Game Playing

Game Tree Search

- Zero Sum Games
- Game Trees
- Minimax
- Evaluation Functions
- Alpha-Beta Minimax
- Variation of Minimax
- Optimizations

Why study game playing?

- Games allow us to experiment with easier versions of real-world situations
- Hostile agents act against our goals
- Games have a finite set of moves
- Games are fairly easy to represent
- Good idea to decide about what to think
- Perfection is unrealistic, must settle for good
- One of the earliest areas of AI
- Claude Shannon and Alan Turing wrote chess programs in 1950s
- The opponent introduces uncertainty
- The environment may contain uncertainty (backgammon, poker)
- Search space too hard to consider exhaustively
- Chess has about 10^{40} legal positions with 10^{154} nodes in the search tree
- Efficient and effective search strategies even more critical
- Games are fun to target!

Types of Games

	Deterministic	Stochastic
Perfect Information	chess, checkers, go, othello	backgammon, monopoly
Imperfect Information	Wumpus world	bridge, poker, scrabble, slots, nuclear war

Zero Sum Games

- Focus primarily on "adversarial games", in particular on two-player, zero-sum games

As Player 1 gains strength

$\longrightarrow$

Player 2 loses strength

$\longleftarrow$

- and vice versa. The sum of the two strengths is always 0.

Search Applied to Adversarial Games

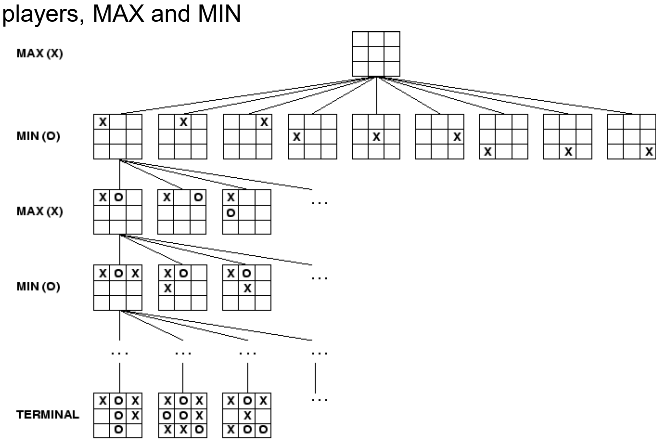
- Initial state
  Current board position (description of current game status)
- Operators
  Legal moves a player can make
- Terminal nodes
  leaf nodes in the tree, these indicate the game is over
- Utility function
  payoff function, value of the outcome of a game
  Tic-tac-toe, utility is -1, 0, or 1.
- Many games represent idealistic simulations of real-world functions (chess represents war) and are easier to form as search problems than real-world problems

Using Search

- Search could be used to find a perfect sequence of moves except the following problems arise:
  - There exists an adversary who is trying to minimize your chances of winning every other move. You cannot control his/her move.
  - Search trees can be VERY large
    - chess has 10^{40} nodes in the search space
  - Players have a finite time to move.
    - With single-agent search (15 puzzle), can afford to wait
    - Some 2-player games have time limits
  - Solution? Search to *n* levels in the tree (*n* “ply”), evaluate the nodes at the *n*th level, and head for the best looking node

Game Trees

- Two players, MAX and MIN

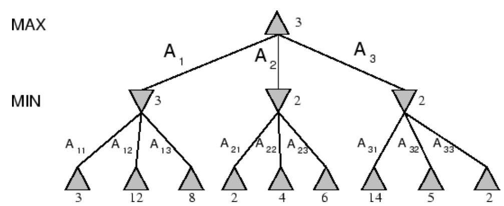


- In this case, assume we are searching ahead 5 moves (ply=5) Moves (and levels) alternate between two players

Minimax Algorithm

- Search the tree to the end
- Assign utility values to terminal nodes
- Find the best move for MAX (this is MAX's turn) assuming:
  MAX will make the move that maximizes utility
  MIN will make the move that minimizes MAX's utility

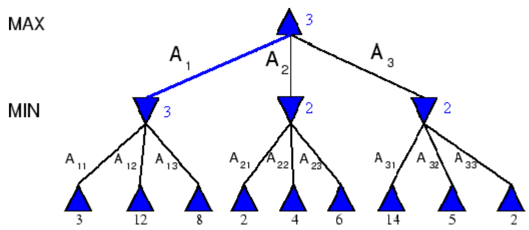
- MAX should take action A_1.



