n) = g(n) + h(n),

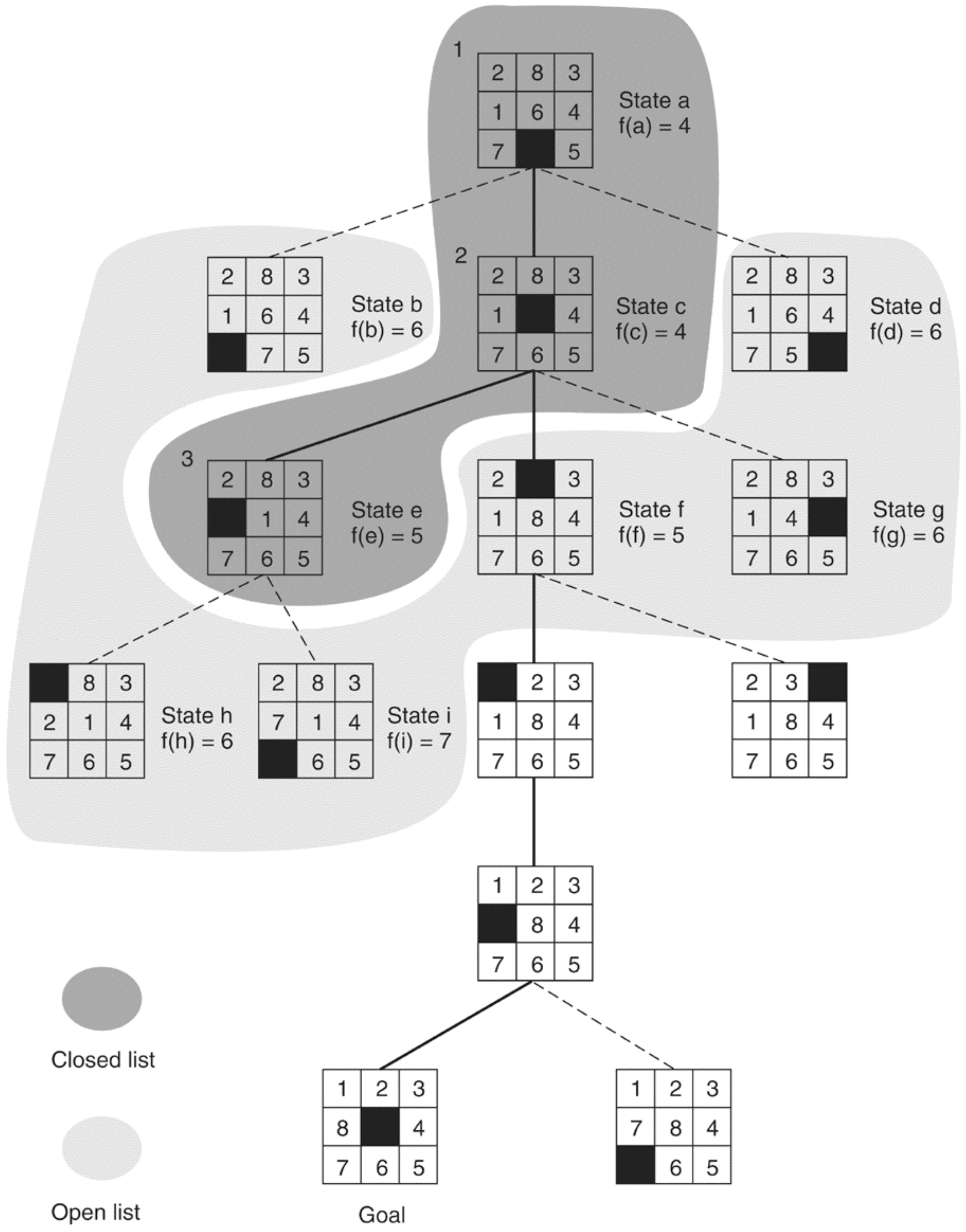
g(n) = actual distance from n to the start state, and

h(n) = number of tiles out of place.

The successive stages of OPEN and CLOSED

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!**


Level of search g(n) =
1
State a f(a) = 4
g(n) = 0
2
State b f(b) = 6
State c f(c) = 4
State d f(d) = 6
g(n) = 1
3
4
State e f(e) = 5
State f f(f) = 5
State g f(g) = 6
g(n) = 2
5
State h f(h) = 6
State i f(i) = 7
State j f(j) = 5
State k f(k) = 7
g(n) = 3
6
State l f(l) = 5
g(n) = 4
7
State m f(m) = 5
State n f(n) = 7
g(n) = 5
Goal



1
2 8 3
1 6 4
7 5
State a
f(a) = 4
2 8 3
1 6 4
7 5
State b
f(b) = 6
2
2 8 3
1 4
7 6 5
State c
f(c) = 4
2 8 3
1 6 4
7 5
State d
f(d) = 6
3
2 8 3
1 4
7 6 5
State e
f(e) = 5
2 3
1 8 4
7 6 5
State f
f(f) = 5
2 8 3
1 4
7 6 5
State g
f(g) = 6
8 3
2 1 4
7 6 5
State h
f(h) = 6
2 8 3
7 1 4
6 5
State i
f(i) = 7
2 3
1 8 4
7 6 5
2 3
1 8 4
7 6 5
1 2 3
8 4
7 6 5
1 2 3
8 4
7 6 5
Goal
1 2 3
7 8 4
6 5
Closed list
Open list

Algorithm A

Consider the evaluation function $f(n) = g(n) + h(n)$ where

n is any state encountered during the search
g(n) is the cost of n from the start state
h(n) is the heuristic estimate of the distance n to the goal

If this evaluation algorithm is used with the best_first_search algorithm of Section 4.1, the result is called *algorithm A*.

Algorithm A*

If the heuristic function used with algorithm A is *admissible*, the result is called *algorithm A** (pronounced A-star).

A heuristic is *admissible* if it never overestimates the cost to the goal.

The A* algorithm always finds the optimal solution path whenever a path from the start to a goal state exists (the proof is omitted, optimality is a consequence of admissability).

Monotonicity

A heuristic function h is *monotone* if

1. For all states n_i and n_J, where n_J is a descendant of n_i,

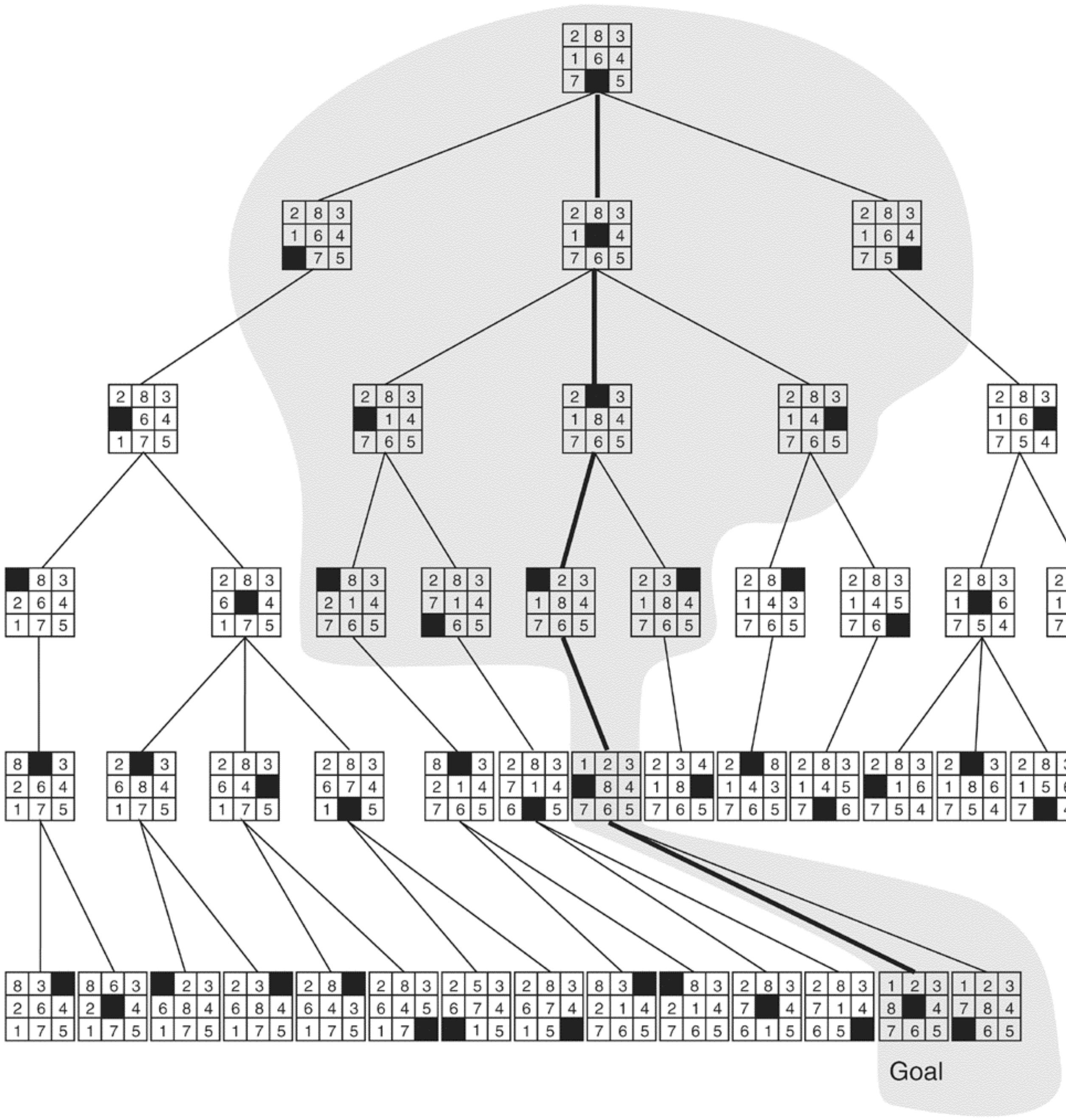
   $h(n_i) - h(n_J) \leq cost(n_i, n_J)$,

   where cost (n_i, n_J) is the actual cost (in number of moves) of going from state n_i to n_J.

2. The heuristic evaluation of the goal state is zero, or h(Goal) = 0.

Informedness

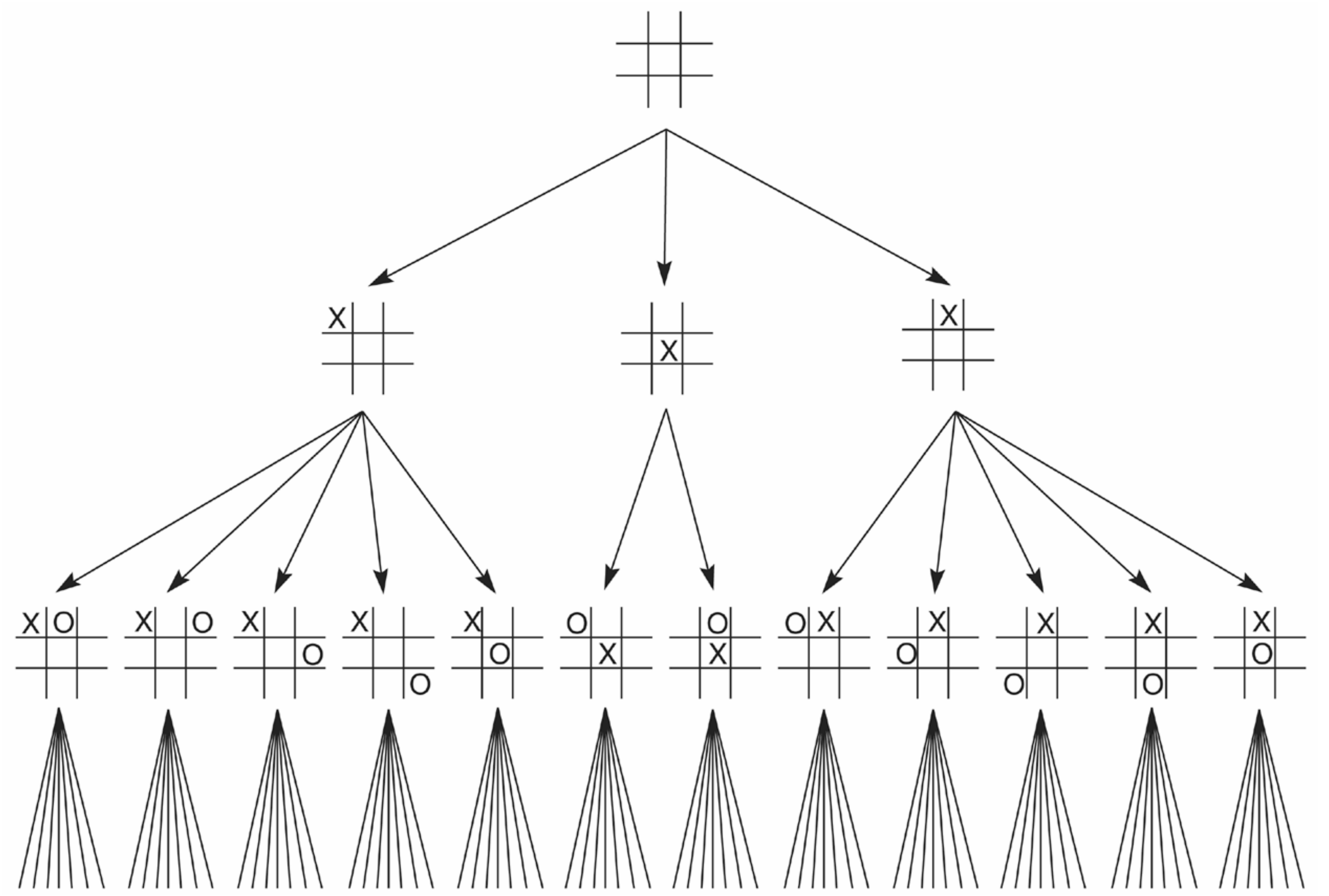
For two A* heuristics h_1 and h_2, if $h_1(n) \leq h_2(n)$, for all states n in the search space, heuristic h_2 is said to be more ***informed*** than h_1.


