Adversarial Search and Game Playing

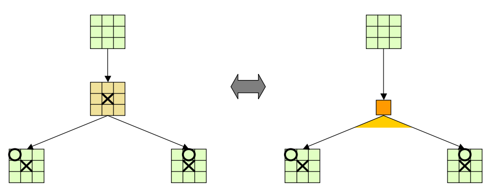
(Where making good decisions requires respecting your opponent)

R&N: Chap. 6

- Games like Chess or Go are compact settings that mimic the uncertainty of interacting with the natural world
- For centuries humans have used them to exert their intelligence
- Recently, there has been great success in building game programs that challenge human supremacy

Relation to Previous Lecture

- Here, uncertainty is caused by the actions of another agent (MIN), who competes with our agent (MAX)



Relation to Previous Lecture

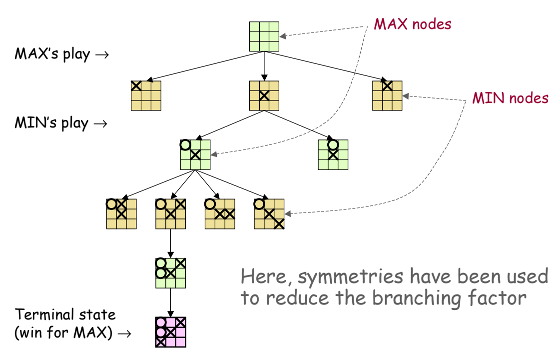
- Here, uncertainty is caused by the actions of another agent (MIN), who competes with our agent (MAX)
- MIN wants MAX to fail (and vice versa)
- No plan exists that guarantees MAX's success regardless of which actions MIN executes (the same is true for MIN)
- At each turn, the choice of which action to perform must be made within a specified time limit
- The state space is enormous: only a tiny fraction of this space can be explored within the time limit

Specific Setting

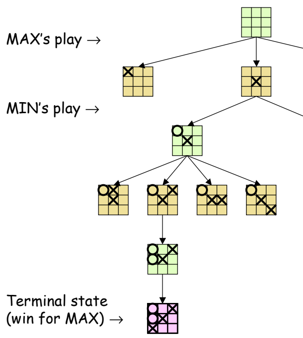
Two-player, turn-taking, deterministic, fully observable, zero-sum, time-constrained game

- State space
- Initial state
- Successor function: it tells which actions can be executed in each state and gives the successor state for each action
- MAX's and MIN's actions alternate, with MAX playing first in the initial state
- Terminal test: it tells if a state is terminal and, if yes, if it's a win or a loss for MAX, or a draw
- All states are fully observable

Game Tree



Game Tree



In general, the branching factor and the depth of terminal states are large

Chess:
- Number of states: ~10^{40}
- Branching factor: ~35
- Number of total moves in a game: ~100

Choosing an Action: Basic Idea

1) Using the current state as the initial state, build the game tree uniformly to the maximal depth h (called horizon) feasible within the time limit
2) Evaluate the states of the leaf nodes
3) Back up the results from the leaves to the root and pick the best action assuming the worst from MIN

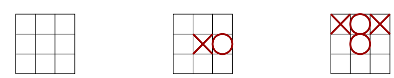
→ Minimax algorithm

Evaluation Function

- Function e: state s → number e(s)
- e(s) is a heuristics that estimates how favorable s is for MAX
- e(s) > 0 means that s is favorable to MAX (the larger the better)
- e(s) < 0 means that s is favorable to MIN
- e(s) = 0 means that s is neutral

Example: Tic-tac-Toe

e(s) = number of rows, columns, and diagonals open for MAX – number of rows, columns, and diagonals open for MIN



8–8 = 0 | 6–4 = 2 | 3–3 = 0

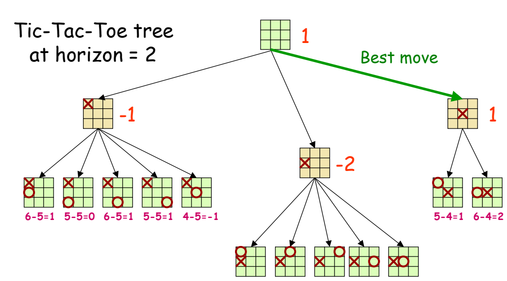
Construction of an Evaluation Function

- Usually a weighted sum of "features":

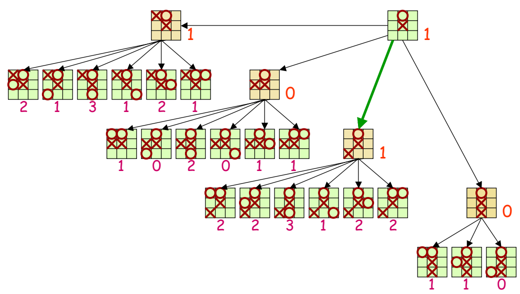
$$e(s)=\sum_{i=1}^{n} w_i f_i(s)$$

- Features may include
  - Number of pieces of each type
  - Number of possible moves
  - Number of squares controlled

Backing up Values



Continuation



Why using backed-up values?

