Game Playing

Game playing is an idealization of worlds in which hostile agents act so as to diminish one‘s well-being.

Introduction: Games as Search Problems

Why Game Playing

- Simple Rules
- Fully accessible world means *precise representation* of a game as a search through a space of possible game positions

Characteristics

- The presence of an opponent makes the decision problem somewhat more complicated than the simple search problems.
- The opponent introduces uncertainty- all game playing programs must deal with the contingency problem.
- **Much too hard to solve**. It relies heavily on one‘s past experience - thus games are much more like the real world than the standard search problems.
- **Time** is very important - must make the best use of time to reach good decisions, when reaching optimal decision is impossible.

Perfect Decisions in Two-Person Games :

A game can be formally defined as a kind of search problem with the following components:

The **initial state** : board position, whose move, etc..

A set of **operators** : define legal moves.

A terminal test: determines when the game is over, i.e. determines the **terminal states**.

A **utility function** (also called the **payoff function**) : gives a numeric value for the outcome of a game.

Minimax Algorithm

✧The algorithm must find for **MAX** a strategy that will lead to a winning terminal state regardless of what **MIN** does, including the correct move for **MAX** for each possible move by **MIN**.

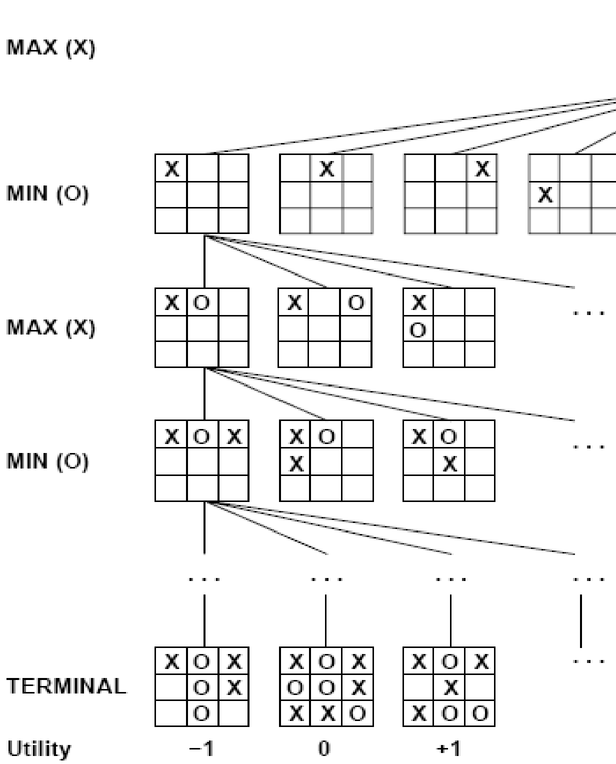


Figure 1: A (partial) search tree for the game of Tic-Tac-Toe.

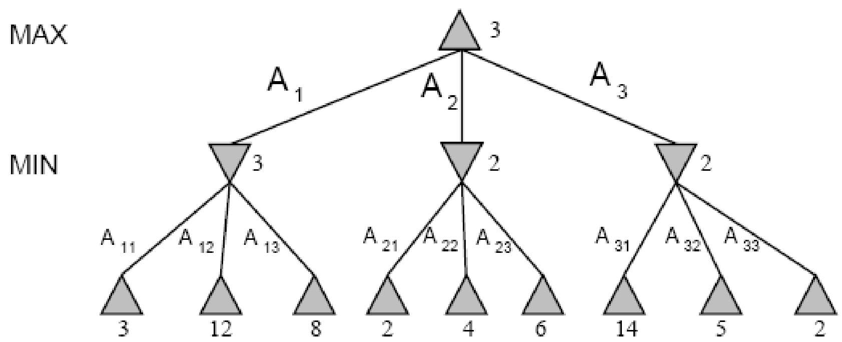


Figure 2: A two-ply game tree as generated by the minimax algorithm.

✧The algorithm must do the minimax decision, in which it maximizes the utility value under the assumption that the opponent will play perfectly to minimize it.

✧The algorithm:

1. Generate the whole game tree.
2. Apply utility function to leave states to get their utility values
3. Recursively do from the leaves: **min** chooses the min among choices, **max** chooses the max, backing up toward the root, one player at a time.
4. At the top **max** goes to the node that maximizes outcome.