Goal

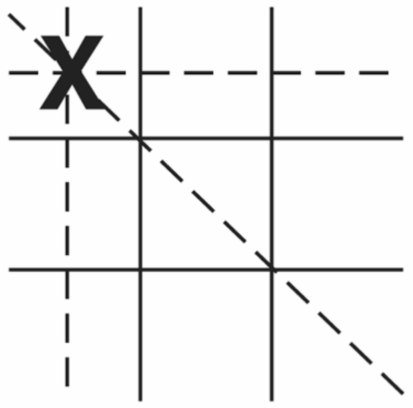
Game playing

Games have always been an important application area for heuristic algorithms. The games that we will look at in this course will be two-person board games such as Tic-tac-toe, Chess, or Go.

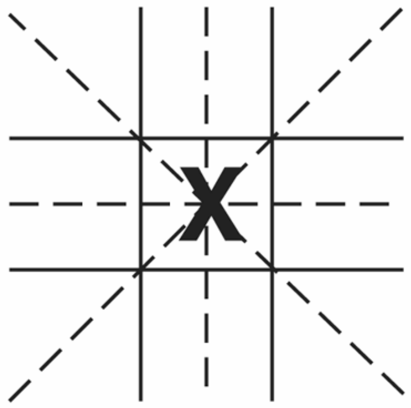
First three levels of tic-tac-toe state space reduced by symmetry



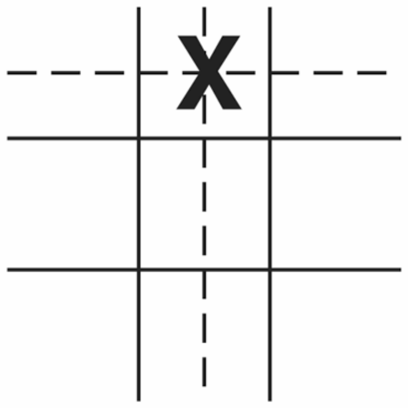
The “most wins” heuristic



Three wins through a corner square

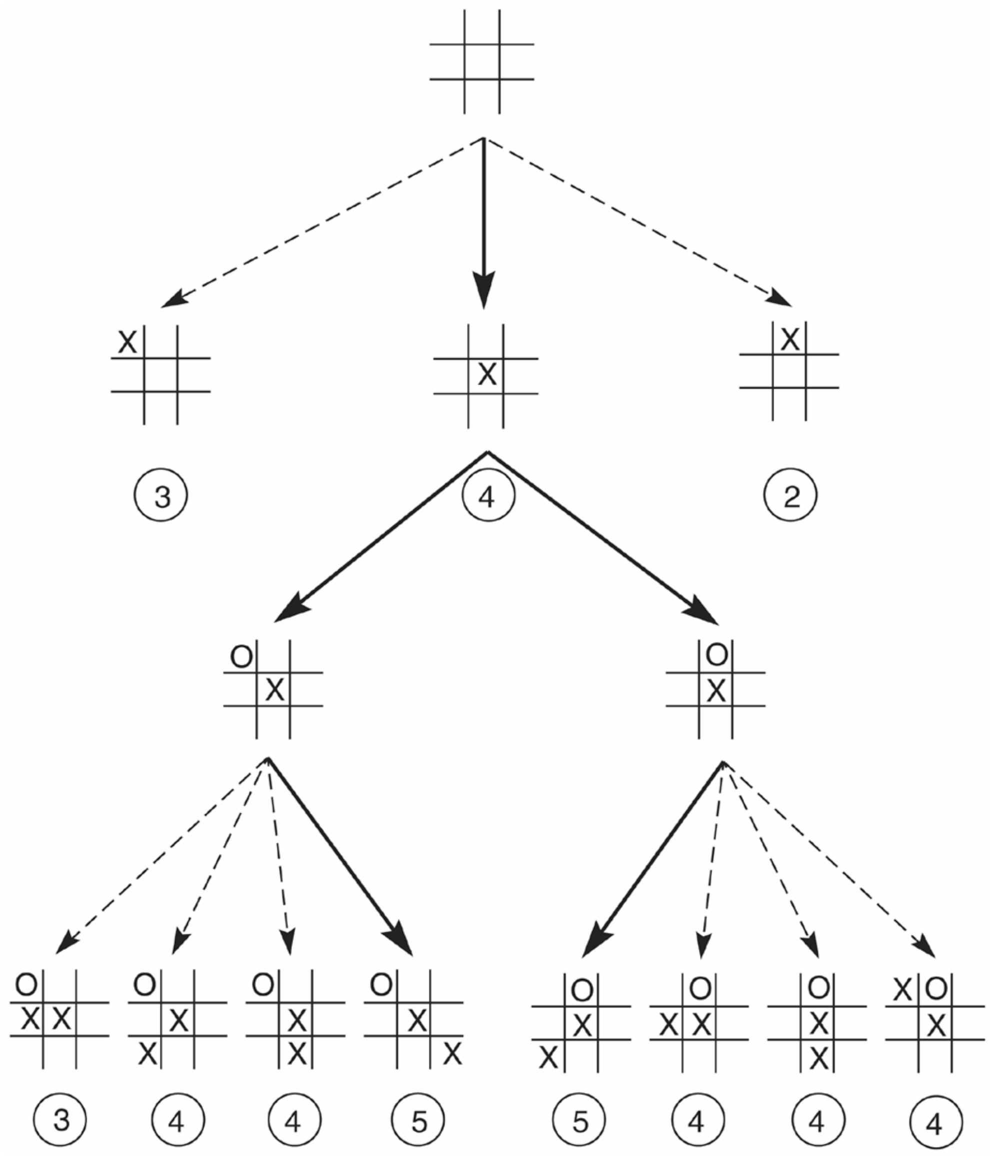


Four wins through the center square



Two wins through a side square

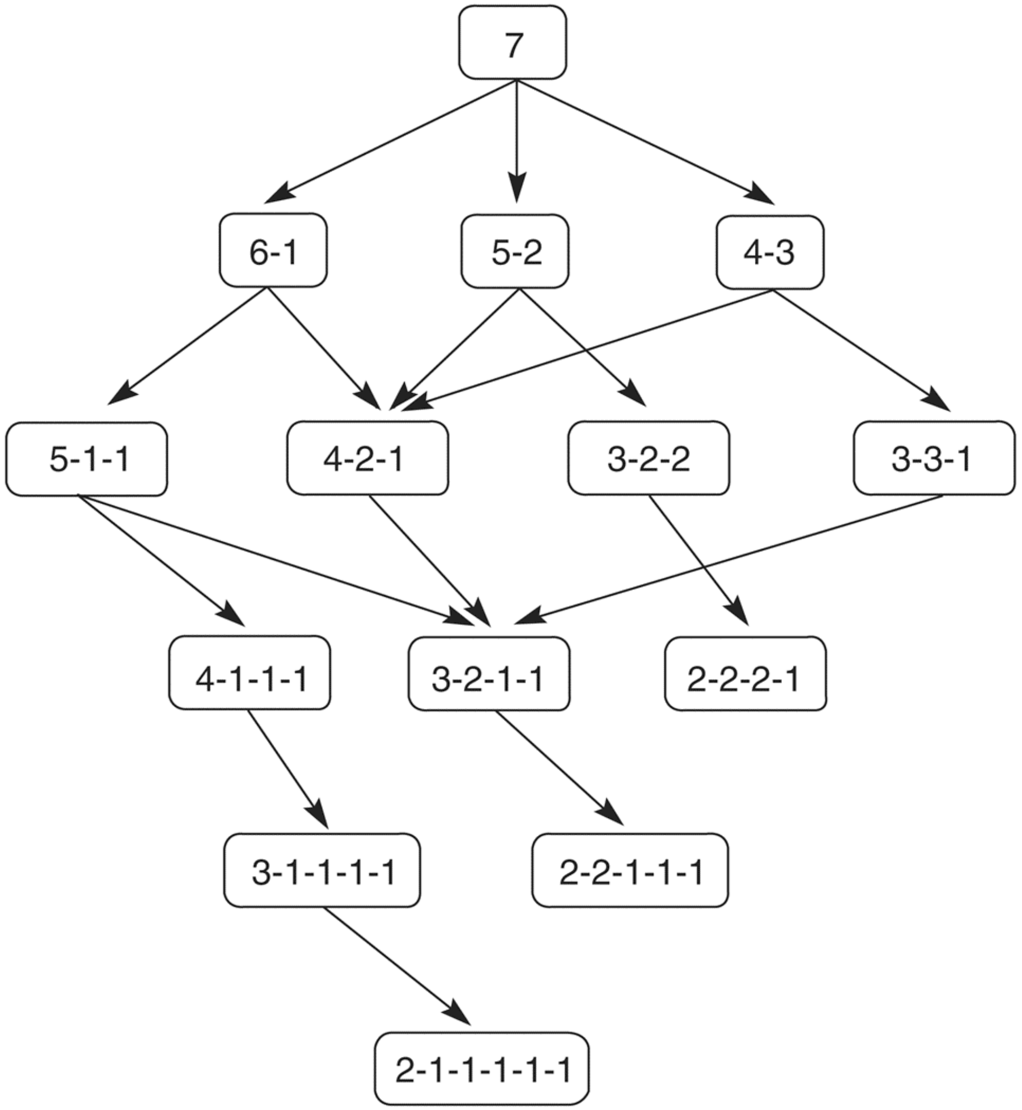
Heuristically reduced state space for tic-tac-toe



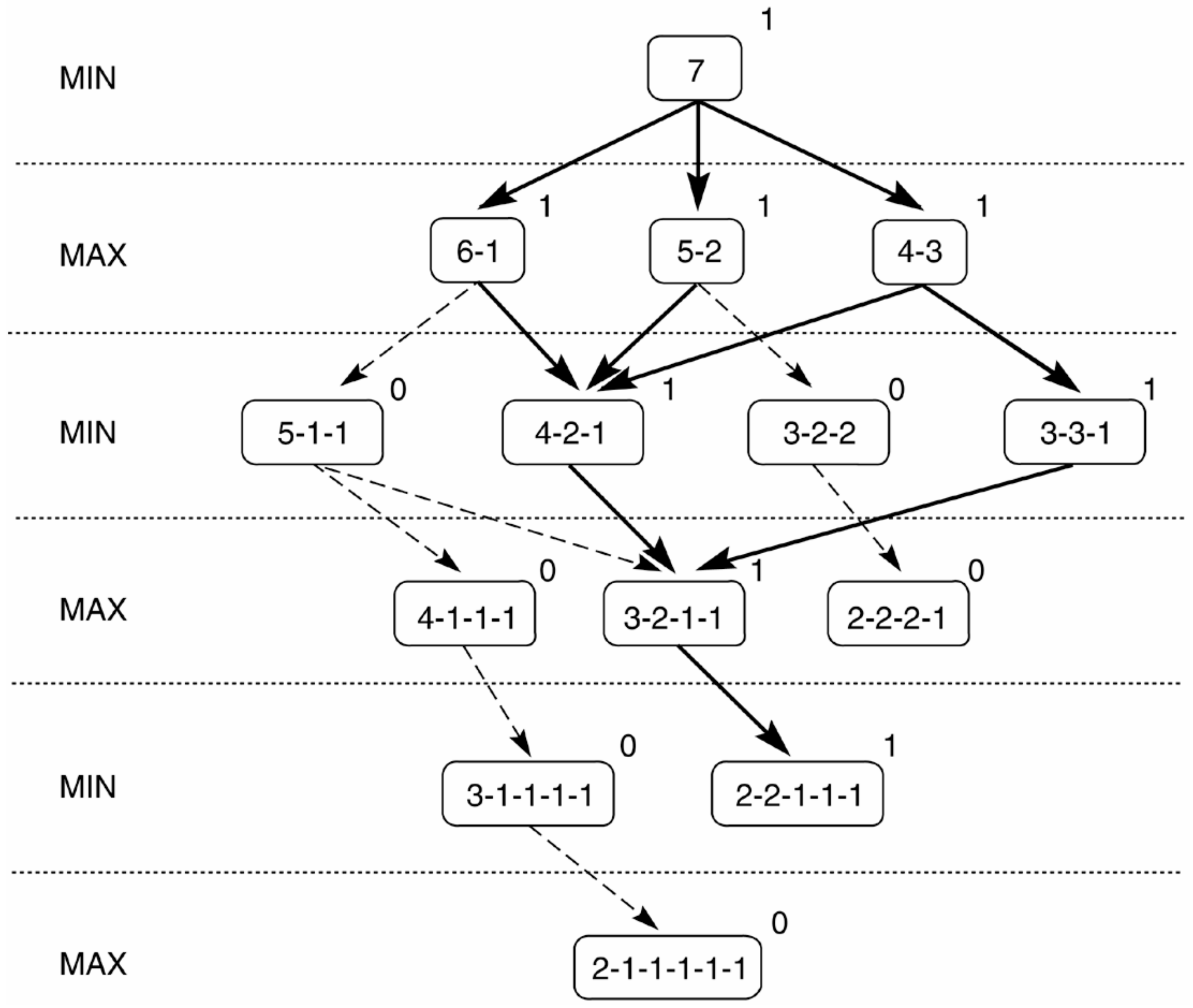
A variant of the game nim

- A number of tokens are placed on a table between the two opponents
- A move consists of dividing a pile of tokens into two nonempty piles of different sizes
- For example, 6 tokens can be divided into piles of 5 and 1 or 4 and 2, but not 3 and 3
- The first player who can no longer make a move loses the game
- For a reasonable number of tokens, the state space can be exhaustively searched

State space for a variant of nim



Exhaustive minimax for the game of nim



Two people games

- One of the earliest AI applications
- Several programs that compete with the best human players:
  - Checkers: beat the human world champion
  - Chess: beat the human world champion (in 2002 & 2003)
  - Backgammon: at the level of the top handful of humans
  - Go: no competitive programs
  - Othello: good programs
  - Hex: good programs

Search techniques for 2-person games

- The search tree is slightly different: It is a *two-ply tree* where levels alternate between players
- Canonically, the first level is “us” or the player whom we want to win.
- Each final position is assigned a payoff:
  - win (say, 1)
  - lose (say, -1)
  - draw (say, 0)
- We would like to maximize the payoff for the first player, hence the names MAX & MINIMAX

The search algorithm

- The root of the tree is the current board position, it is MAX's turn to play

- MAX generates the tree as much as it can, and picks the best move assuming that Min will also choose the moves for herself.

- This is the *Minimax algorithm* which was invented by Von Neumann and Morgenstern in 1944, as part of game theory.