- At each non-leaf node N, the backed-up value is the value of the best state that MAX can reach at depth h if MIN plays well (by the same criterion as MAX applies to itself)
- If e is to be trusted in the first place, then the backed-up value is a better estimate of how favorable STATE(N) is than e(STATE(N))

Minimax Algorithm

1. Expand the game tree uniformly from the current state (where it is MAX's turn to play) to depth h
2. Compute the evaluation function at every leaf of the tree
3. Back-up the values from the leaves to the root of the tree as follows:
   a. A MAX node gets the maximum of the evaluation of its successors
   b. A MIN node gets the minimum of the evaluation of its successors
4. Select the move toward a MIN node that has the largest backed-up value

Minimax Algorithm

1. Expand the game tree uniformly from the current state (where it is MAX's turn to play) to depth h
2. Compute the evaluation function at every leaf of the tree
3. Back-up the values from the leaves to the root of the tree as
   a. A MAX no
   successors
   b. A MIN node gets the minimum of the evaluation of its successors
4. Select the move toward a MIN node that has the largest backed-up value

Horizon: Needed to return a decision within allowed time

Repeated States

Left as an exercise

[Distinguish between states on the same path and states on different paths]

Game Playing (for MAX)

Repeat until a terminal state is reached

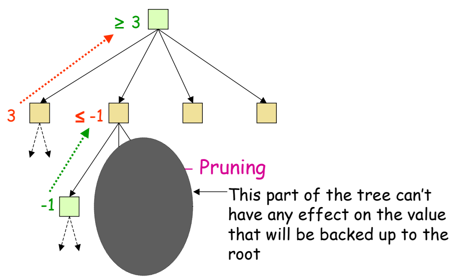
1. Select move using Minimax
2. Execute move
3. Observe MIN's move

Note that at each cycle the large game tree built to horizon h is used to select only one move

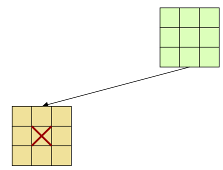
All is repeated again at the next cycle (a sub-tree of depth h-2 can be re-used)

Can we do better?

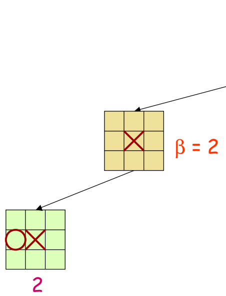
Yes ! Much better !



Example

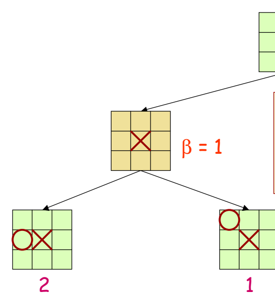


Example



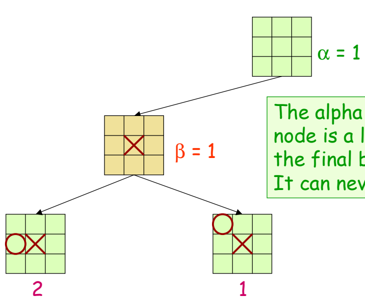
The beta value of a MIN node is an upper bound on the final backed-up value. It can never increase

Example



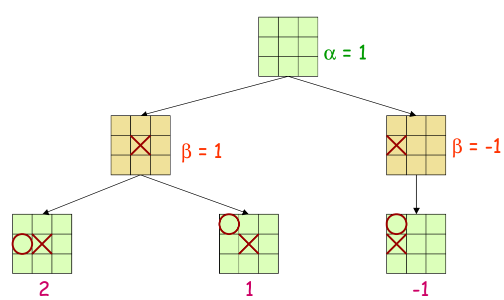
The beta value of a MIN node is an upper bound on the final backed-up value. It can never increase