Minimax Algorithm

```
Function Minimax(state)
   v = Max-Value(state)
   return action in Successors(state) with value v

Function Max-Value(state)
   if Terminal-Test(state)
        return Utility(state)
   v = -INF
   for ∀s ∊ Successors(state)
        v = max(v, Min-Value(s))
   return v

Function Min-Value(state)
   if Terminal-Test(state)
        return Utility(state)
   v = INF
   for ∀s ∊ Successors(state)
        v = min(v, Max-Value(s))
   return v
```

Examples



Best action to take is A_1.

Minimax Properties

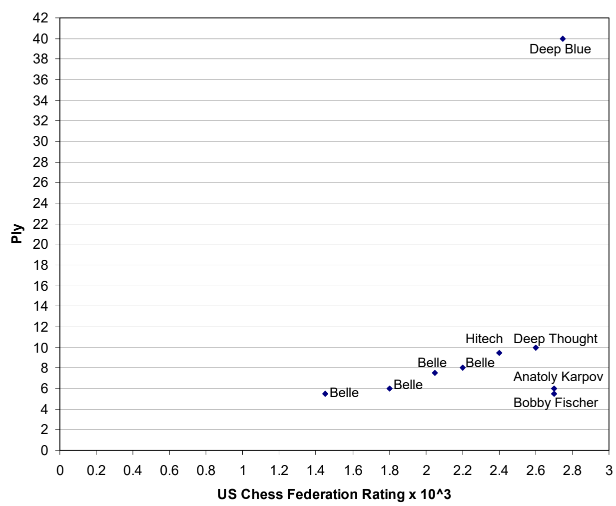
- Complete, if tree is finite
- Optimal, if play against optimal opponent (or opponent with same strategy)
- Time complexity is $O(b^m)$
- Space complexity is $O(bm)$ (depth-first exploration)
- If we have 100 seconds to make a move and we can explore 10^4 nodes/second, then we can consider 10^6 nodes per move

- Standard approach is
  - Apply a cutoff test (depth limit, quiescence)
  - Evaluate nodes at cutoff (heuristic evaluation function estimates desirability of position)

Static Board Evaluator (Evaluation Function)

- Because we can't look all the way to the end of the game, look ahead n-ply moves, evaluate the nodes there using a “static board evaluator” (SBE).
- Example for Tic-Tac-Toe: Number of unblocked lines with ‘X’s minus Number of unblocked lines with ‘O’s
- There is a tradeoff between:
  Stupid, fast SBE; Massive Search
  These are Type “A” systems
  vs.
  Smart, slow SBE; Very Little Search
  These are Type “B” systems
- Humans are type “B” systems
  - For chess, 4 ply is human novice, 8 ply is human master, 12 ply is grand master
- Computer chess systems have been much more successful using type “A” approach.
  - They get better by searching more ply

Comparisons



Properties of Evaluation Functions

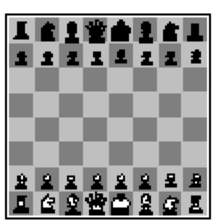
- Performance of a game playing program is dependent on the quality of its evaluation function.
- A good evaluation function should:
  - Order the terminal states
  - Be fast
  - For nonterminal states, the function should return a good approximation of the chances of winning

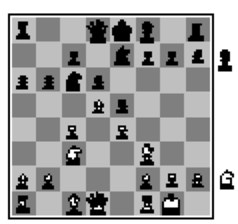
Example: Chess

- SBE is typically linear weighted sum of set of features

  $Eval(s) = w_1f_1(s)+w_2f_2(s)+\ldots+w_nf_n(s)$
- For chess, these weights could be:
  - Pawn = 1
  - Knight/Bishop = 3
  - Rook = 5
  - Queen 9
- With the functions being the difference in the number of pieces between the sides