- The same problem with other search trees: the tree grows very quickly, exhaustive search is usually impossible.

Special technique 1

- MAX generates the full search tree (up to the leaves or terminal nodes or final game positions) and chooses the best one:
  win or tie

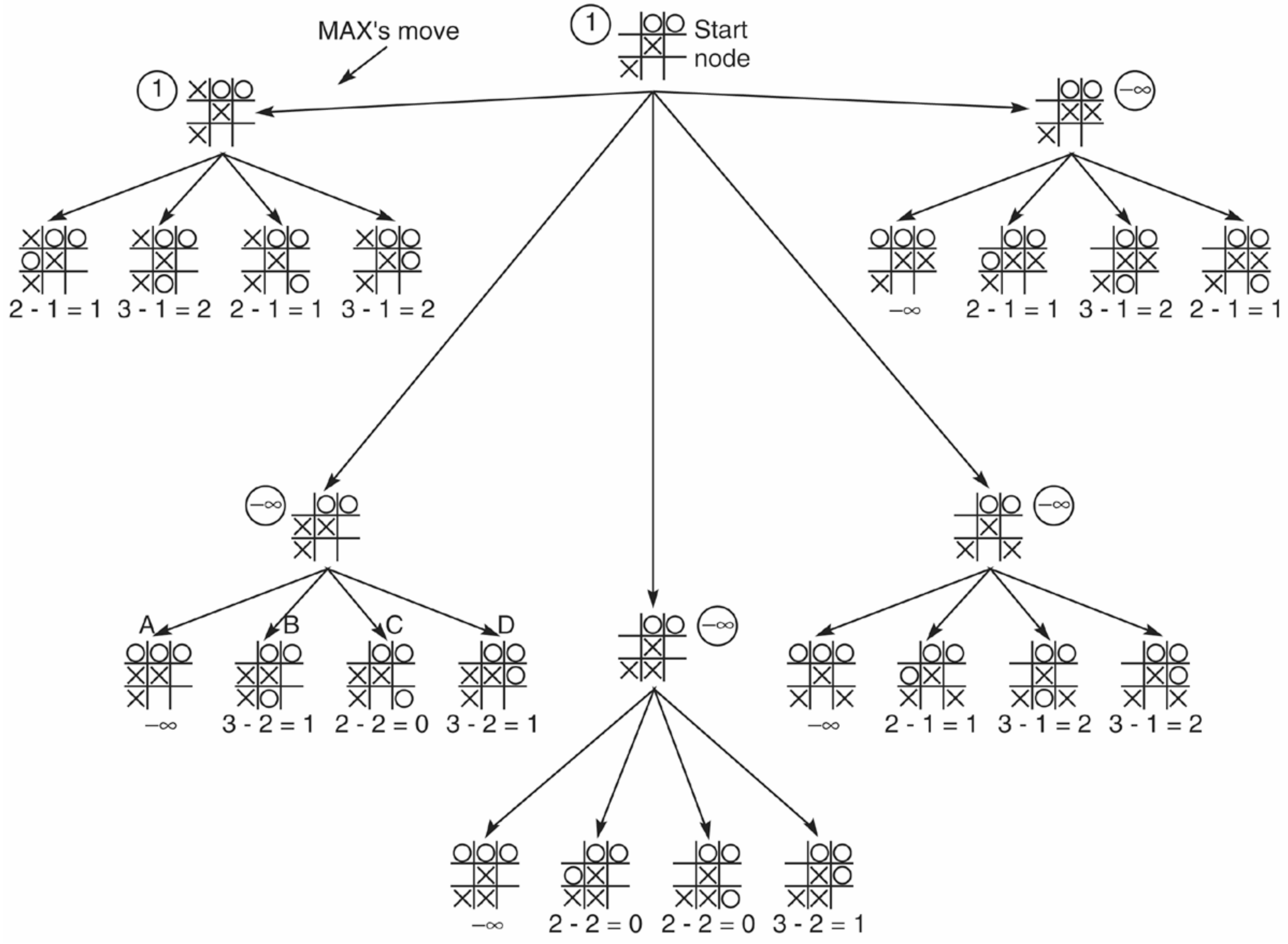
- To choose the best move, values are propogated upward from the leaves:
  - MAX chooses the maximum
  - MIN chooses the minimum

- This assumes that the full tree is not prohibitively big

- It also assumes that the final positions are easily identifiable

- We can make these assumptions for now, so let’s look at an example

Two-ply minimax applied to X's move near the end of the game (Nilsson, 1971)



Special technique 2

- Notice that the tree was not generated to full depth in the previous example

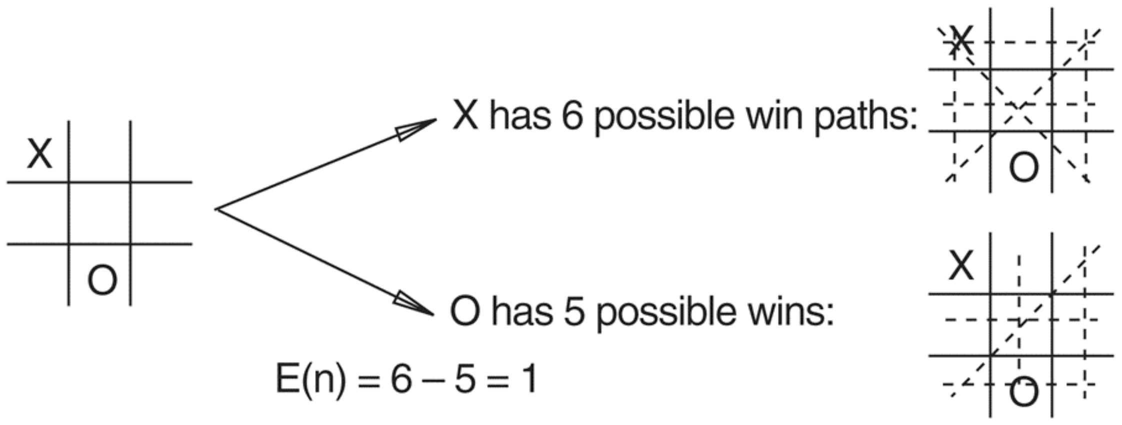
- When time or space is tight, we can't search exhaustively so we need to implement a cut-off point and simply not expand the tree below the nodes who are at the cut-off level.

- But now the leaf nodes are not final positions but we still need to evaluate them:

  use heuristics

- We can use a variant of the "most wins" heuristic

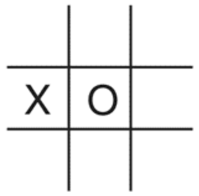
Heuristic measuring conflict



X has 6 possible win paths:

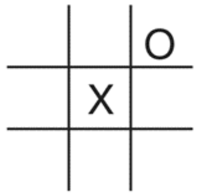
O has 5 possible wins:

$E(n) = 6 - 5 = 1$



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

$E(n) = 4 - 6 = -2$



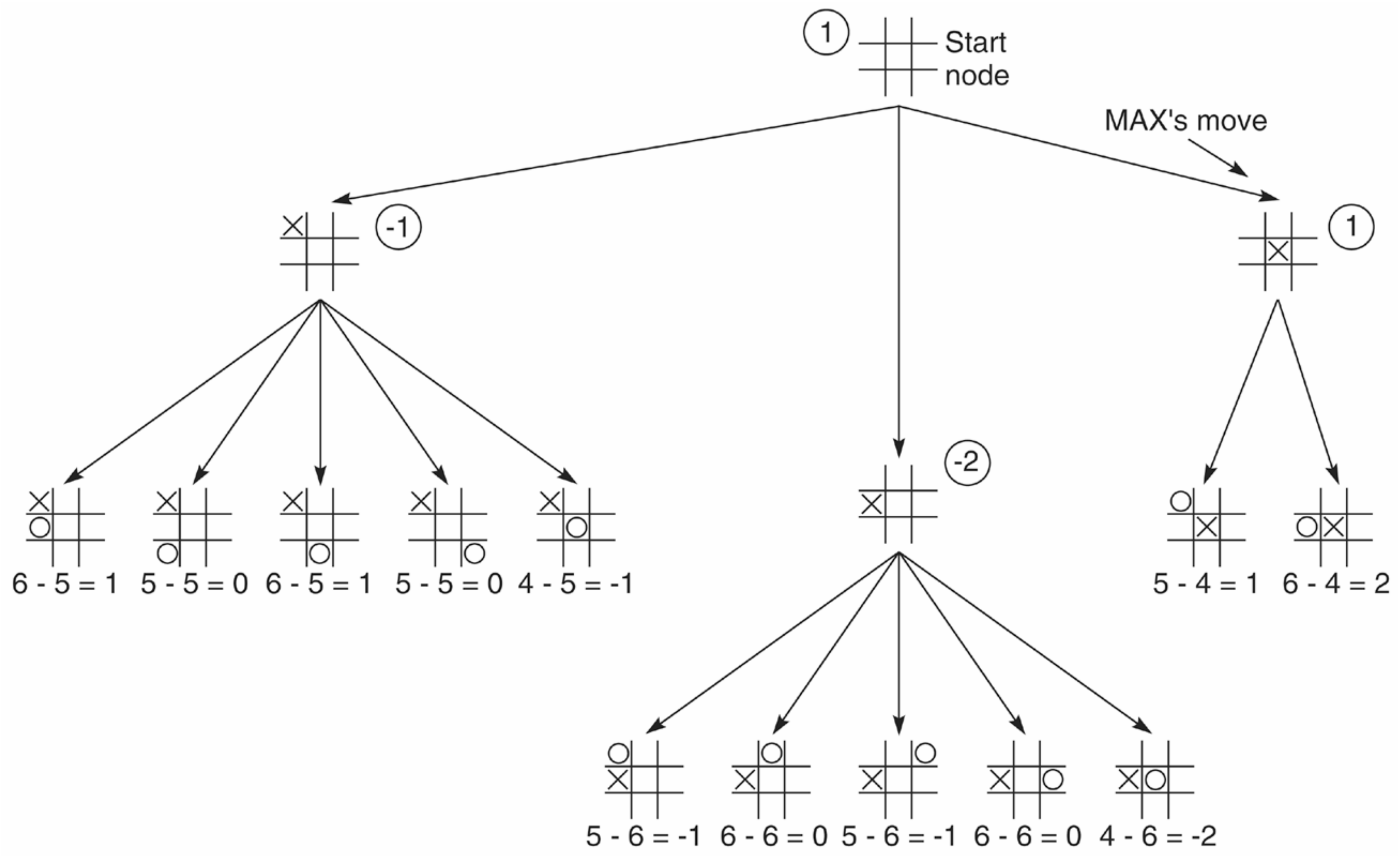
X has 5 possible win paths;
O has 4 possible wins

$E(n) = 5 - 4 = 1$

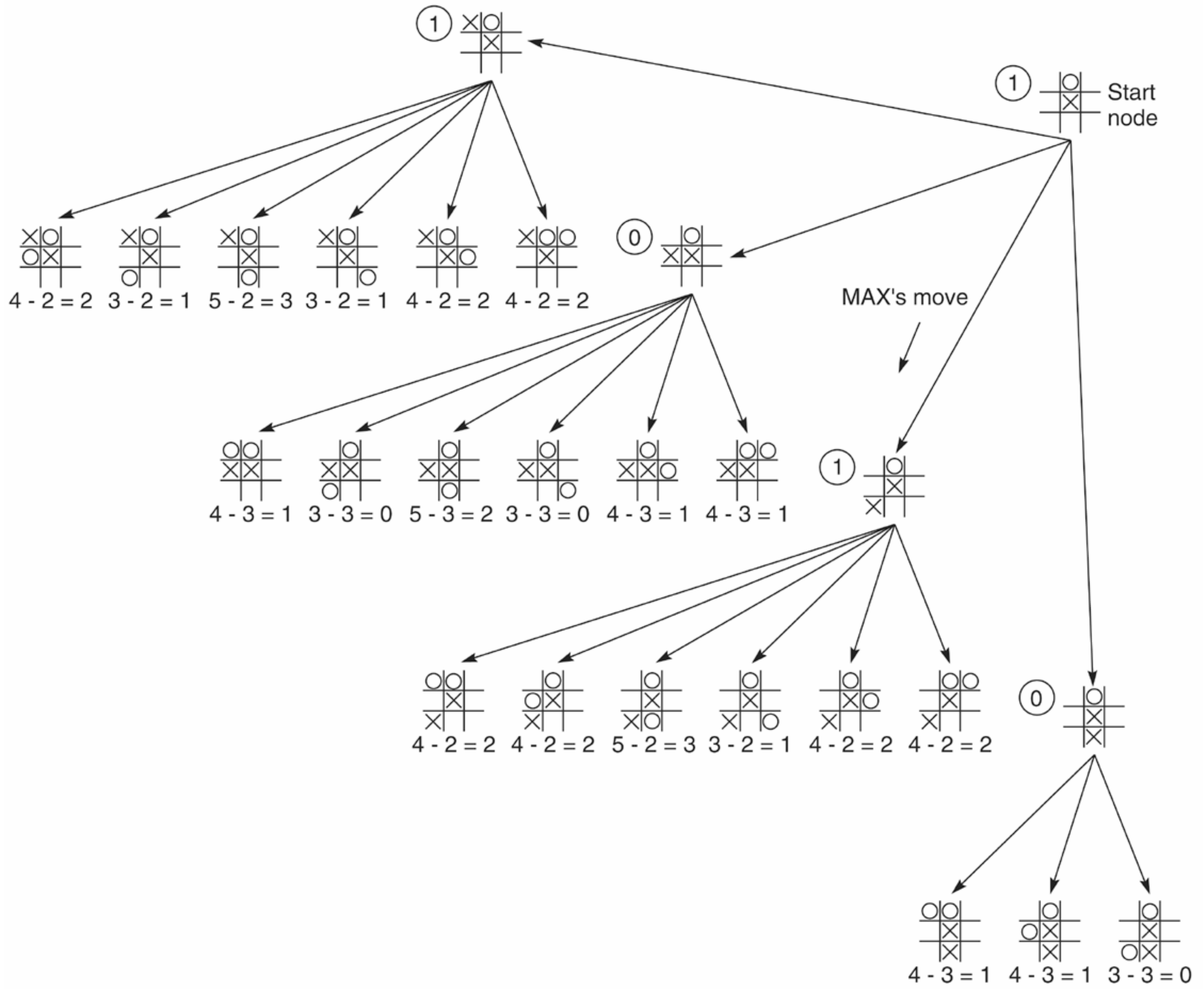
Calculation of the heuristic

- E(n) = M(n) – O(n) where
  - M(n) is the total of My (MAX) possible winning lines
  - O(n) is the total of Opponent’s (MIN) possible winning lines
  - E(n) is the total evaluation for state n
- Take another look at the previous example
- Also look at the next two examples which use a cut-off level (a.k.a. *search horizon*) of 2 levels