Example

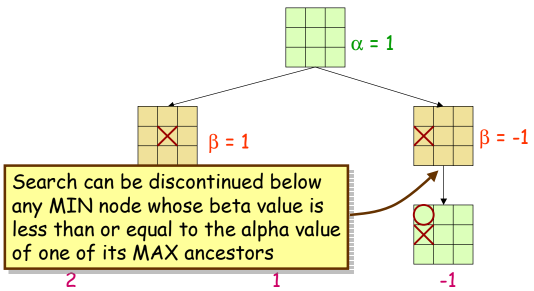


The alpha value of a MAX node is a lower bound on the final backed-up value. It can never decrease

Example



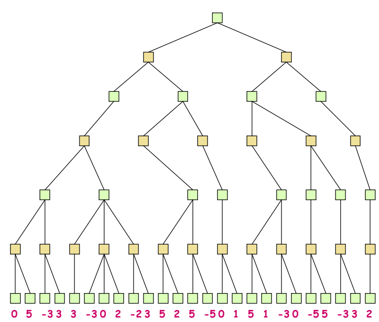
Example



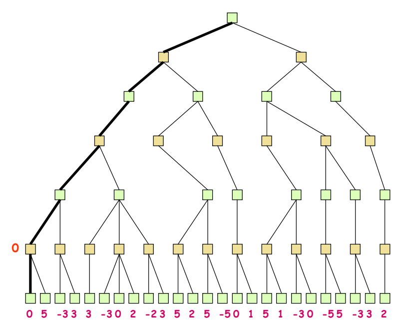
Alpha-Beta Pruning

- Explore the game tree to depth h in depth-first manner
- Back up alpha and beta values whenever possible
- Prune branches that can't lead to changing the final decision