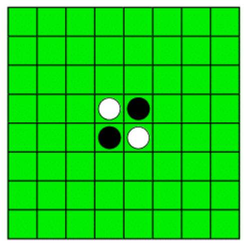
  $f_1(s)$ = (number of white queens) - (number of black queens)





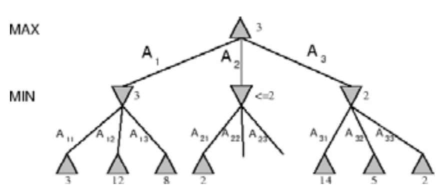
Example: Othello

- SBE1: number of white pieces - number of black pieces
- SBE2: weighted squares



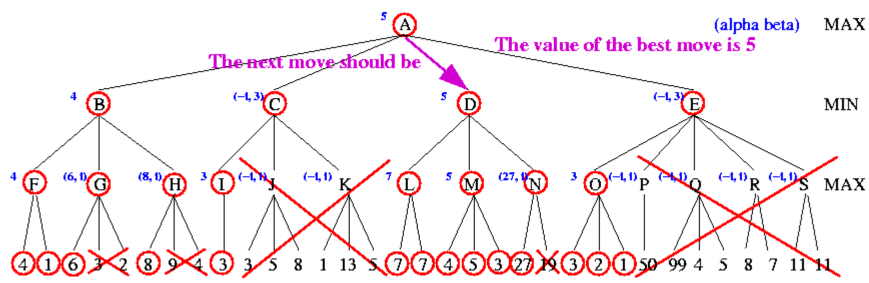
Alpha-Beta Pruning

- If time limits ply then Alpha-Beta pruning simplifies the search space without eliminating optimality by applying common sense
- For example:
  - In chess, if one route allows the queen to be captured, and a better move is available, don't search further down bad route.
  - If one route assumes a stupid move by opponent, ignore the route.



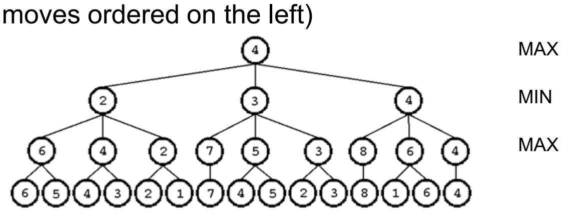
- Maintain [alpha, beta] window at each node during depth-first search alpha = lower bound on max value, beta = upper bound on min value.

Example

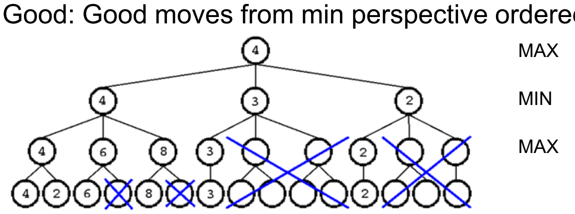


Bad and Good Cases for Alpha-Beta Pruning

- Bad: Worst moves from min perspective encountered first (worst moves ordered on the left)



- Good: Good moves from min perspective ordered first

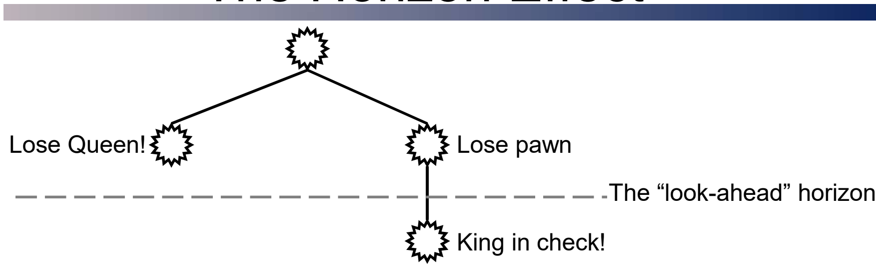


- If we can order moves, we get more benefit from alpha-beta pruning.