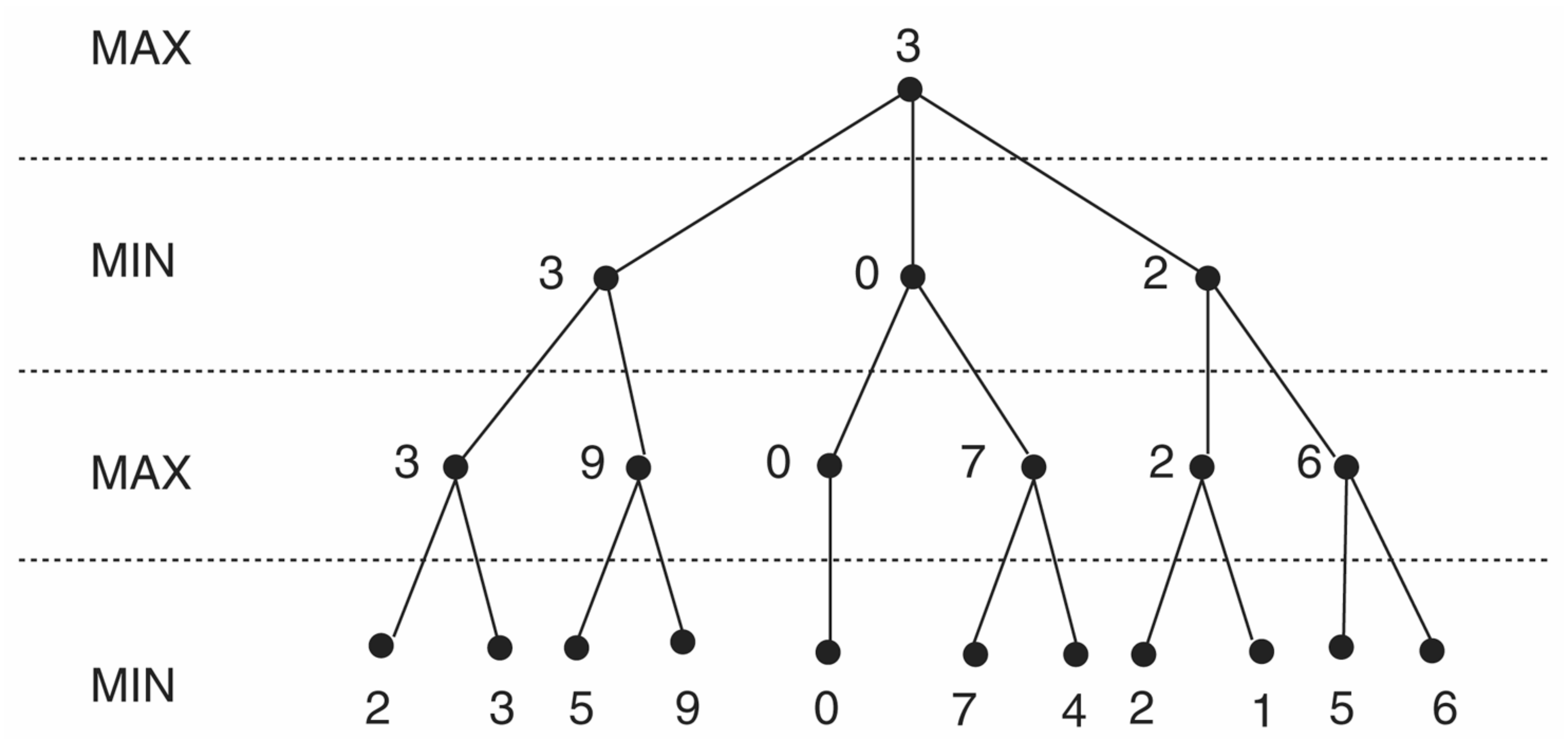
Two-ply minimax applied to the opening move of tic-tac-toe (Nilsson, 1971)



Two-ply minimax and one of two possible second MAX moves (Nilsson, 1971)



Minimax applied to a hypothetical state space (Fig. 4.15)



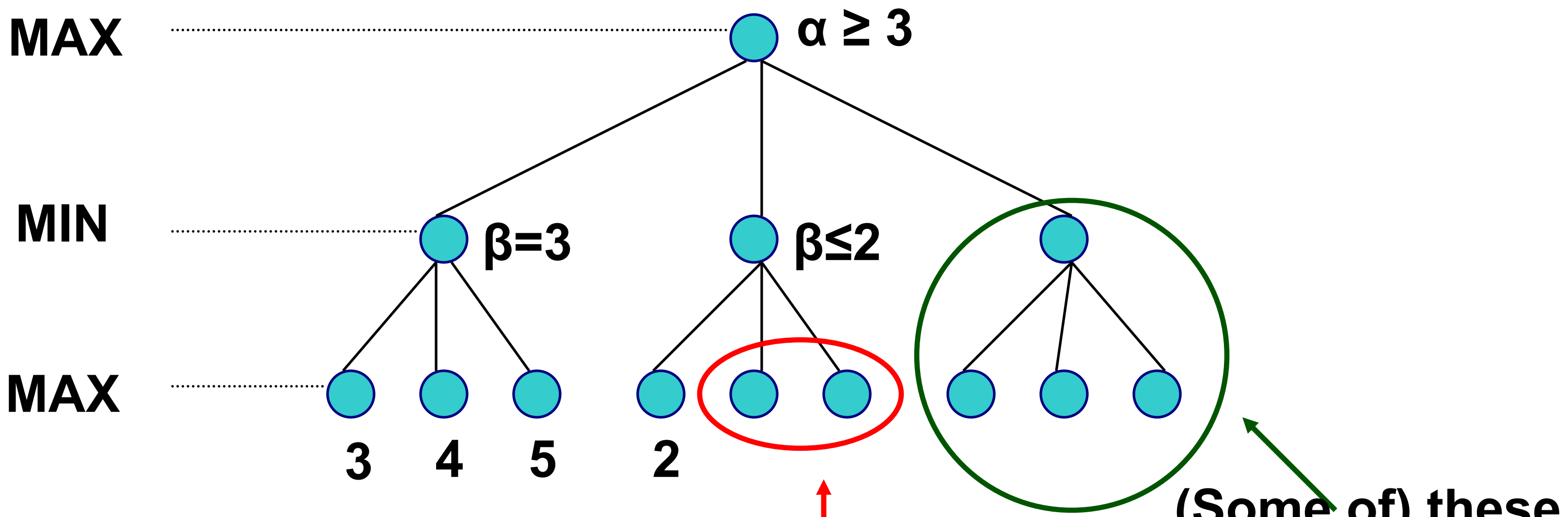
Special technique 3

- Use alpha-beta pruning
- Basic idea: if a portion of the tree is obviously good (bad) don't explore further to see how terrific (awful) it is
- Remember that the values are propagated upward. Highest value is selected at MAX's level, lowest value is selected at MIN's level
- Call the values at MAX levels *α values*, and the values at MIN levels *β values*

The rules

- Search can be stopped below any MIN node having a beta value less than or equal to the alpha value of any of its MAX ancestors
- Search can be stopped below any MAX node having an alpha value greater than or equal to the beta value of any of its MIN node ancestors

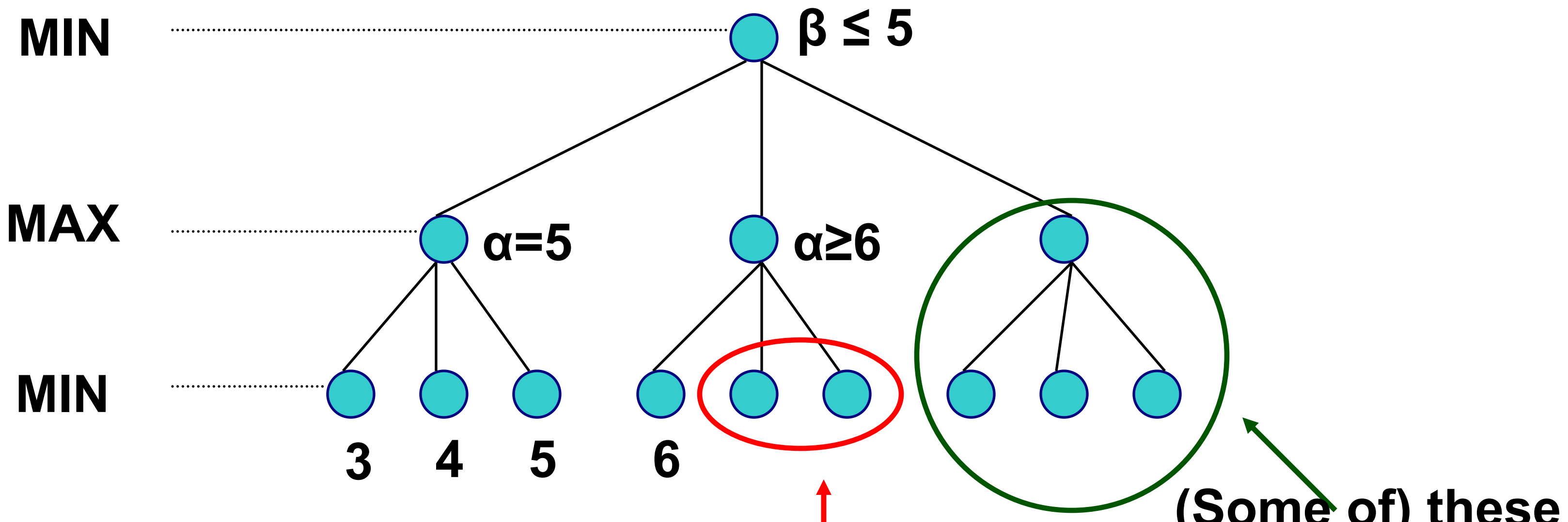
Example with MAX



(Some of) these still need to be looked at

As soon as the node with value 2 is generated, we know that the beta value will be less than 3, we don't need to generate these nodes (and the subtree below them)

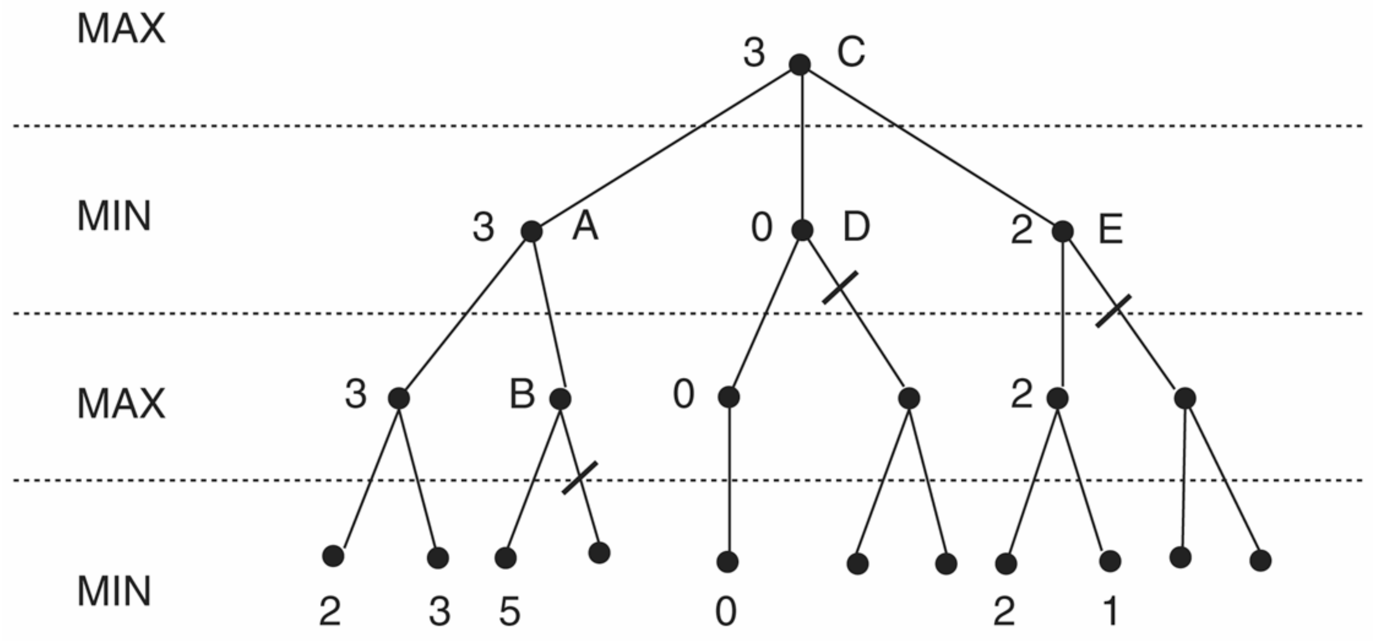
Example with MIN



(Some of) these still need to be looked at

As soon as the node with value 6 is generated, we know that the alpha value will be larger than 6, we don't need to generate these nodes (and the subtree below them)

Alpha-beta pruning applied to the state space of Fig. 4.15



A has $\beta = 3$ (A will be no larger than 3)

