Chapter 12
Adversarial Search

Two-Agent Games (1)

- Idealized Setting
  - The actions of the agents are interleaved.

- Example
  - Grid-Space World
  - Two robots: “Black” and “White”
  - Goal of Robots
    - White: to be in the same cell with Black
    - Black: to prevent this from happening
  - After settling on a first move, the agent makes the move, senses what the other agent does, and then repeats the planning process in sense/plan/act fashion.

Two-Agent Games (2)

- Two-agent, perfect information, zero-sum games
- Two agents move in turn until either one of them wins or the result is a draw.
- Each player has a complete model of the environment and of its own and the other’s possible actions and their effects.
Minimax Procedure (1)

- Two player: MAX and MIN
- Task: find a “best” move for MAX
- Assume that MAX moves first, and that the two players move alternately.
- MAX node
  - nodes at even-numbered depths correspond to positions in which it is MAX’s move next
- MIN node
  - nodes at odd-numbered depths correspond to positions in which it is MIN’s move next

Minimax Procedure (2)

- Complete search of most game graphs is impossible.
  - For Chess, $10^{40}$ nodes
    - $10^{22}$ centuries to generate the complete search graph
    - assuming that a successor could be generated in 1/3 of a nanosecond
    - The universe is estimated to be on the order of $10^8$ centuries old.
  - Heuristic search techniques do not reduce the effective branching factor sufficiently to be of much help.
- Can use either breadth-first, depth-first, or heuristic methods, except that the termination conditions must be modified.

Minimax Procedure (3)

- Estimate of the best-first move
  - applying a static evaluation function to the leaf nodes
  - measure the “worth” of the leaf nodes.
  - The measurement is based on various features thought to influence this worth.
  - It is customary in analyzing game trees to adopt the convention
    - game positions favorable to MAX cause the evaluation function to have a positive value
    - positions favorable to MIN cause the evaluation function to have negative value
    - Values near zero correspond to game positions not particularly favorable to either MAX or MIN.

Minimax Procedure (4)

- Good first move extracted
  - Assume that MAX were to choose among the tip nodes of a search tree, he would prefer that node having the largest evaluation.
    - The backed-up value of a MAX node parent of MIN tip nodes is equal to the maximum of the static evaluations of the tip nodes.
  - MIN would choose that node having the smallest evaluation.
Minimax Procedure (5)

- After the parents of all tip nodes have been assigned backed-up values, we back up values another level.
  - \( MAX \) would choose that successor \( MIN \) node with the largest backed-up value
  - \( MIN \) would choose that successor \( MAX \) node with the smallest backed-up value.
  - Continue to back up values, level by level from the leaves, until the successors of the start node are assigned backed-up values.

Example : Tic-Tac-Toe (1)

- \( MAX \) marks crosses and \( MIN \) marks circles and it is \( MAX \)'s turn to play first.
  - With a depth bound of 2, conduct a breadth-first search
  - evaluation function \( e(p) \) of a position \( p \)
    - If \( p \) is not a winning for either player,
      \[ e(p) = (\text{no. of complete rows, columns, or diagonals that are still open for } MAX) - (\text{no. of complete rows, columns, or diagonals that are still open for } MIN) \]
    - If \( p \) is a win of \( MAX \),
      \[ e(p) = \infty \]
    - If \( p \) is a win of \( MIN \)
      \[ e(p) = -\infty \]

Example : Tic-Tac-Toe (2)

- First move

Example : Tic-Tac-Toe (3)
The Alpha-Beta Procedure (1)

- Only after tree generation is completed does position evaluation begin ⇒ inefficient
- Remarkable reductions in the amount of search needed are possible if perform tip-node evaluations and calculate backed-up values simultaneously with tree generation.
- After the node marked A is generated and evaluated, there is no point in generating nodes B, C, and D.
  - MIN has A available and MIN could prefer nothing to A.

The Alpha-Beta Procedure (2)

- Alpha value
  - depending on the backed-up values of the other successors of the start node, the final backed-up value of the start node may be greater than -1, but it cannot be less
- Beta value
  - depending on the static values of the rest of node successors, the final backed-up value of node can be less than -1, but it cannot be greater
- Note
  - The alpha values of MAX nodes can never decrease.
  - The beta values of MIN nodes can never increase.

The Alpha-Beta Procedure (3)

- Rules for discontinuing the search
  1. Search can be discontinued below any MIN node having a beta value less than or equal to the alpha value of any of its MAX node ancestors. The final backed-up value of this MIN node can be set to its beta value.
  2. Search can be discontinued below any MAX node having an alpha value greater than or equal to the beta value of any of its MIN node ancestors. The final backed-up value of this MAX node can be set to its alpha value.
**The Alpha-Beta Procedure (4)**

- **How to compute alpha and beta values**
  - The alpha value of a MAX node is set equal to the current largest final backed-up value of its successors.
  - The beta value of a MIN node is set equal to the current smallest final backed-up value of its successors.

- **Cut-off**
  - Alpha cut-off
    - search is discontinued under rule 1.
  - Beta cut-off
    - search is discontinued under rule 2.