✧The code:

```
function MINIMAX-DECISION(game) returns an operator

  for each op in OPERATORS[game] do
     VALUE[op] ← MINIMAX-VALUE(APPLY(op, game), game)
  end
  return the op with the highest VALUE[op]
--------------------------------------------------------------
function MINIMAX-VALUE(state, game) returns a utility value

  if TERMINAL-TEST[game](state) then
     return UTILITY[game](state)
  else if MAX is to move in state then
     return the highest MINIMAX-VALUE of SUCCESSORS(state)
  else
     return the lowest MINIMAX-VALUE of SUCCESSORS(state)
```

Figure 3: An algorithm for calculating minimax decision.

✧Finds **optimal** strategy

✧**Time**: O(branch_factor^depth)

✧**Space**: O(branch_factor*depth) (if implement as DFS)

✧**The Problem**: Must assume that the whole tree can be generated.

Imperfect Decisions:

The algorithm must cut of the search earlier and apply a heuristic **evaluation function** to the leaves of the tree, thus -

Utility function $\Rightarrow$ EVAL function

Terminal test $\Rightarrow$ CUTOFF-TEST

Evaluation Function

Returns an ***estimate*** of the expected utility of the game from a given position.

✧Usually use **material values**. For example:

Pawn = 1,
knight or bishop = 3,
rook = 5, and
queen = 9, etc
"good pawn structure" and "king safety" = 0.5

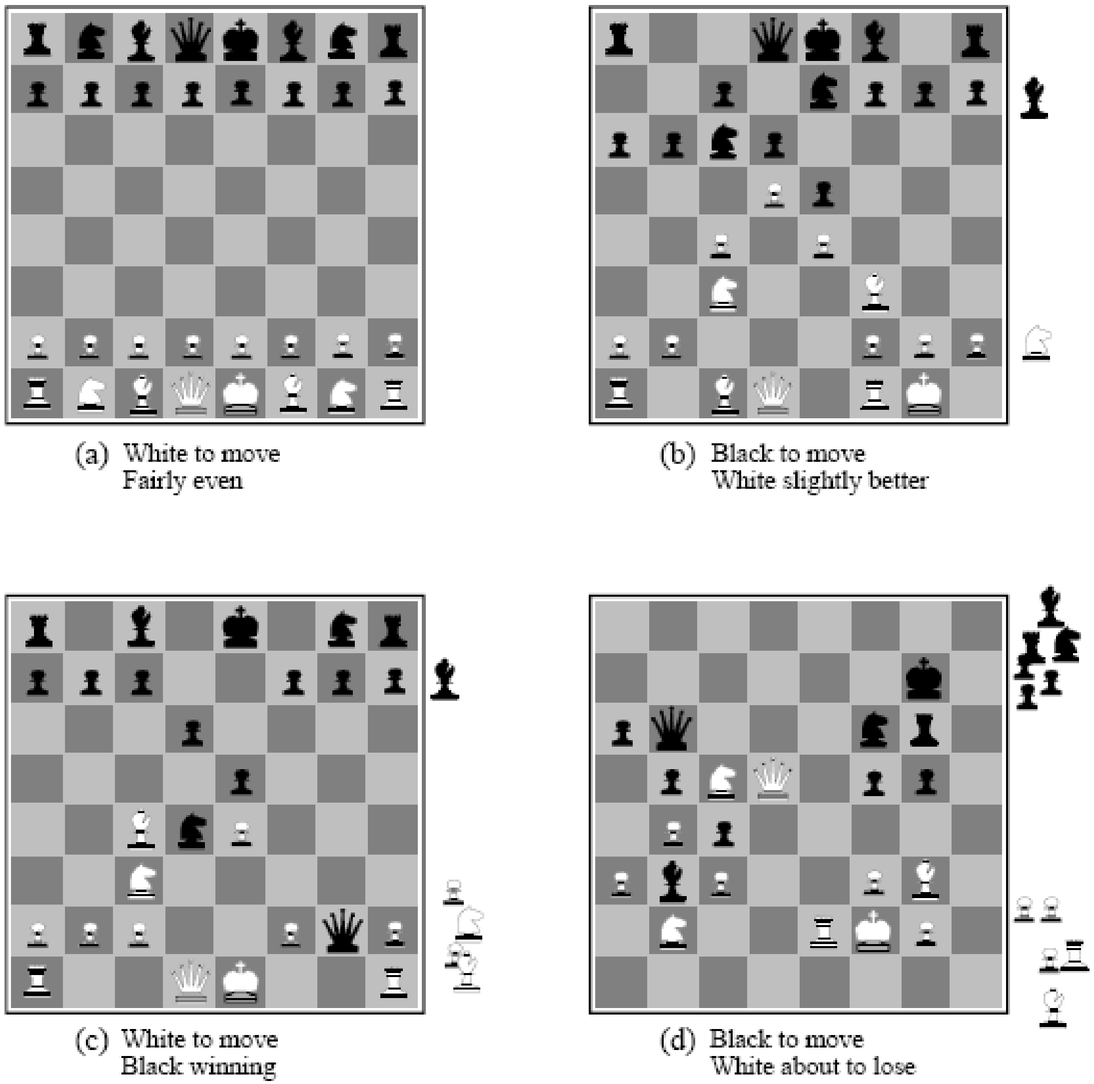


Figure 4: Some chess positions and their evaluations.