B is β pruned, since $5 > 3$

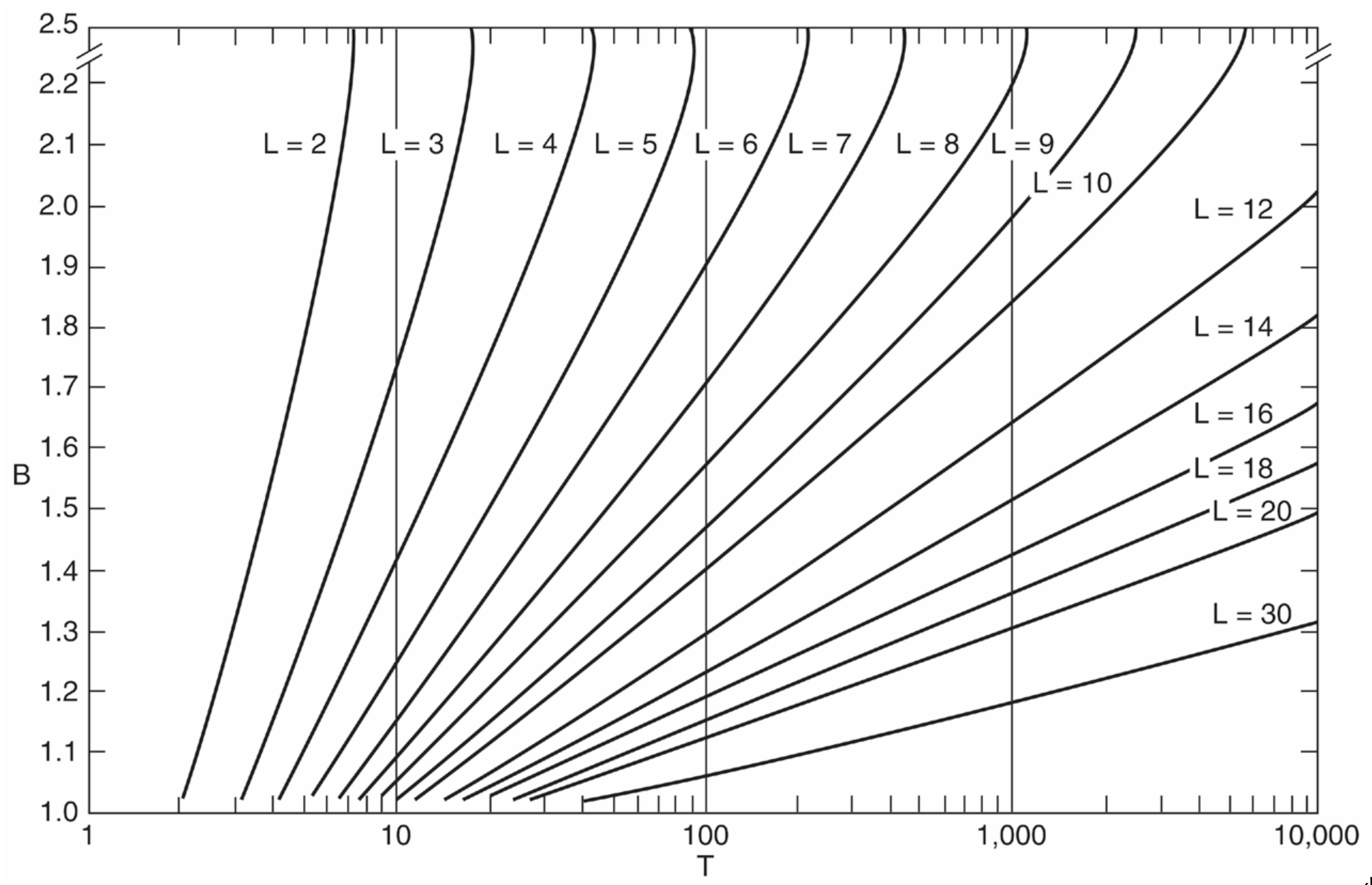
C has $\alpha = 3$ (C will be no smaller than 3)

D is α pruned, since $0 < 3$

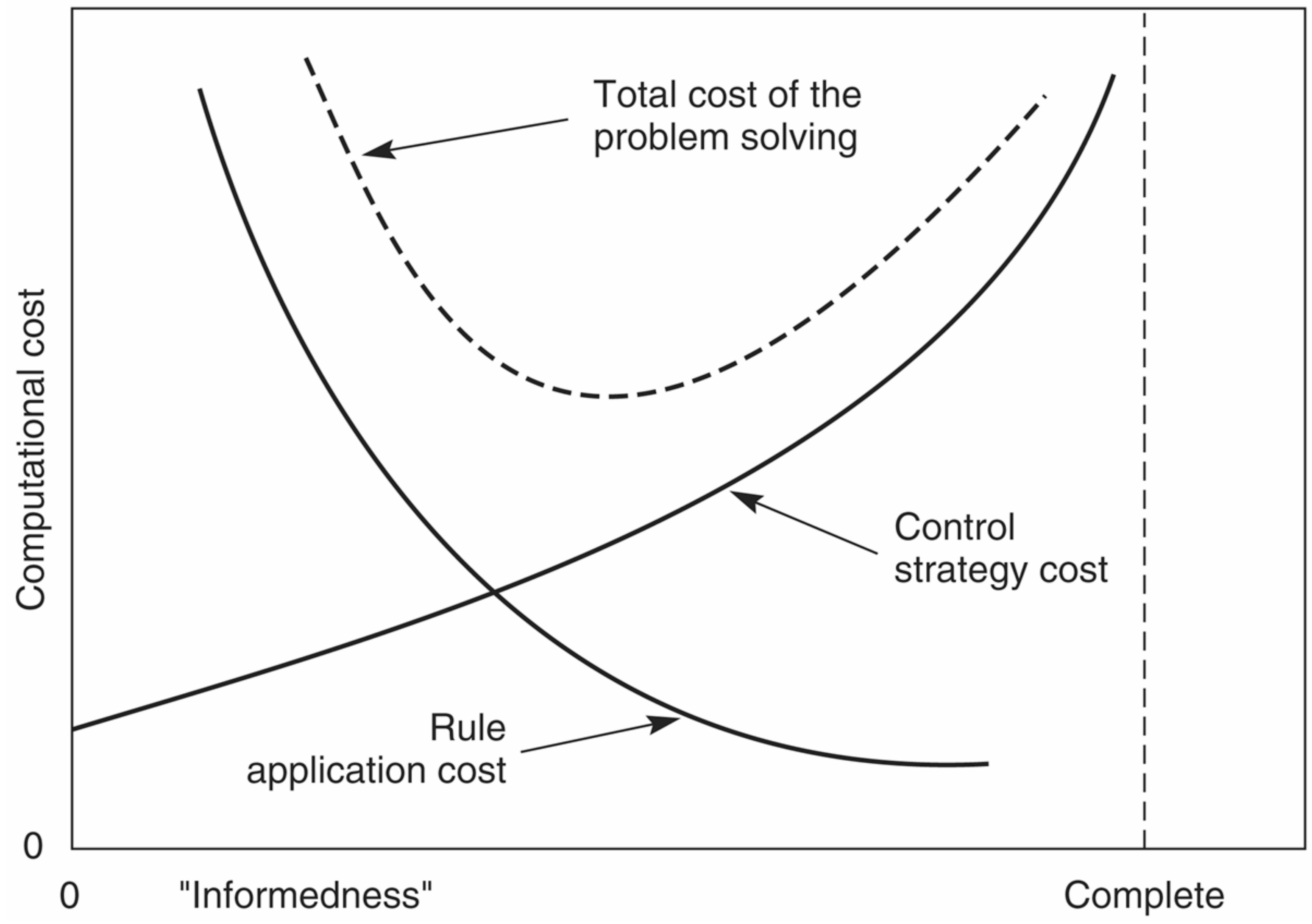
E is α pruned, since $2 < 3$

C is 3

Number of nodes generated as a function of branching factor B, and solution length L (Nilsson, 1980)

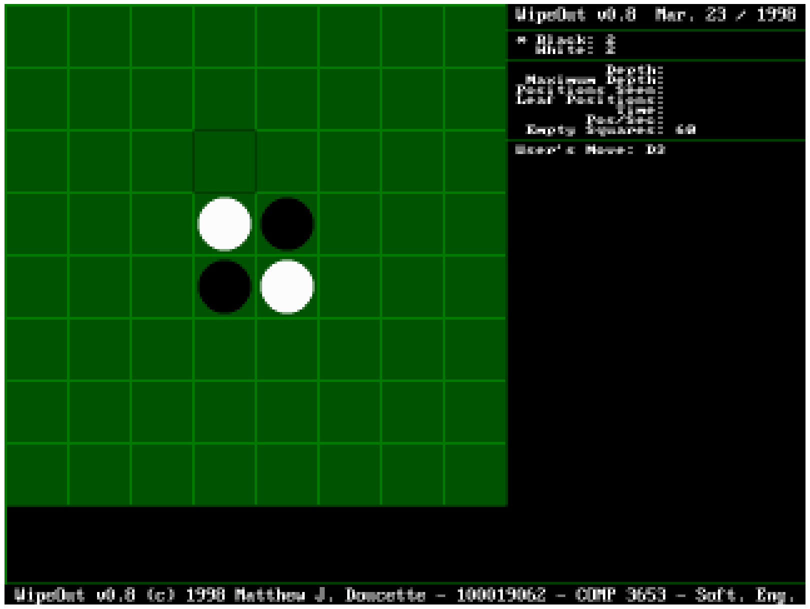


Informal plot of cost of searching and cost of computing heuristic evaluation against heuristic informedness (Nilsson, 1980)



Othello (a.k.a. reversi)

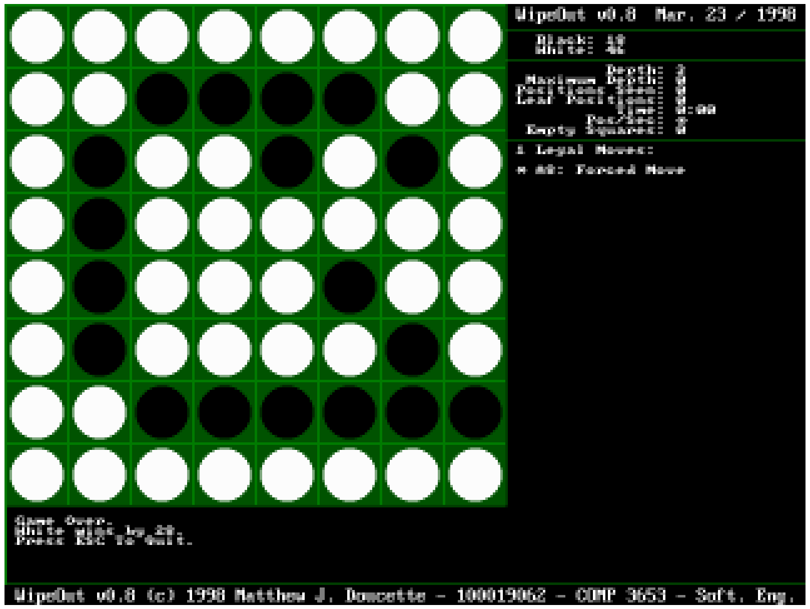
- 8x8 board of cells
- The tokens have two sides: one black, one white
- One player is putting the white side and the other player is putting the black side
- The game starts like this:



```
WipeOut v0.8  Mar. 23 / 1998
* Black: 2
  White: 2
                Depth:
  Maximum Depth:
Positions Seen:
Leaf Positions:
                 Time:
              Pos/Sec:
  Empty Squares: 60
User's Move: D3
WipeOut v0.8 (c) 1998 Matthew J. Doucette - 100019062 - COMP 3653 - Soft. Eng.
```

Othello

- The game proceeds by each side putting a piece of his own color
- The winner is the one who gets more pieces of his color at the end of the game
- Below, white wins by 28



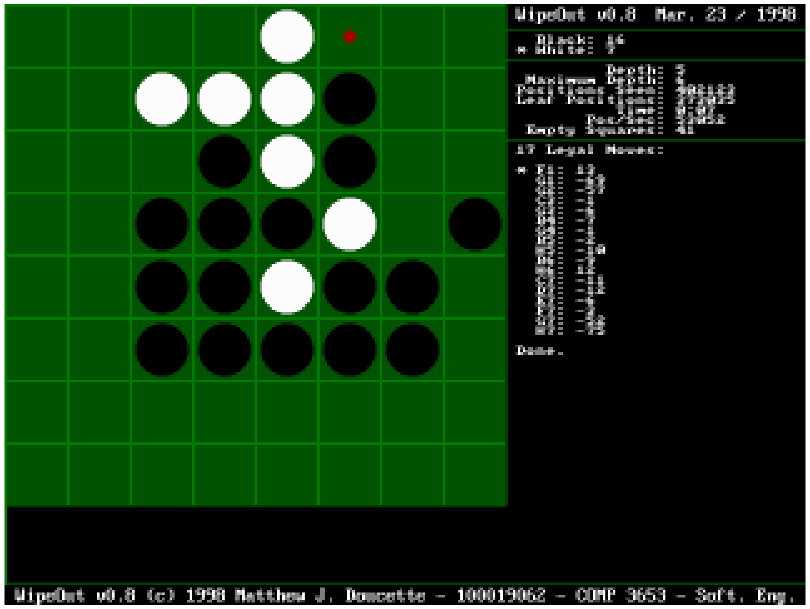
```
WipeOut v0.8  Mar. 23 / 1998

   Black: 18
   White: 46

            Depth: 3
  Maximum Depth: 0
Positions Seen: 0
Leaf Positions: 0
           Time: 0:00
        Pos/Sec: 0
  Empty Squares: 0

1 Legal Moves:
* 00: Forced Move
```

```
Game Over.
White wins by 28.
Press ESC to Quit.
```

```
WipeOut v0.8 (c) 1998 Matthew J. Doucette - 100019062 - COMP 3653 - Soft. Eng.
```

Othello

- When a black token is put onto the board, and on the same horizontal, vertical, or diagonal line there is another black piece such that every piece between the two black tokens is white, then all the white pieces are flipped to black
- Below there are 17 possible moves for white



```
WipeOut v0.8  Mar. 23 / 1998

  Black: 16
* White: 7

          Depth: 5
  Maximum Depth: 6
 Positions Seen: [illegible]
Leaf Positions: [illegible]
           Time: [illegible]
        Pos/Sec: [illegible]
  Empty Squares: 41

17 Legal Moves:

* F1:  12
[illegible]

Done.
```

```
WipeOut v0.8 (c) 1998 Matthew J. Doucette - 100019062 - COMP 3653 - Soft. Eng.
```

Othello

- A move can only be made if it causes flipping of pieces. A player can pass a move iff there is no move that causes flipping. The game ends when neither player can make a move