Alpha-Beta Properties

- Pruning does not affect final result
- Good move ordering improves effectiveness of pruning
- With perfect ordering, time complexity is O($b^{m/2}$) otherwise O($b^{3m/4}$)
- Odd/Even Issue
  - (optimistic vs pessimistic)

Problems with a fixed ply: The Horizon Effect



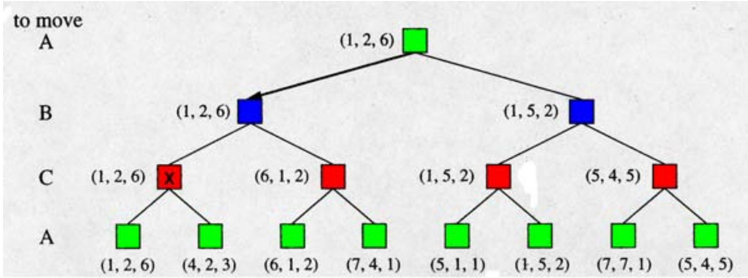
- Inevitable losses are postponed or unachievable goals appear achievable
- Short-term gains mask unavoidable consequences (traps)

Solutions

- How to counter the horizon effect
  - Feedover
    - Do not cut off search at non-quiescent board positions (dynamic positions)
    - Example, king in danger
    - Keep searching down the path until reach quiescent (stable) nodes
    - Realization Probability Search (RPS)
  - Secondary Search
    - Search further down selected path to ensure this is best move
  - Progressive Deepening (IDS)
    - Search one ply, then two ply, etc., until run out of time (similar to IDS)

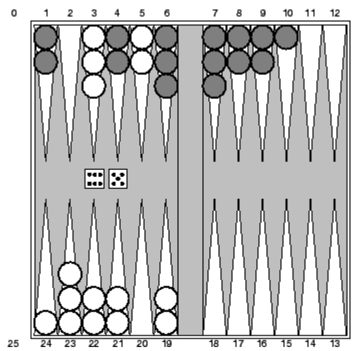
Variations on 2-Player Games

- 3-player games
  - 1) each player maximizes his/her utility
  - 2) each node stores a vector of utilities
  - 3) entire vector is backed up

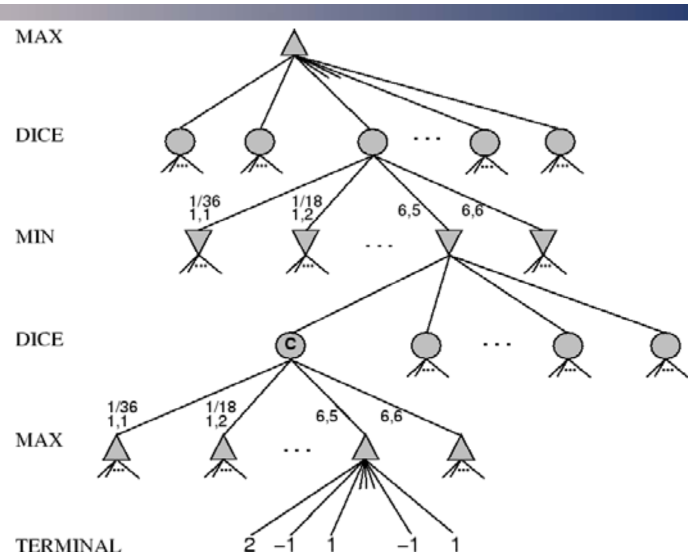


Nondeterministic Games

- In backgammon, the dice rolls determine legal moves



Nondeterministic Games



Status of AI Game Players

- Tic-Tac-Toe - Tied for best player in world (with every human over age 12)
- Othello – Logistello beat world champion Takeshi Murakami
  Computer better than any human, human champions now refuse to play the computer
- Scrabble - Maven beat world champions Joel Sherman and Matt Graham
- Backgammon – TD-Gammon plays near level of world's strongest grandmasters
  Uses a three-ply search, neural network with 160 hidden units
  Search is expensive in backgammon, because all possible dice rolls must be considered
- Bridge – Gib is ranked among top players in the world.
- Poker – Pokie plays at a strong intermediate level.