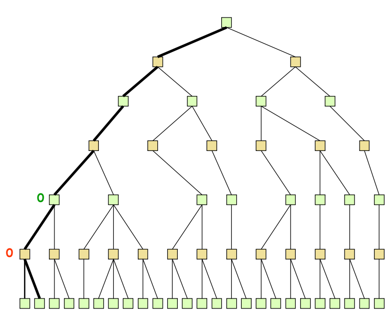
Example



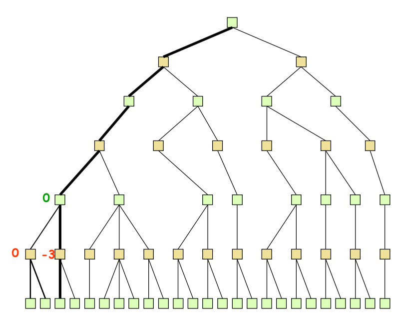
Example



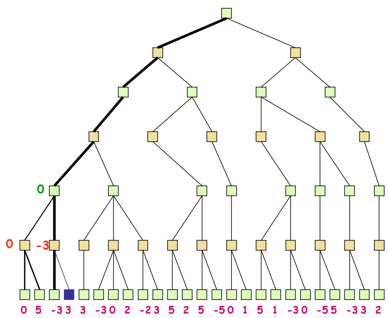
Example



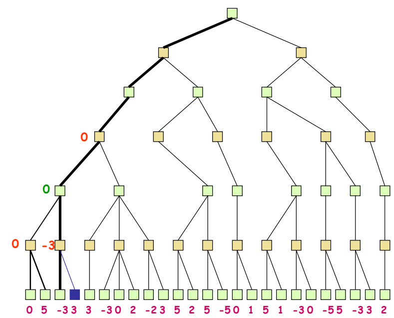
Example



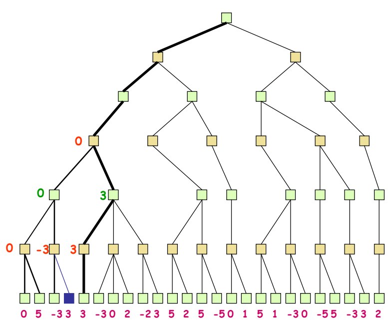
Example



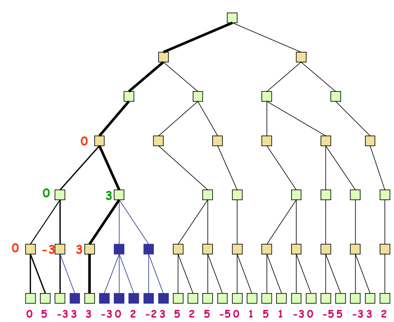
Example



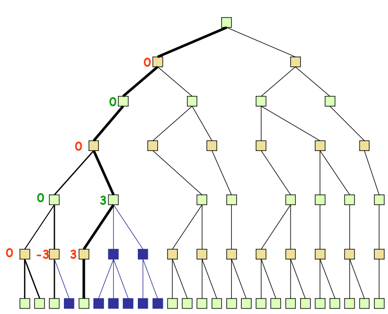
Example



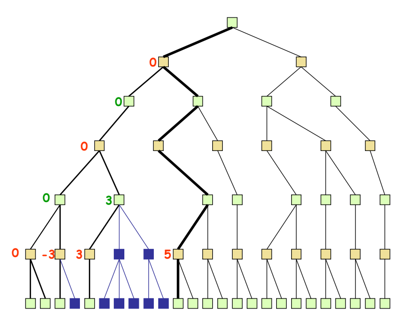
Example



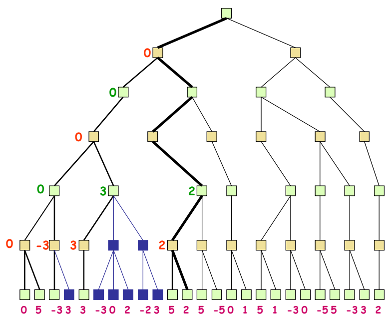
Example



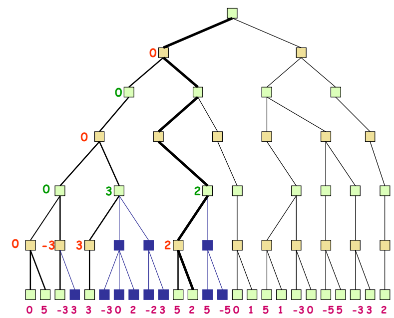
Example



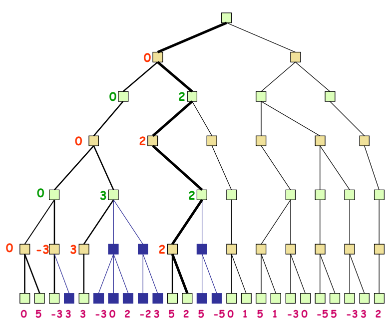
Example



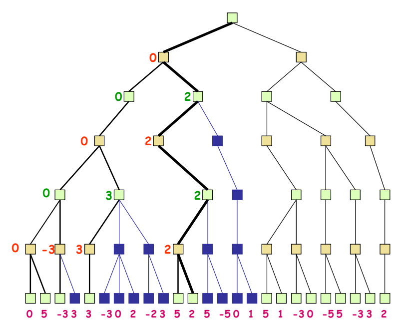
Example



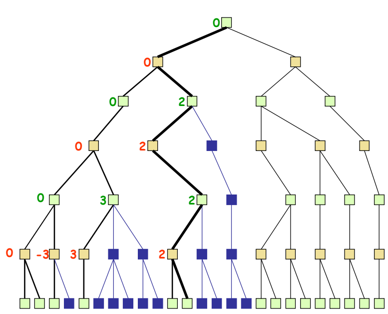
Example



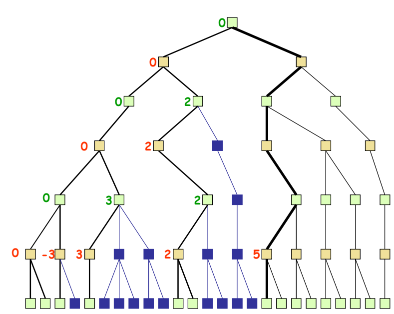
Example



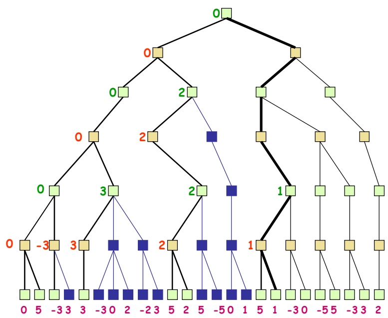
Example



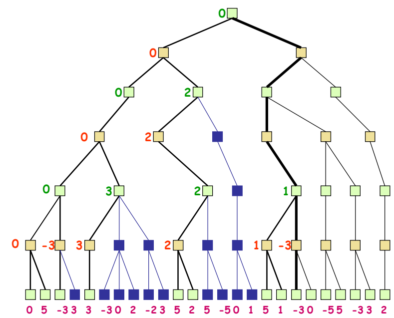
Example



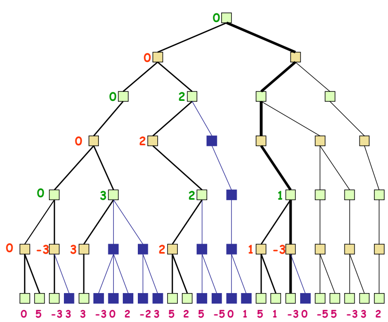