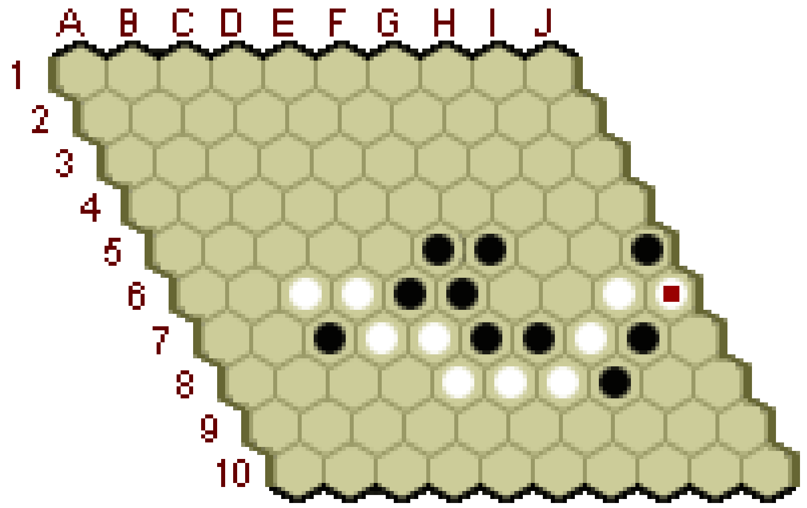
- the snapshots are from www.mathewdoucette.com/artificialintelligence

- the description is from home.kkto.org:9673/courses/ai-xhtml

- AAAI has a nice repository: www.aaai.org
Click on AI topics, then select “games & puzzles” from the menu

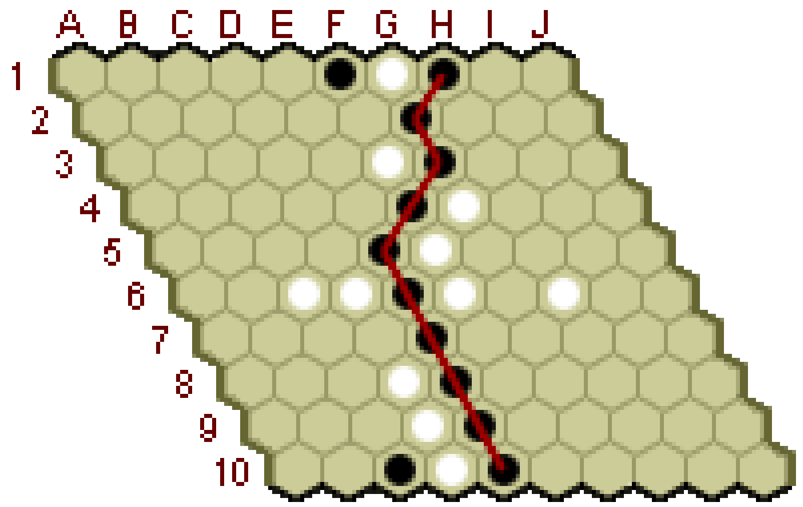
Hex

- Hexagonal cells are arranged as below . Common sizes are 10x10, 11x11, 14x14, 19x19.
- The game has two players: Black and White
- Black always starts (there is also a swapping rule)
- Players take turns placing their pieces on the board



Hex

• **The object of the game is to make an uninterrupted connection of your pieces from one end of your board to the other**



- **Other properties**
  - **First player always wins**
  - **No ties**

•Hex

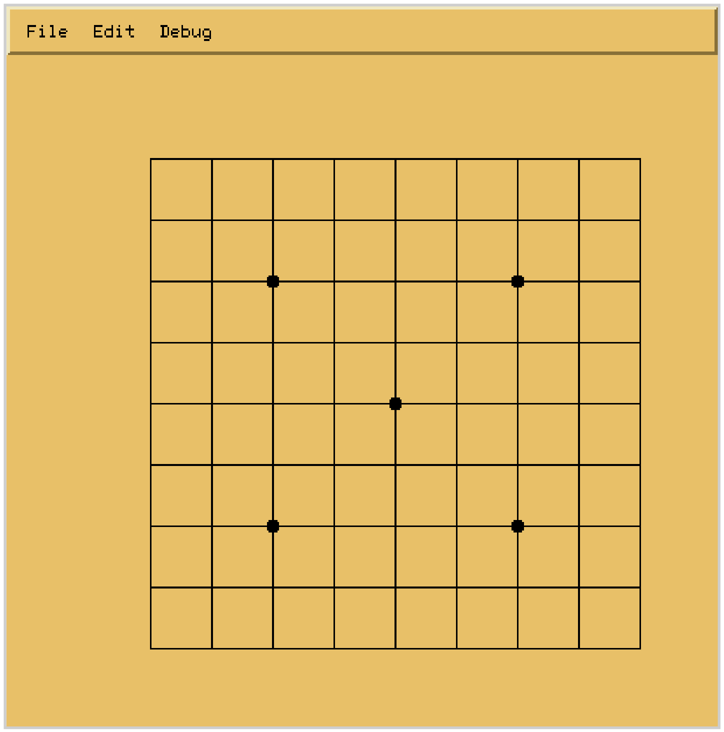
- Invented independently by Piet Hein in 1942 and John Nash in 1948.
- Every empty cell is a legal move, thus the game tree is wide b = ~80 (chess b = ~35, go b = ~250)
- Determining the winner (assuming perfect play) in an arbitrary Hex position is PSPACE-complete [Rei81].
- How to get knowledge about the “potential” of a given position without massive game-tree search?

Hex

- There are good programs that play with heuristics to evaluate game configurations
- hex.retes.hu/six
- home.earthlink.net/~vanshel
- cs.ualberta.ca/~javhar/hex
- www.playsite.com/t/games/board/hex/rules.html

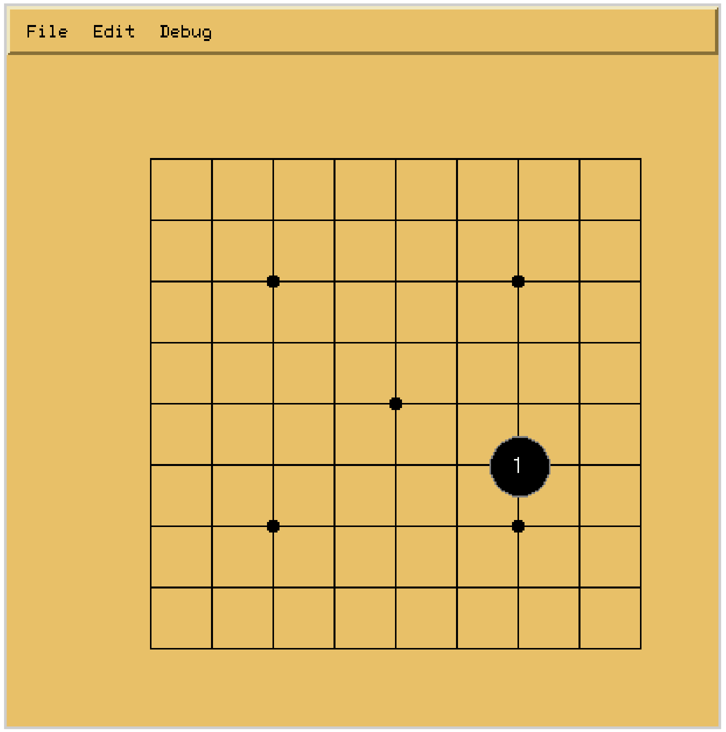
The Game of Go

Go is a two-player game played using black and white stones on a board with 19x19, 13x13, or 9x9 intersections.



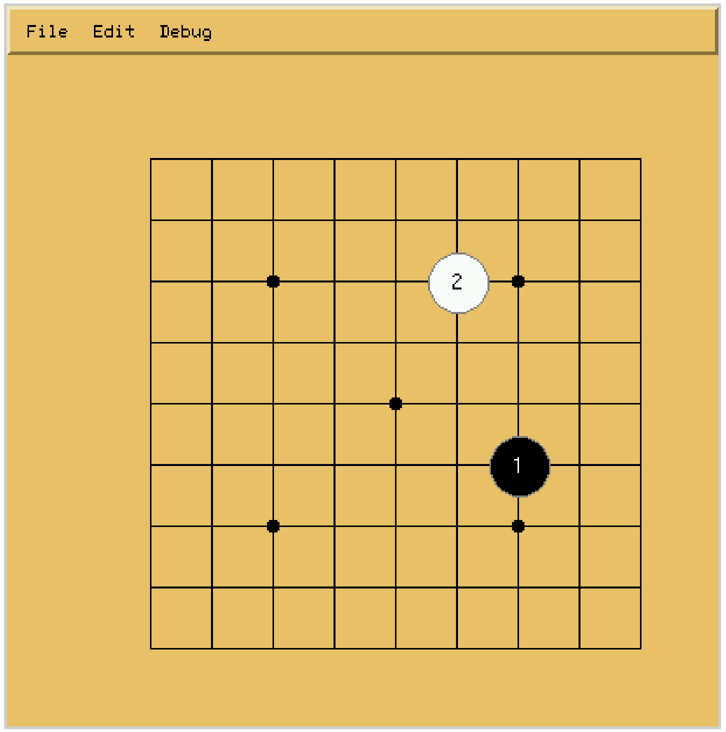
The Game of Go

Players take turns placing stones onto the intersections.
Goal: surround the most territory (empty intersections).

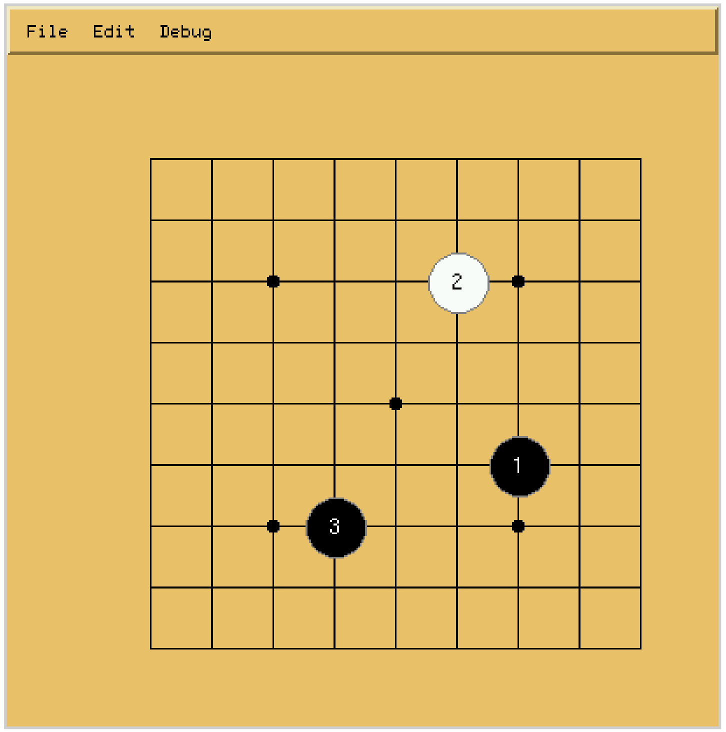


The Game of Go

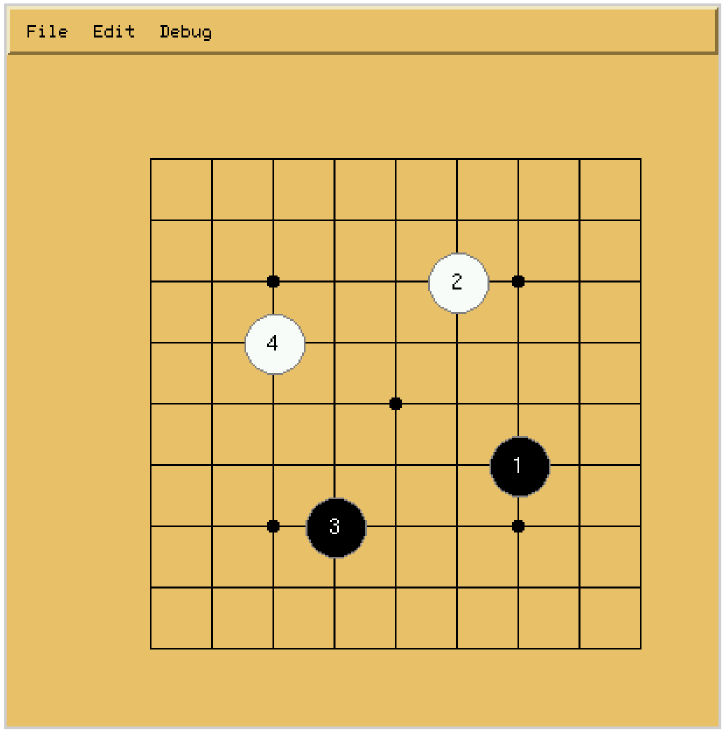
Once placed onto the board, stones are not moved.



