Min Max Search

Min-Max Search is used in AI to implement systems that can play ***zero-sum perfect-information games*** (but not only…)

Typically two players board games: Chess, Checkers, …

It is an adaptation of hill climbing heuristic search alternating the choice of what state to choose next, according to the min or the max of the heuristic evaluation of the board position, depending which player turn is.

The idea is that while player A is *trying* to maximize the evaluation of the position, the opponent is *believed* to be minimizing the same evaluation.

Heuristic Evaluation

H_0 is an evaluation function of the configuration situation in the game. The evaluation is defined from the point of view of one player (say Max). Intuitevily it defines how good is a given configuration for the player.

Given H_0 we compute the evaluation H_L of a given configuration with a lookahed **L**

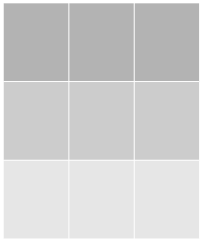
H_0 Examples

Chess …..

Othello/Reversi ….

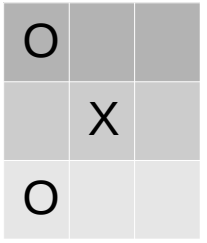
H_0 TicTacToe

0

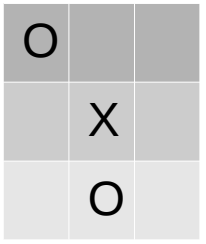


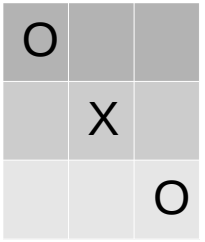
1 all path to winng
0 otherwise
-1 all path to loosing

-1

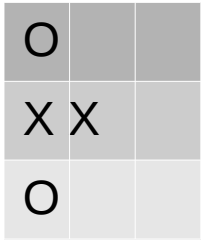


-1

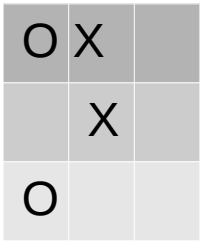




0



-1



MinMax Evaluation Prediction

float $\boldsymbol{H_0}$ *(state s);*

float $\boldsymbol{H_L}$ *(state s, int l){*
if (l==0) return $\boldsymbol{H_0}$*(s);*
if (l even) return max({$\boldsymbol{H_L}$ *(x, l-1) : x in neighbors(s)});*
if (l odd) return min({$\boldsymbol{H_L}$ *(x, l-1) : x in neighbors(s)});*
}

state MinMax (state s, int L){
return argmax$\boldsymbol{H_L}$*(neighbors(s),L-1);*
}