✧The performance of a game-playing program is extremely dependent on the quality of its evaluation function.

✧Characteristics of a good evaluation function:

1. Should agree with the real utility function on **terminal states**.
2. Should be **quick** to compute. is usually a trade-off between the accuracy of the evaluation function and its time cost.
3. Should accurately reflect the actual chances of winning.

✧Types of evaluation functions:

1. **Weighted linear function**: sum of weights time features

$$\mathbf{w_1 f_1 + w_2 f_2 + ... + w_n f_n}$$

Assume that the value of a piece can be judged independently of the other pieces present on the game board.

2. **Non-linear**: (e.g. neural nets for backgammon)

Cutting Off Search

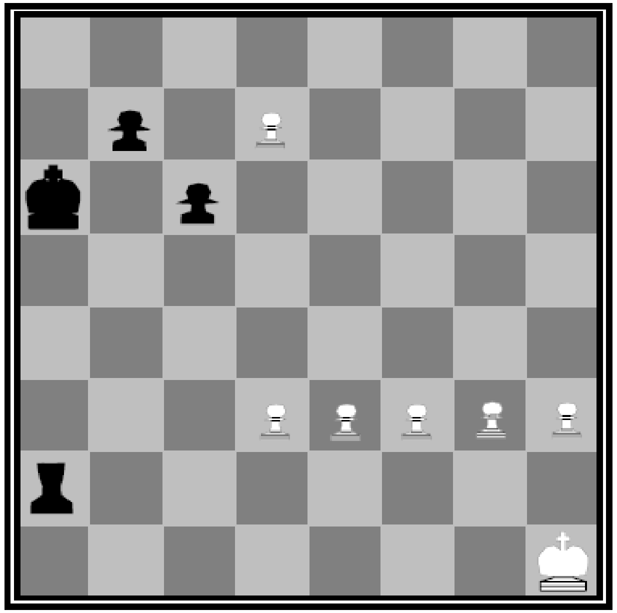
✧When to cutoff searching: Using

1. Fixed depth limit - the cutoff test succeeds for all nodes at or below depth ***d***.
2. Iterative deepening - when time runs out, the program returns the move selected by the deepest completed search.

- ✧Problems: May cutoff before some really good move for opponent. For example, in **(*Figure 4.d*)**, it seems white is winning but actually its queen is losing.
- ✧The Solution: Continue search down the move until **quiescence** positions are reached.

A quiescent position is a position that is impossible to exhibit wild swing, i.e. neither player gains much. This kind of search is called the **Quiescence search**.

✧Another problem: The **horizon problem**: when opponent is about to gain, program may try to stall the inevitable by pushing that gain beyond its horizon.



Black to move

Figure 5: The horizon problem. A series of checks by the black rook forces the inevitable queening move by white "over the horizon" and makes this position look like a slight advantage for black, when it is really a sure win for white.

The problem with fixed-depth search is that it believes that these stalling moves have avoided the damaging move by the opponent.

Alpha-Beta Pruning:

In reality, under time constraint, it is impossible to search many ply if every node in the search tree is examined.

For example: Can search 1000 position a second.
With branching factor 35,
150 seconds per move can do only 3-4 ply only.

The alpha-beta pruning algorithm

Can still compute the correct minimax decision after eliminating a branch of the search tree.
Using

alpha α = best choice so far at any choice point along the path for MAX, and

beta β = best choice(i.e. the lowest-value) so far for Min.

to prun a subtree (i.e. terminate the recursive call) as soon as it is known to be worse than the current α or β value