Example



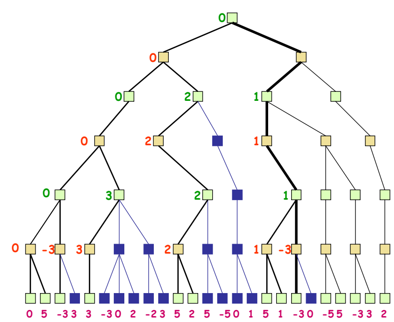
Example



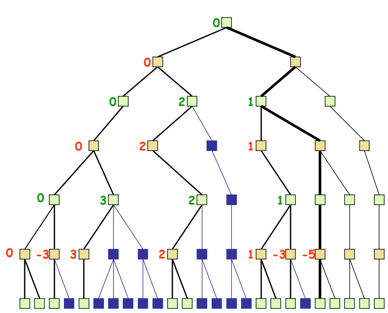
Example



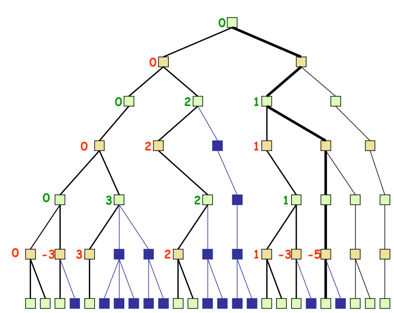
Example



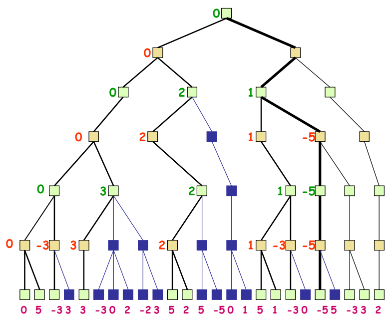
Example



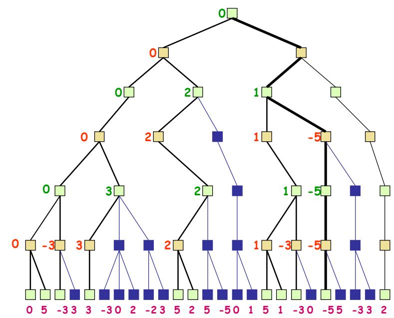
Example



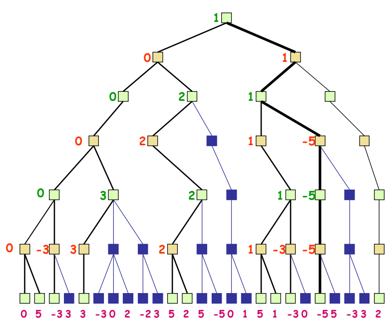
Example



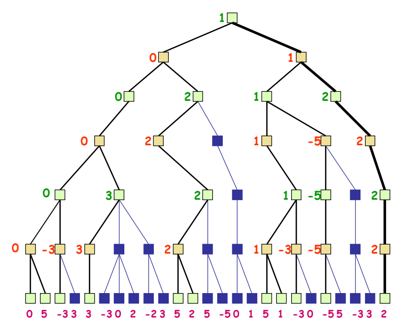
Example



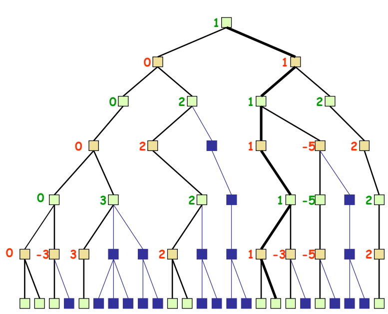
Example



Example



Example

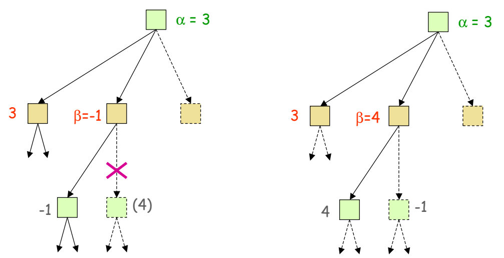


Alpha-Beta Algorithm

- Update the alpha/beta value of the parent of a node N when the search below N has been completed or discontinued
- Discontinue the search below a MAX node N if its alpha value is ≥ the beta value of a MIN ancestor of N
- Discontinue the search below a MIN node N if its beta value is ≤ the alpha value of a MAX ancestor of N

How much do we gain?