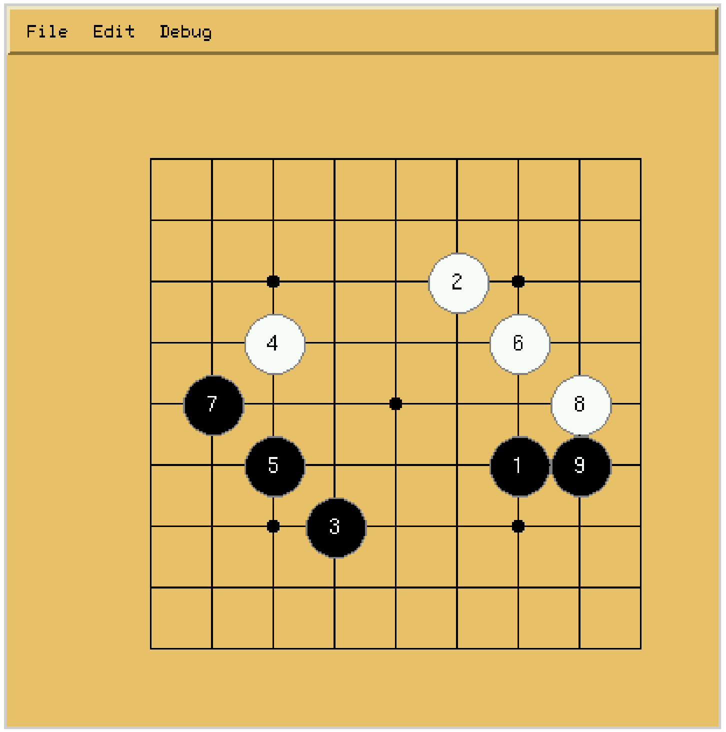
The Game of Go



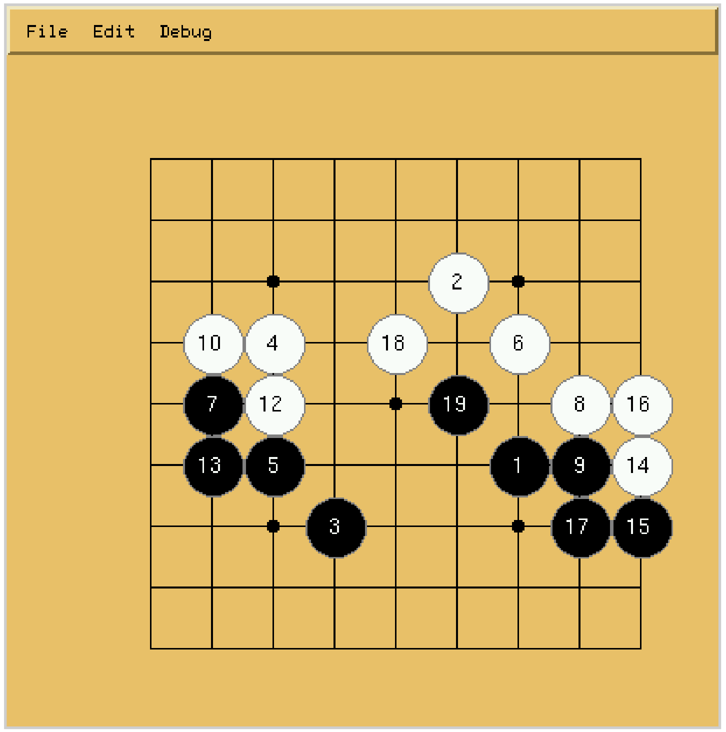
The Game of Go



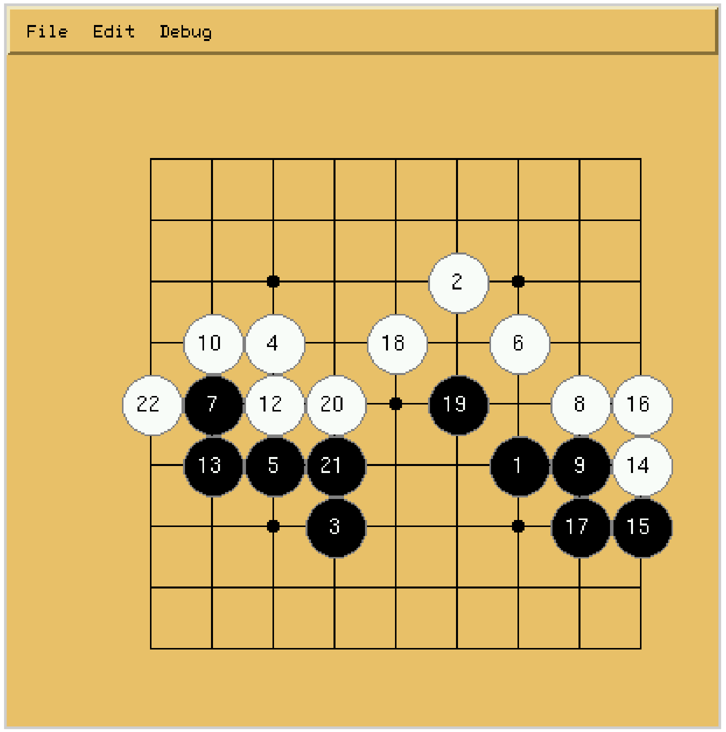
The Game of Go



The Game of Go

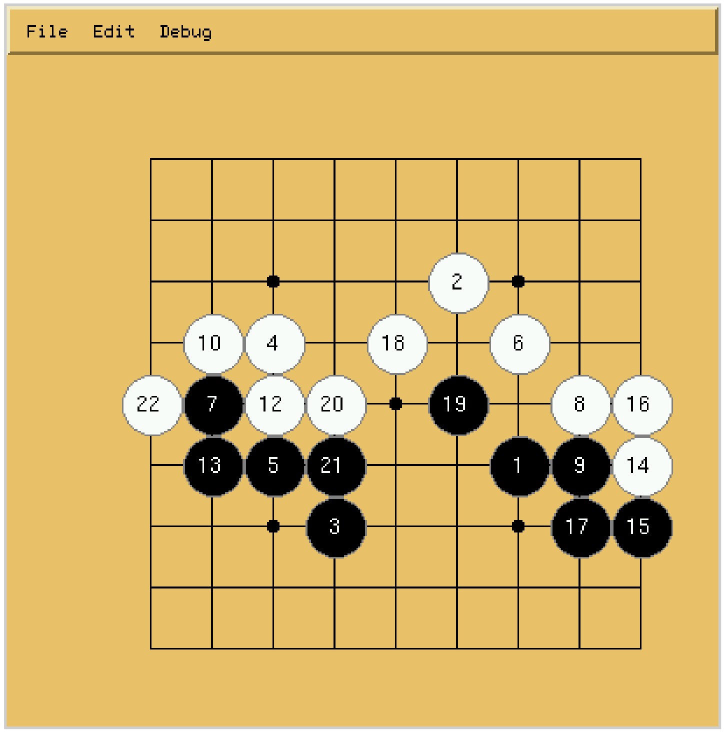


The Game of Go



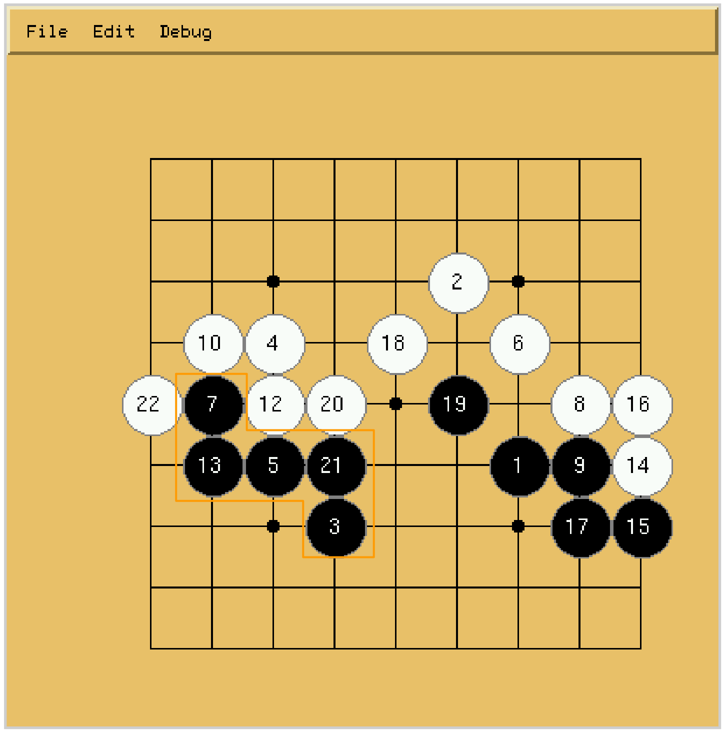
The Game of Go

A **block** is a set of adjacent stones (up, down, left, right) of the same color.



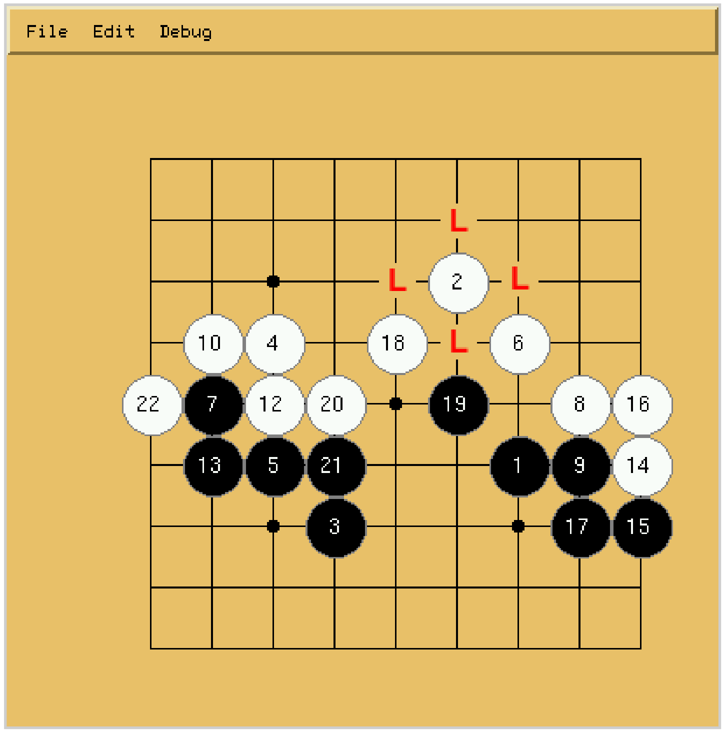
The Game of Go

A **block** is a set of adjacent stones (up, down, left, right) of the same color.

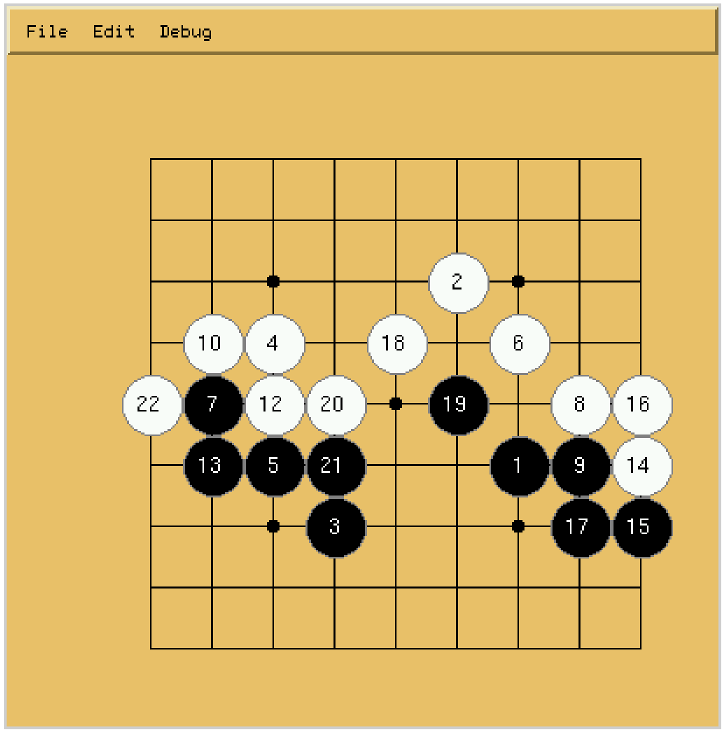


The Game of Go

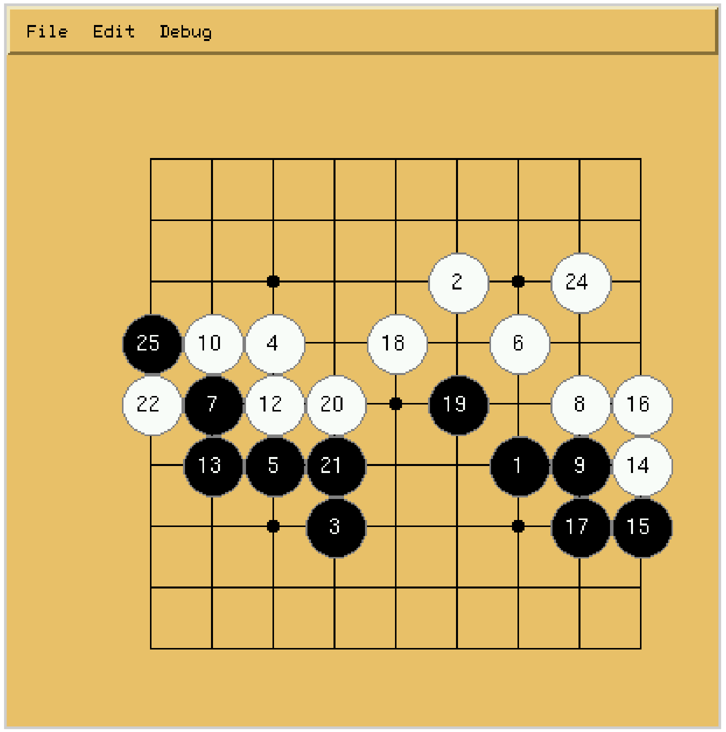
A **liberty** of a block is an empty intersection adjacent to one of its stones.



The Game of Go

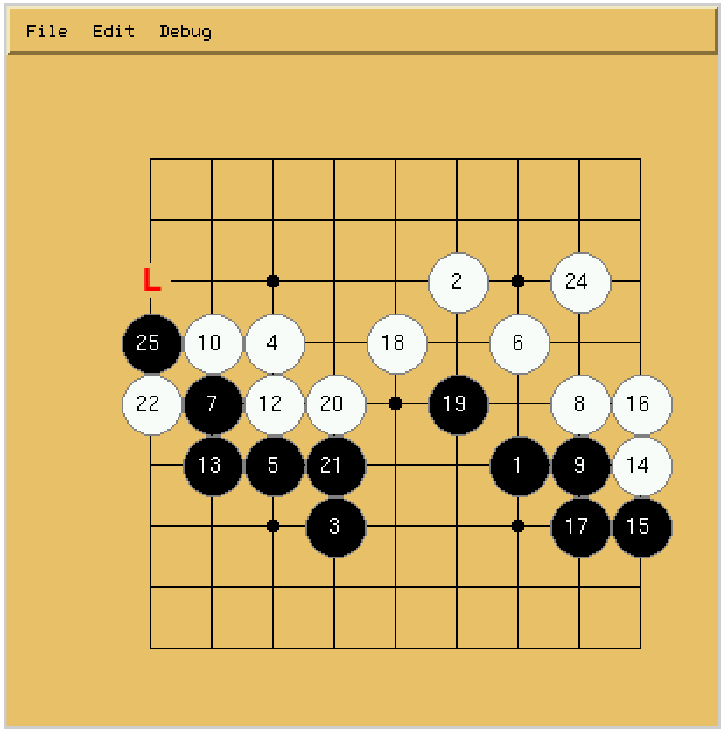


The Game of Go



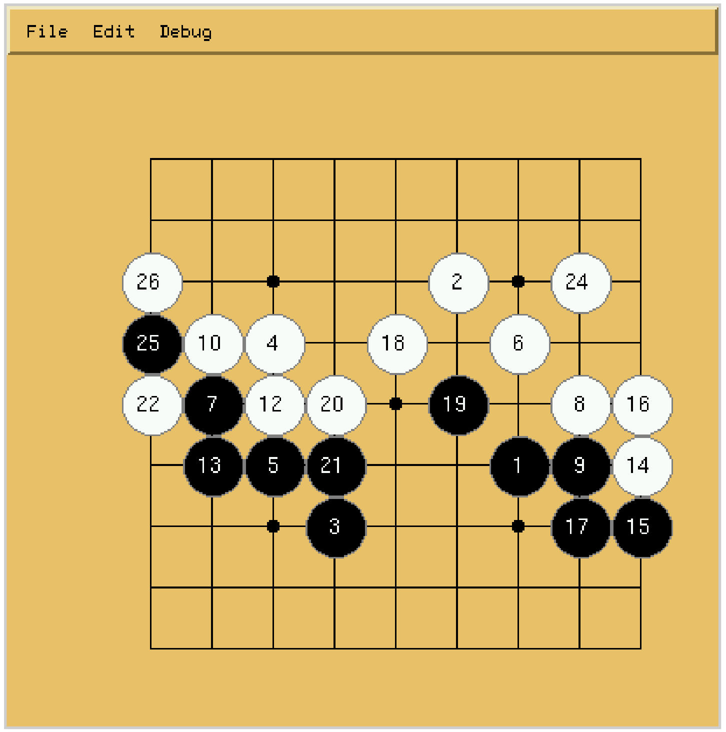
The Game of Go

If a block runs out of liberties, it is captured. Captured blocks are removed from the board.