Complexity of Games

	State-space complexity ($\log_{10}$)	Game-tree complexity ($\log_{10}$)
Nine men's morris	10	50
Awari	12	32
Connect four	14	21
Checkers	18	31
Crossings	23	79
Lines of Action (6x6)	24	56
Othello (6x6)	28	58
Qubic	30	34
Draughts	30	54
Amazons	40	220
Chinese Chess (Xangqi)	48	150
Chess	50	123
Hex	57	150
Epaminondas	61	137
Shogi	71	105
Go-Moku (Pente)	105	70
Renju	105	140
Go (5x5)	172	360

Examples

- Checkers
  Chinook ended 40-year reign of human champion Marion Tinsley in 1994
  Used endgame database for all positions involving 8 or fewer pieces on the board.

- Chess
  Deep Blue beat human champion Gary Kasparov in six-game match in 1997
  Deep Blue searches 200M positions/second, and searches up to 40 ply

- Go
  Human champions have only recently (2007) competed against computers, which are not yet at a strong level.

Optimization of Search
(Single Agent and Game)

- Move Ordering with Iterative Deepening
- Transposition Tables
- History Heuristic
- Killer Move Heuristic
- Databases
  - Opening Move Database
  - End Game Database
  - Pattern Database

Transposition Tables

- Position in table and was previously searched *d* ply deep, but current position requires a search to depth *d'*
  - *d'* < *d*: a more accurate result than is needed is available for use!
  - *d'* = *d*: the appropriate accuracy is available
  - *d'* > *d*: the entry is not accurate enough to use
- If $V <= \alpha < \beta$, then V is an upper bound on the correct value
- If $\alpha < V < \beta$, then V is an accurate value
- If $\alpha < \beta <= V$, then V is a lower bound on the correct value

Saving a State

```
void TTSave( state s; int value; int alpha; int beta; int depth ) {
    if( value <= alpha )
       bound = UPPER;
    else if( value >= beta )
       bound = LOWER;
    else bound = ACCURATE;
    AddToTT( s, value, bound, depth );
}
```

When Storing Track the:

- State (Zobrist Hash)
- Search result
- Value
- Bound
- Accuracy
- Best move (for use later on)

Checking the TT

```
ptr = TTLookup( state );
if( ptr != NULL && ptr->depth >= d ) {
    if( ptr->bound == LOWER )
        alpha = MAX( alpha, ptr->value );
    if( ptr->bound == UPPER )
        beta = MIN( beta, ptr->value );
    if( ptr->bound == ACCURATE )
        alpha = beta = ptr->value;
    if( alpha >= beta ) /* TT causes a cutoff */
        return( ptr->value );
}
```

Using the TT

```
int AlphaBeta( state s, int alpha, int beta, int depth ) {
    if( terminal node || depth == 0 ) return( Evaluate( s );
    /* Look in TT before searching */
    ptr = TTLookup( s );
    …
    /* If no cutoff, search */
    …
    /* Save TT result before returning */
    TTSave( s, value, alpha, beta, depth );
    return( value );
}
```

A 15-Puzzle Experiment

- IDA*
- Transposition Table
  - 2^18 Entries
- End Game Database
  - All Position <= 22 moves from end
- Pattern Database
  - All Subset of 8 tiles
- 1707-fold improvement!

IDA*	36,302,808,031	100.0
+ TT	13,662,973,000	37.64
+DB	19,419,742,608	53.49
+TT +DB	8,869,627,254	24.43
+ PDB	34,987,894	0.10
+TT +DB +PDB	21,261,747	0.06

J. Culberson and J. Schaeffer. “Pattern Databases”, *Computational Intelligence*, vol. 14, no. 4, pp. 318-334,1998.

Evaluation Features for Games

- Mobility
- Piece Count
- Square Control
- Piece Control
- Automatic Discovery
- Are Features Independent?