Consider these two cases:



How much do we gain?

- Assume a game tree of uniform branching factor b
- Minimax examines $O(b^h)$ nodes, so does alpha-beta in the worst-case
- The gain for alpha-beta is maximum when:
  - The MIN children of a MAX node are ordered in decreasing backed up values
  - The MAX children of a MIN node are ordered in increasing backed up values
- Then alpha-beta examines $O(b^{h/2})$ nodes [Knuth and Moore, 1975]
- But this requires an oracle (if we knew how to order nodes perfectly, we would not need to search the game tree)
- If nodes are ordered at random, then the average number of nodes examined by alpha-beta is $\sim O(b^{3h/4})$

Heuristic Ordering of Nodes

- Order the nodes below the root according to the values backed-up at the previous iteration

Other Improvements

- Adaptive horizon + iterative deepening
- Extended search: Retain k>1 best paths, instead of just one, and extend the tree at greater depth below their leaf nodes to (help dealing with the "horizon effect")
- Singular extension: If a move is obviously better than the others in a node at horizon h, then expand this node along this move
- Use transposition tables to deal with repeated states
- Null-move search

State-of-the-Art

Checkers: Tinsley vs. Chinook

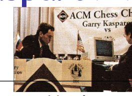
Name: Marion Tinsley
Profession: Teach mathematics
Hobby: Checkers
Record: Over 42 years loses only 3 games of checkers
World champion for over 40 years

Mr. Tinsley suffered his 4th and 5th losses against Chinook

Chinook

First computer to become official world champion of Checkers!

Chess: Kasparov vs. Deep Blue



Kasparov		Deep Blue
5'10"	Height	6' 5"
176 lbs	Weight	2,400 lbs
34 years	Age	4 years
50 billion neurons	Computers	32 RISC processors + 256 VLSI chess engines
2 pos/sec	Speed	200,000,000 pos/sec
Extensive	Knowledge	Primitive
Electrical/chemical	Power Source	Electrical
Enormous	Ego	None

1997: Deep Blue wins by 3 wins, 1 loss, and 2 draws

Jonathan Schaeffer

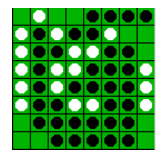
Chess: Kasparov vs. Deep Junior

Deep Junior

8 CPU, 8 GB RAM, Win 2000
2,000,000 pos/sec
Available at $100

August 2, 2003: Match ends in a 3/3 tie!

Othello: Murakami vs. Logistello



Takeshi Murakami
World Othello Champion

1997: The Logistello software crushed Murakami by 6 games to 0

Go: Goemate vs. ??

Name: Chen Zhixing
Profession: Retired
Computer skills:
self-taught programmer
Author of Goemate (arguably the best Go program available today)

Gave Goemate a 9 stone handicap and still easily beat the program, thereby winning $15,000

Jonathan Schaeffer

Go: Goemate vs. ??

Name: Chen Zhixing
Profession: Retired
Computer skills:

Go has too high a branching factor for existing search techniques

Current and future software must rely on huge databases and pattern-recognition techniques

thereby winning $15,000

Jonathan Schaeffer

Secrets

- Many game programs are based on alpha-beta + iterative deepening + extended/singular search + transposition tables + huge databases + ...
- For instance, Chinook searched all checkers configurations with 8 pieces or less and created an endgame database of 444 billion board configurations
- The methods are general, but their implementation is dramatically improved by many specifically tuned-up enhancements (e.g., the evaluation functions) like an F1 racing car

Perspective on Games: Con and Pro

Chess is the Drosophila of artificial intelligence. However, computer chess has developed much as genetics might have if the geneticists had concentrated their efforts starting in 1910 on breeding racing Drosophila. We would have some science, but mainly we would have very fast fruit flies.

John McCarthy

Saying Deep Blue doesn't really think about chess is like saying an airplane doesn't really fly because it doesn't flap its wings.

Drew McDermott

Other Types of Games

- Multi-player games, with alliances or not
- Games with randomness in successor function (e.g., rolling a dice)
  → Expectminimax algorithm
- Games with partially observable states (e.g., card games)
  → Search of belief state spaces

See R&N p. 175-180