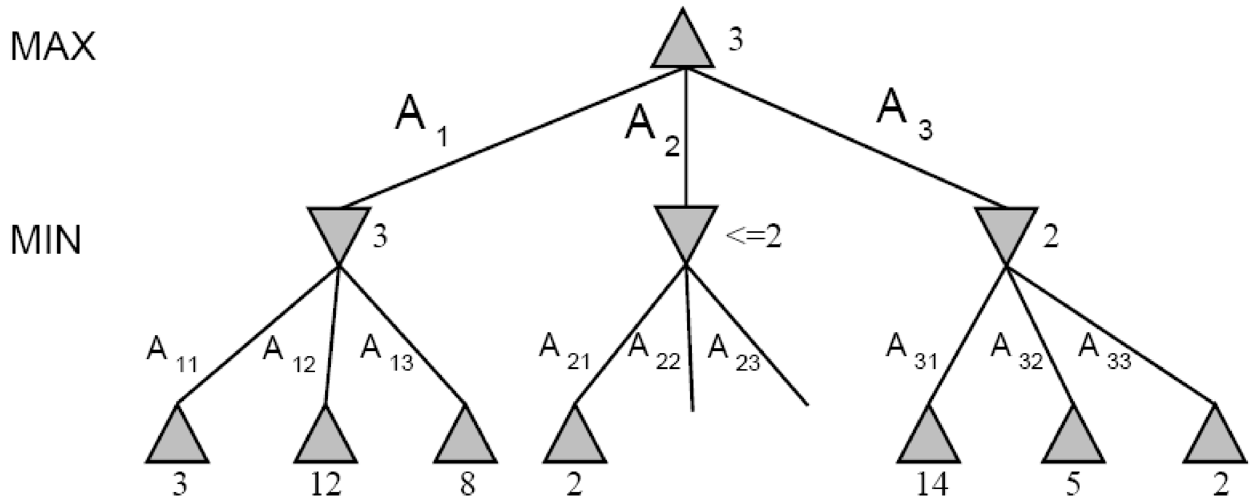


Figure 6: The two-ply games tree as generated by alpha-beta.

```
function MINIMAX-DECISION(state) returns an action
    v ← MAX-VALUE(state)
    return the action in SUCCESSORS(state) with value v

function MAX-VALUE(state) returns a utility value
    if TERMINAL-TEST(state) then return UTILITY(state)
    v ← −∞
    for a, s in SUCCESSORS(state) do
        v ← MAX(v, MIN-VALUE(s))
    return v

function MIN-VALUE(state) returns a utility value
    if TERMINAL-TEST(state) then return UTILITY(state)
    v ← ∞
    for a, s in SUCCESSORS(state) do
        v ← MIN(v, MAX-VALUE(s))
    return v
```

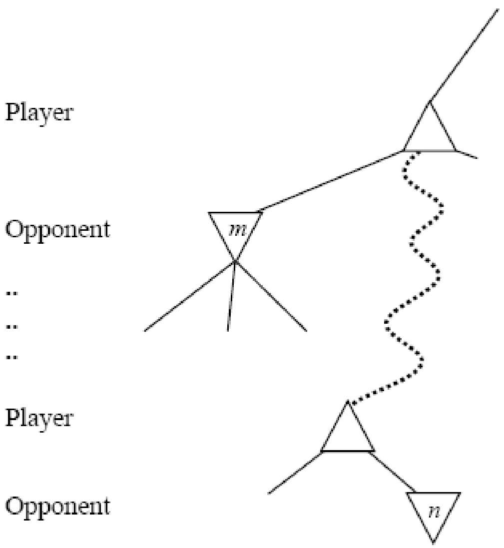


Figure 7: Alpha-beta pruning: the general case. If m is better than n for Player, we will never get to n in play.

Effectiveness of alpha-beat pruning

- Effectiveness depends on the order successors are examined.
- If the most likely to be the best successor nodes can be examined first(?) then the algorithm needs **O(branch^(depth/2))** to pick the best move instead of O(branch^depth) of minimax.
- Effective branch factor is **sqrt(branch)** and the alpha-beta would be able to look **twice** as far in same time.

Randomly pick next successor:

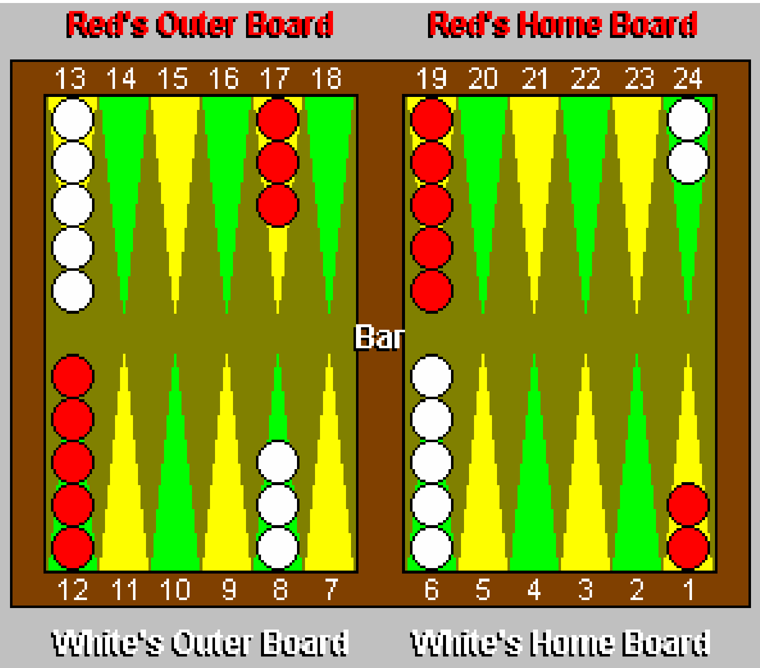
Asymptopic complexity O((b/log b)^depth) (approx O(b^depth)). Not good enough!