Newer Searches

- Aspiration Window
- MTD(f) – Principle Variation Search (PVS)
- Proof Number Search (pn, pn^2, PDS, PNS*)
- Threat Search
- Null Move Search
- ProbCut
- UCB for Trees (UCT)
- Enhanced Realization Probability Search (ERPS)

Proof Number (pn) Search

```
Function ProofNumberSearch(state)
   Evaluate(state)
   SetProofAndDisproofNumbers(state)
   while state.proof ≠ 0 and state.disproof ≠ 0 and ResourcesAvailable() do
    mostProvingNode = SelectMostProving(state)
    DevelopNode(mostProvingNode)
    UpdateAncestors(mostProvingNode)
   if state.proof = 0
    state.value = true
   else if state.disproof = 0
    state.value = false
   else
    state.value = unknown

Function Evaluate(state) // end game test
   assigns state.value one of true, false, unknown
   state.evaluated = true

Function GenerateAllChildren(state)
   state.numberOfChildren is set
   state.children[1..state.numberOfChildren] point to children
   state.expanded = true
```

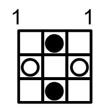
pn Search (cont')

```
Function SelectMostProving(state)
   while state.expanded
    case state.type
        or:
            i = 1
            while state.children[i].proof ≠ state.proof
                i = i + 1
        and:
            i = 1
            while state.children[i].disproof ≠ state.disproof
                i = i + 1
    state = state.children[i]
   return state

Function DevelopNode(state)
   GenerateAllChildren(state)
   for i = 1 to state.numberOfChildren
    Evaluate(state.children[i])
    SetProofAndDisproofNumbers(state.children[i])
```

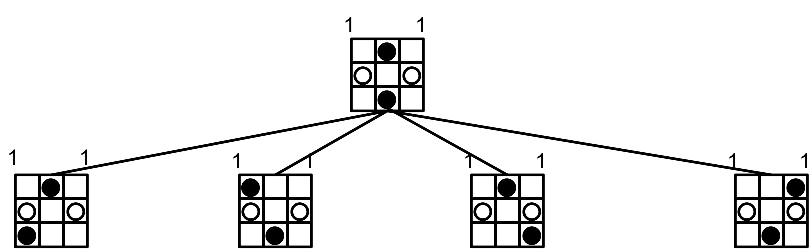
pn Search (cont')

```
Function SetProofAndDisproofNumbers(state)
   if state.expanded
    case state.type
        or:
            state.proof = min_{N∈Children(state)} N.proof
            state.disproof = Σ_{N∈Children(state)} N.proof
        and:
            state.proof = Σ_{N∈Children(state)} N.proof
            state.disproof = min_{N∈Children(state)} N.proof
   else if state.evaluated
    case state.value
        true:    state.proof = 0; state.disproof = INF
        false:   state.proof = INF; state.disproof = 0
        unknown: state.proof = state.disproof = 1
   else
    state.proof = state.disproof = 1

Function UpdateAncestors(state)
   while state ≠ nil
    SetProofAndDisproofNumbers(state)
    state = state.parent
```