- **Alpha-Beta Procedure**
  - The whole process of keeping track of alpha and beta values and making cut-offs when possible

**The Alpha-Beta Procedure (5)**

- **Pseudocode**

```
\texttt{Pseudocode}
\begin{align*}
&AB(n; \alpha, \beta) \\
&1: \text{ if } n \text{ at depth bound, return } AB(n) = \text{ static evaluation of } n. \text{ Otherwise, let } \\
&\pi_1, \ldots, \pi_k, \ldots, \pi_h \text{ be the successors of } n \text{ (in order), set } k \leftarrow 1 \text{ and if } n \text{ is a} \\
&M\text{AX node, go to step } 2; \text{ else go to step } 2'. \\
&2: \text{ Set } \alpha \leftarrow \min(\alpha, AB(\pi_k; \alpha, \beta)). \\
&3: \text{ If } \alpha \geq \beta, \text{ return } \alpha; \text{ else continue.} \\
&4: \text{ If } k = h, \text{ return } \alpha; \text{ else proceed to } \\
&\pi_{k+1}, \text{ i.e., set } k \leftarrow k + 1 \text{ and go to step } 2. \\
&2': \text{ Set } \beta \leftarrow \max(\beta, AB(\pi_k; \alpha, \beta)). \\
&3': \text{ If } \beta \leq \alpha, \text{ return } \beta; \text{ else continue.} \\
&4': \text{ If } k = h, \text{ return } \beta; \text{ else proceed to } \\
&\pi_{k+1}, \text{ i.e., set } k \leftarrow k + 1 \text{ and go to step } 2'.
\end{align*}
```

**Search Efficiency (1)**

- **Notation**
  - \( b \): depth of tree
  - \( d \): number of successors of every node (except a tip node)
  - \( b^d \): number of tip nodes
  - Suppose that an alpha-beta procedure generated successors in the order of their true backed-up values.
    - This order maximizes the number of cut-offs that will minimize the number of tip nodes generated.
  - \( N_f \): this minimal number of tip nodes
Search Efficiency (2)

- [Slager & Dixon 1969, Knuth & Moore 1975]
  
  \[ N_d = \begin{cases} 
  2b^{d/2} - 1 & \text{for even } d \\
  b^{(d+1)/2} + b^{(d-1)/2} - 1 & \text{for odd } d 
  \end{cases} \]

- The number of tip nodes of depth \( d \) that would be generated by optimal alpha-beta search is about the same as the number of tip nodes that would have been generated at depth \( d/2 \) without alpha-beta.

- Alpha-beta, with perfect ordering, reduces the effective branching factor from \( b \) to approximately \( \sqrt{b} \).

- [Pearl 1982]
  
  - The average branching factor is reduced to \( \frac{1}{\sqrt{b}} \) approximately.

Other Important Matters (1)

- Various Difficulties
  
  - Search might end at a position in which MAX (or MIN) is able to make a great move.
  
  - Make sure that a position is quiescent before ending search at that position.

  - Quiescent position
    
    - Its static value is not much different from what its backed-up value would be by looking a move or two ahead.

Search Efficiency (3)

- The most straightforward method for ordering successor nodes
  
  - to use the static evaluation function.

- Side effect of using a version of iterative deepening
  
  - Depending on the time resources available, search to deeper plies can be aborted at any time, and the move judged best by the search last completed can be made.

Other Important Matters (2)

- Horizon Effect
  
  - There can be situations in which disaster or success lurks just beyond the search horizon.

- Both minimax and alpha-beta extension assume that the opposing player will always make its best move.

  - There are occasions in which this assumption is inappropriate.

  - Minimax would be inappropriate if one player had some kind of model of the other player’s strategy.
Games of Chance (1)

- **Backgammon**
  - MAX's and MIN's turns now each involve a throw of the die.
  - Imagine that at each dice throw, a fictitious third player, DICE, makes a move.

![Diagram of Backgammon game tree]

Games of Chance (2)

- **Expectimaxing**
  - Back up the expected (average) values of the values of the successors instead of a maximum or minimum.
  - Back up the minimum value of the values of successors of nodes for which it is MIN's move, the maximum value of the values of successors of nodes for which it is MAX's move, and the expected value of the values of successors of nodes for which it is the DICE's move.
  - Introducing a chance move often makes the game tree branch too much for effective searching.
  - Important to have a good static evaluation function.

Learning Evaluation Functions (1)

- **TD-GAMMON**
  - Train Backgammon by training a layered, feedforward neural network.
  - Overall value of a board position
    
    \[ v = p_1 + 2p_2 - p_3 - 2p_4 \]

![Diagram of TD-GAMMON]

Learning Evaluation Functions (2)

- **Temporal difference training of the network**
  - Accomplished during actual play
    
    \[ \Delta W_i = c(v_{t+1} - v_t) \frac{\partial v_t}{\partial W} \]
  - \( v_{t+1} \): the estimate at time \( t + 1 \)
  - \( W \): vector of all weights at time \( t \)
  - Training method have been performed by having the network play many hundreds of thousands of games against itself.
  - Performance of a well-trained network is at or near championship level.