✧Improvements that can be achieved easily:

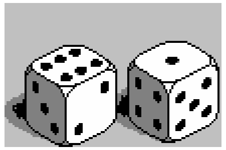
1. Use even a simple ordering scheme (e.g. try to capture first, then threats, then forward moves,...)
2. Do iterative deepening and determine the ordering of successors by values of previous iteration.

Games that include an element of chance:

Games that incorporate luck and skill introduce the **unpredictability** by including a random element such as throwing dice.



A backgammon board with the checkers in their initial arrangement. Numbering of the points is from White's point of view. An alternate arrangement is the reverse of the one shown here, with the home board on the left and the outer board on the right.



Chance nodes: nodes to denote the die roll or whatever associated to random elements.

A game with a random element must include in its game tree chance nodes in addition to MAX and MIN nodes.

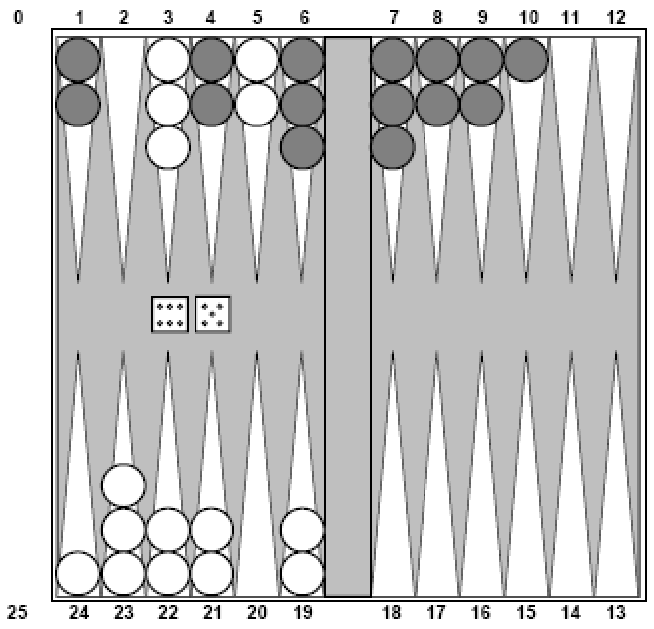


Figure 9: A typical backgammon position. The aim of the game is to move all one's pieces off the board. White moves clockwise toward 25, and black moves counterclockwise toward 0. A pieces can move to any position except one where there are two or more of the opponent's pieces. It it moves to a position with one opponent piece, that piece is captured and has to start its journey again from the beginning. In this position, white has just rolled 6-5 and has the four legal moves:(5-10, 5-11), (5-11, 19-24), (5-10, 10-16), and (5-11, 11-16).

Each possible position no longer has a **definite minimax value**, but an **average** or **expected value** which is taken over all the possible dice roolls that could occur.

Expectimax and Expectimin values

- For terminal nodes, the expected value is calculated by using utility function, just like in deterministic games.
- At a **chance node**(for MAX), compute the expected value **expectimax** of each move by summing the probability of a state by its value.

$$\textbf{expectimax}(\mathrm{C}) = \sum_i P(d_i) \max_{s \in S(C, d_i)} (Utility(s))$$

where

P(d$_i$) is the chance or probability of obtaining a roll.

S(C, di) is the set of positions generated by applying the legal moves for dice roll $P(d_i)$ to the position at C.

expectimax = 0;
for each dice roll $\boldsymbol{d_i}$ from chance node C:
{
s = state with max utility under $\boldsymbol{C},\boldsymbol{d_i}$
expectimax += P($\boldsymbol{d_i}$) * utility(s)
}

✧At a **chance node**(for MIN), compute the expected value **expectimin** of each move by summing the probability of a state by its value using a formula that is analogous to expectimax.