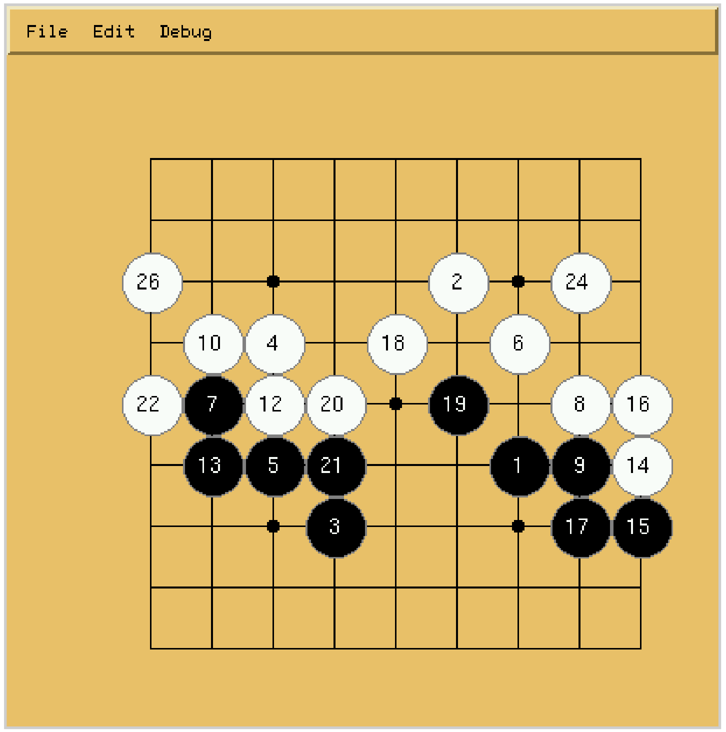
The Game of Go

If a block runs out of liberties, it is captured. Captured blocks are removed from the board.



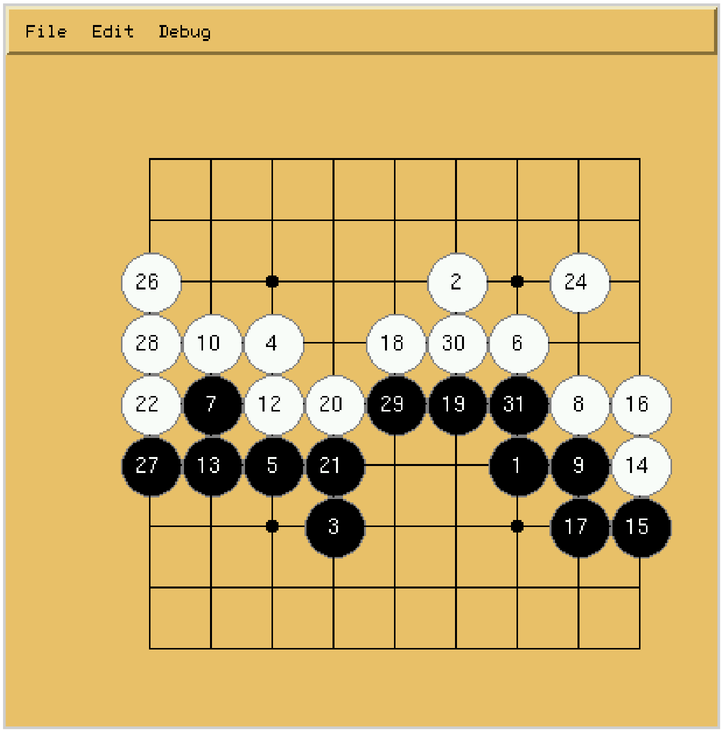
The Game of Go

If a block runs out of liberties, it is captured. Captured blocks are removed from the board.



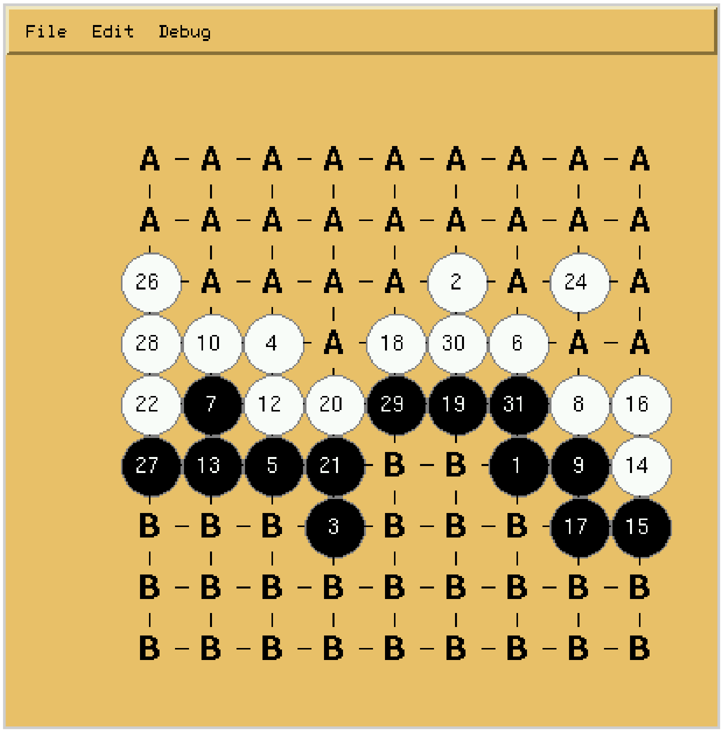
The Game of Go

The game ends when neither player wishes to add more stones to the board.



The Game of Go

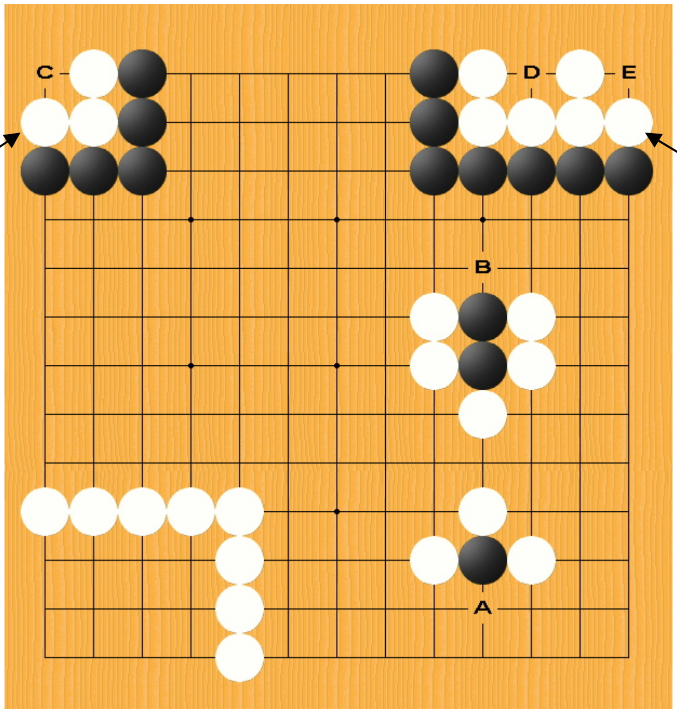
The player with the most enclosed territory wins the game. (With *komi*, White wins this game by 7.5 pts.)



Alive and Dead Blocks

White can capture by playing at A or B. Black can capture by playing at C. Black can't play at D and E simultaneously.

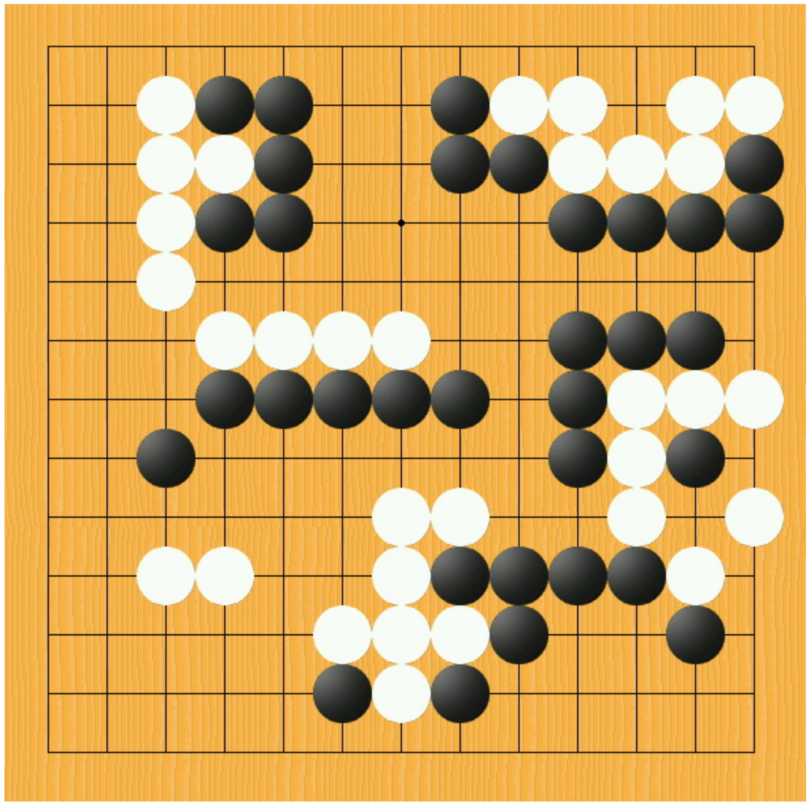
With only one eye, these stones are dead. No need for Black to play at C.



With two eyes at D and E, these White stones are alive.

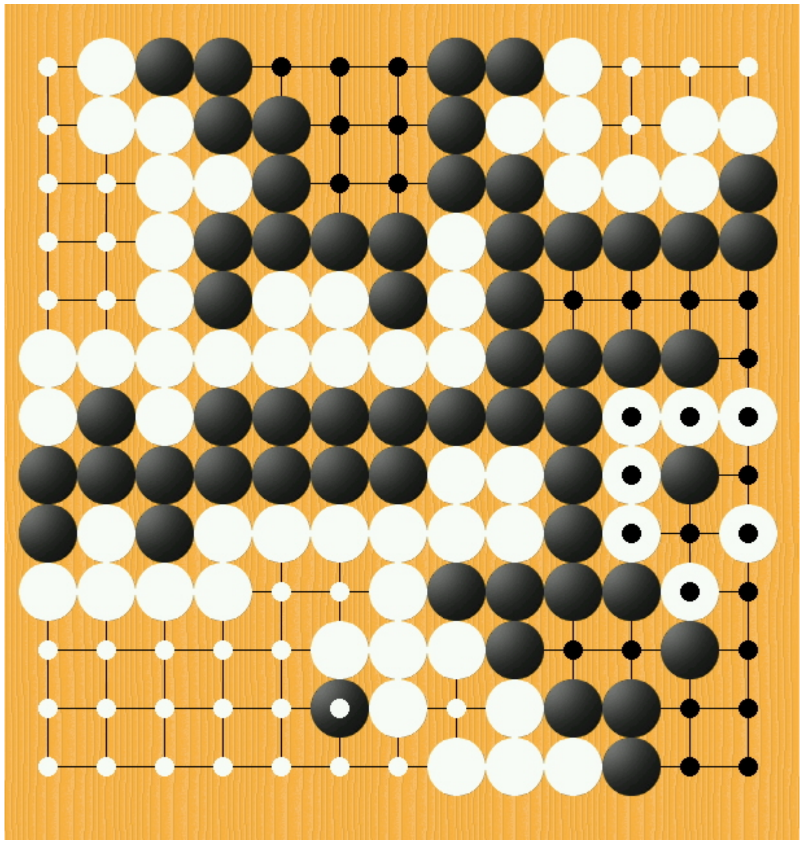
Example on 13x13 Board

What territory belongs to White? To Black?



Example on 13x13 Board

Black ahead by 1 point. With *komi*, White wins by 4.5 pts.



Challenges for Computer Go

Much higher search requirements

- Minimax game tree has $O(b^d)$ positions
- In chess, b = ~35 and d = ~100 half-moves
- In Go, b = ~250 and d = ~200 half-moves
- However, 9x9 Go seems almost as hard as 19x19

Accurate evaluation functions are difficult to build and computationally expensive

- In chess, material difference alone works fairly well
- In Go, only 1 piece type with no easily extracted features

Determining the winner from an arbitrary position is PSPACE-hard (Lichtenstein and Sipser, 1980)

State of the Art

Many Faces of Go v.11 (Fotland), Go4++ (Reiss), Handtalk/Goemate (Chen), GNUGo (many), etc.

Each consists of a carefully crafted combination of pattern matchers, expert rules, and selective search

Playing style of current programs:

- Focus on safe territories and large frameworks
- Avoid complicated fighting situations

Rank is about 6 kyu, though actual playing strength varies from opening (stronger) to middle game (much weaker) to endgame (stronger)