Average MinMax Search

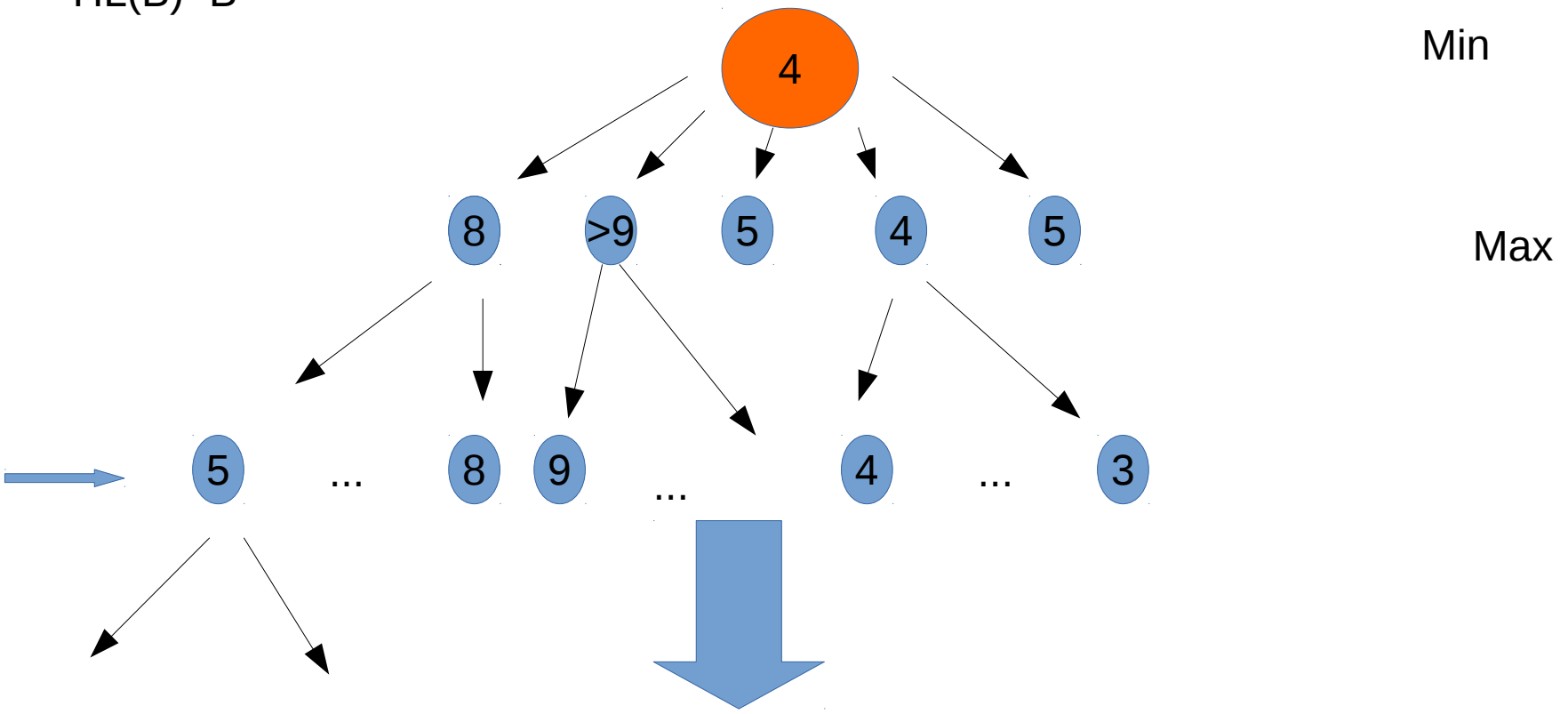
float $\boldsymbol{H_0}$ *(state s);*

float $\boldsymbol{H_L}$ *(state s, int l){*
if (l==0) return $\boldsymbol{H_0}$*(s);*
if (l even) return max({$\boldsymbol{H_L}$ *(x, l-1) : x in neighbors(s)});*
if (l odd) return average({$\boldsymbol{H_L}$ *(x, l-1) : x in neighbors(s)});*
}

state MinMax (state s, int L){
return argmax$\boldsymbol{H_L}$*(neighbors(s));*
}

MinMax Search tree

$HL(B)=B^L$



Problems

Exponential Branching

Horizon Effect

MinMax Search

```
float H_L (state s, int l){
    if (l==0) return H_0(s);

        possiblemoves=neighbors(s);
        possiblemoves =-  (bestmove=first(possiblemoves));
        bestvalue=H_L(bestmove, l-1)
        while (possiblemoves != Æ) {
            possiblemoves =-  (nextmove=first(possiblemoves));
            if (H_L(nextmove, l-1) > bestvalue) && (l even)
                        {bestvalue=H_L(nextmove, l-1);
                         bestmove=nextmove};
              if (H_L(nextmove, l-1) < bestvalue) && (l odd)
                        {bestvalue=H_L(nextmove, l-1);
                         bestmove=nextmove};
        }
return(bestvalue)
```

AlphaBeta Pruning

```
float H_L  (state s, int l,float alfa){
      if (l==0) return H_0 (s);
         possiblemoves=neighbors(s);
         possiblemoves =-  (bestmove=first(possiblemoves));
         bestvalue=H_L (bestmove, l-1)
         while (possiblemoves != Æ) {
             possiblemoves =-  (nextmove=first(possiblemoves));
             if ((H_L (nextmove, l-1,bestvalue) > bestvalue) && (l even) )
                  {bestvalue=H_L L(nextmove, l-1,bestvalue); bestmove=nextmove};
             if ((H_L (nextmove, l-1,bestvalue ) < bestvalue) && (l odd))
                  {bestvalue=H_L (nextmove, l-1,bestvalue); bestmove=nextmove};
             if ((bestvalue < alfa) && (l odd)) break;
             if ((bestvalue > alfa) && (l even)) break;
          }
return(bestvalue)
```