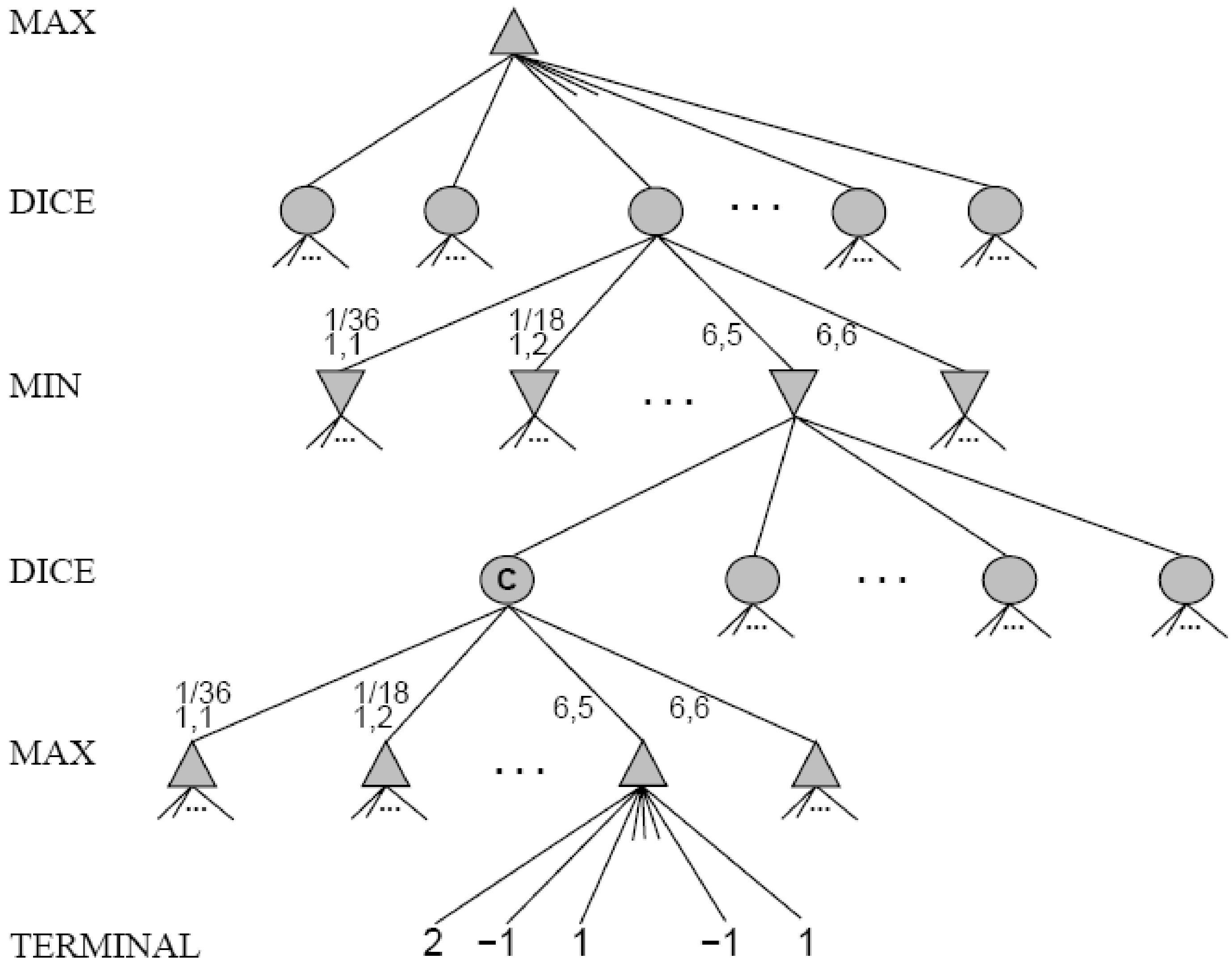


Figure 10: Schematic game tree for a backgammon position.

Position evaluation in games with chance nodes

✧The evaluation function must be a ***positive linear*** transformation of the likelihood of winning from a position.