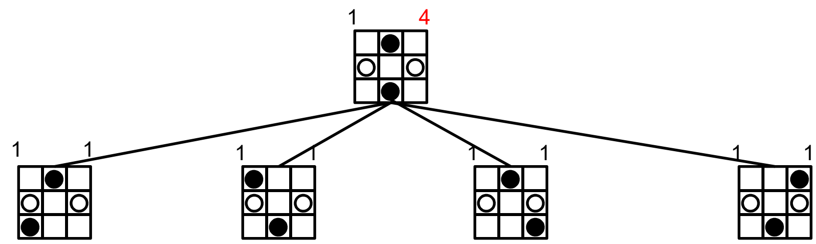
pn Search Example



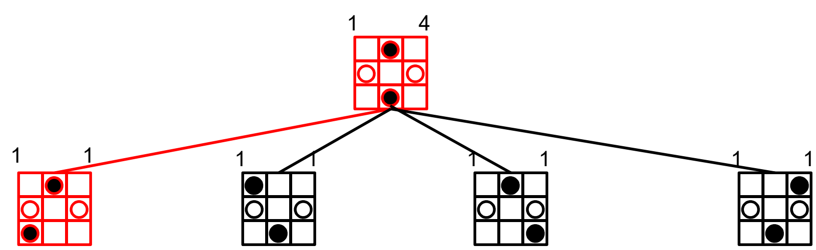
pn Search Example



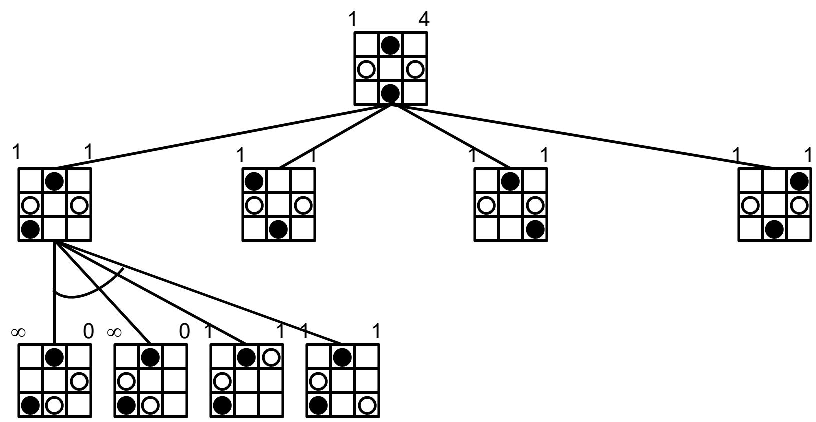
pn Search Example



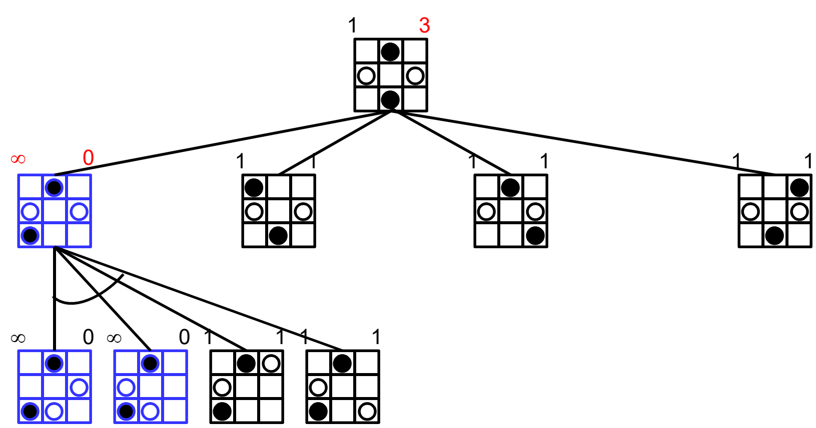
pn Search Example



pn Search Example



pn Search Example



pn Search Example

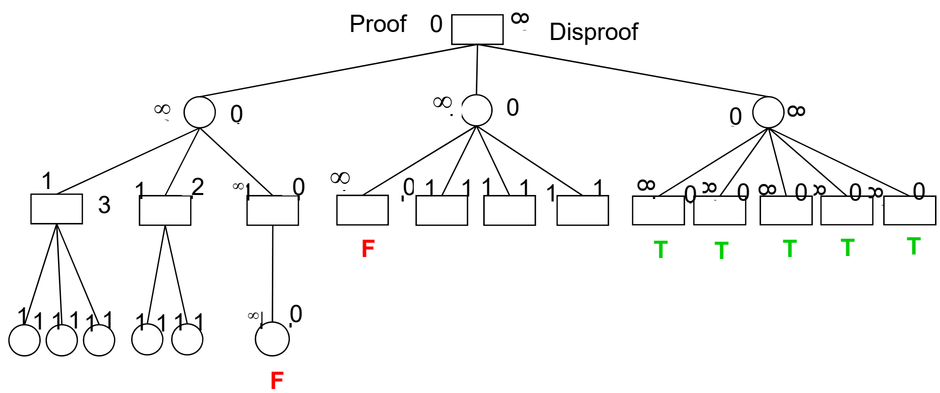
Best-First/Most Proving Path

OR-Selects child with lowest proof

AND-Selects child with lowest disproof

OR

AND



Review

- Game Playing
  - Min-Max Search Algorithm
  - Alpha-Beta Pruning
  - Evaluation Functions
  - Nondeterministic Games
  - Search Optimizations
- Games Illustrate
  - Perfection is unattainable – we must approximate
  - Its good to think about what to think about
  - Uncertainty constrains the assignment of values to states

Next Time

- Representation and Reasoning
  - At which point we introduce and discuss the other side of the AI coin – how to represent knowledge information in a computer.