✧This is to avoid the situations in ***Figure 5.11***

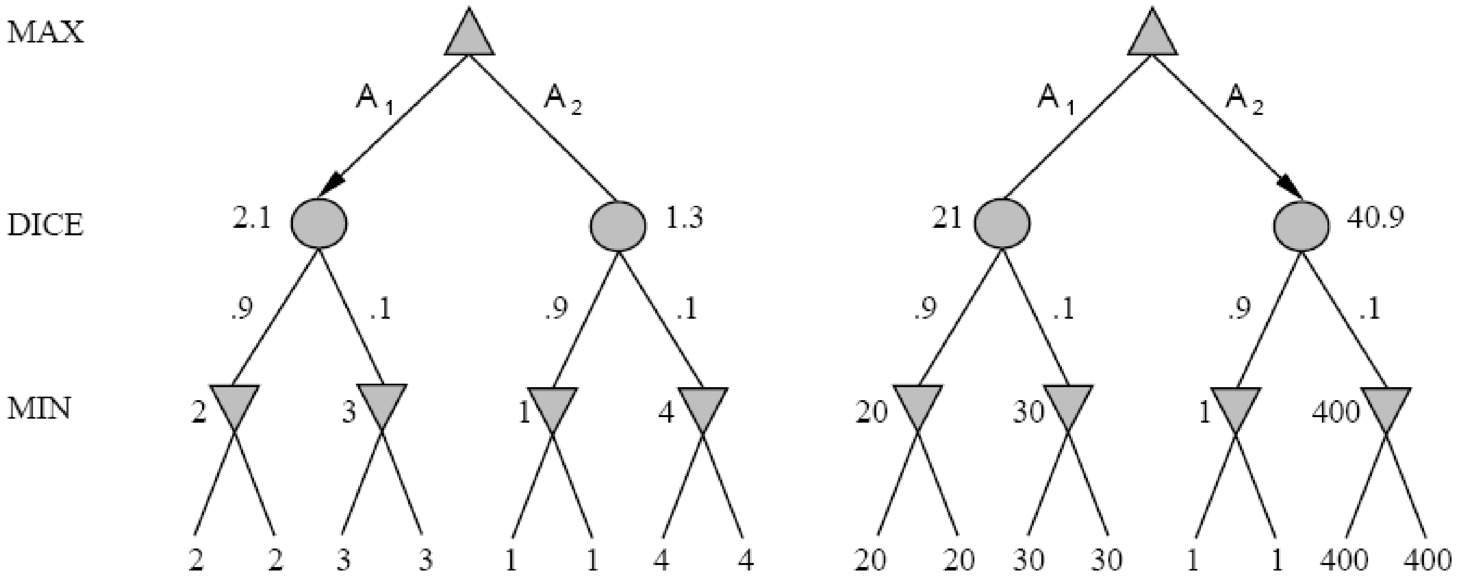


Figure 11: An order-preserving transformation on leaf values changes the best move.

Complexity of Expectiminmax

- **Complexity:O(brandh^depth*num_rolls^depth)**
- IF there are limits on highest and lowest payoffs THEN we can prune.

State-of-the-Art Game Programs:

✧**Chess**: Deep Thought, Deep Blue. ***(Figure 12)***

✧**Checkers**:

1. Samuel's Checker Player: an original game player that learned with a linear evaluation function.
2. Chinook: Better than all but Dr. Tinsley.

✧**Othello**: Computers are better than humans.

✧**Backgammon**: Gerry Tesauro‘s program is reliably ranked among the top three players in the wolrd.

✧**Go**: Not yet, branch factor is too high.

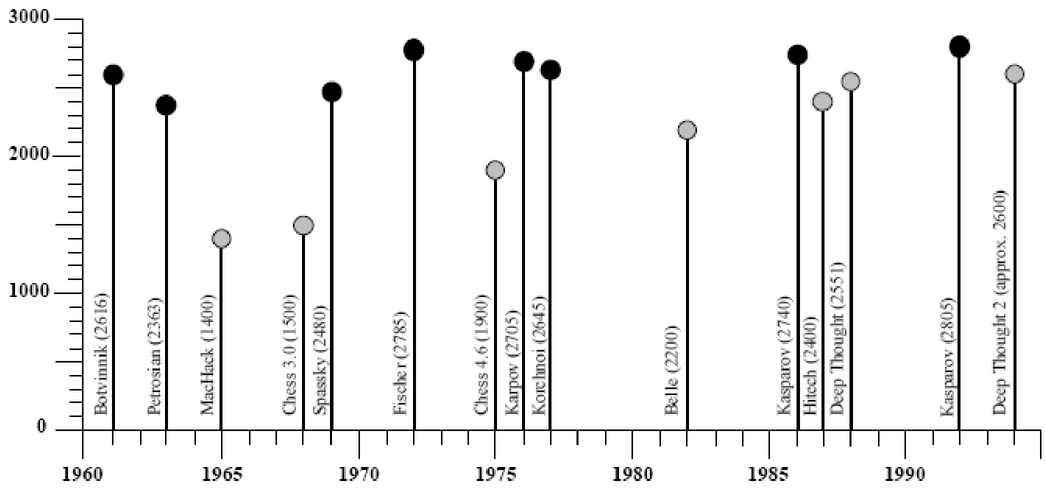
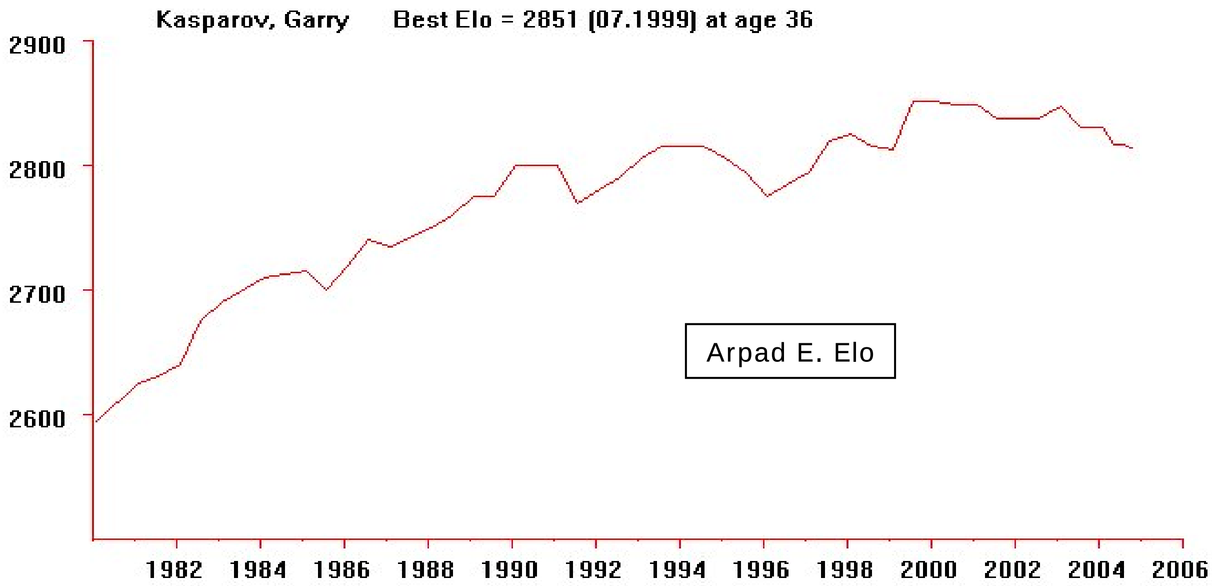


Figure 12: Ratings of human and machine chess champions.



- **Deep Blue**
  - Beaten by Garry Kasparov in 1996. World Champion Garry Kasparov took on Deep Blue computer in a 6 game match. He lost game 1, won game 2, drew games 3 and 4, and won games 5 and 6 to win the match with a 4-2 score.
  - Defeated Garry Kasparov in 1997 match (2 wins 3 draws 1 lose)
  - The sixth and final game in the series ended in a draw after 27 moves, leaving both sides with three points each, having each won a game with three other matches drawn. (Second match, Sun. February 09, 2003)
  - The third match was in New York, on 26 January 2003 Garry Kasparov vs. Deep Junior. Draw.
  - 32 P2SC Processors, capable of searching 50 to 100 billion positions within three minutes
  - 1000 times faster than its predecessor, Deep Thought
- **Pacific Blue**

- Powered by 8,192 IBM Power3 processors and has 160 terabytes of disc storage.
- Process information at 12.3 trillion calculations per second.
- Being used for nuclear weapon testing.

- **Blue Gene**
  - Running on 32 specially constructed CPU chips containing both memory and communication circuits.
  - Possible to do more than a quadrillion calculations per second.
  - Blue Gene will be 500 times faster than Pacific Blue and a thousand times faster than Deep Blue
  - Blue Gene will study gene sequences and the complex shapes of human proteins
  - Also be used to study weather and global climate changes,

Discussion:

Minimax + Alpha-beta : The only solution?

- Minimax is an optimal method for selecting a move from a given search tree *provided the leaf node evaluations are exactly correct*.
- In reality, it might not be the case, see ***(Figure 13)***.
- It might have to combine the evaluation with the *probability distribution*.

Meta-reasoning (reasoning about reasoning)

- The idea of the ***utility of a node expansion*** - good search algorithm should select node expansions of high utility.
- If there are no node expansions whose utility is higher than their cost(in terms of time0, then the algorithm should stop searching and make a move.
- Alpha-beta algorithm is designed to calculate the values of *all the legal moves*- even if there is only **one** legal move.

Goal-directed reasoning or planning

- The actual reasoning method of human player.