Game Playing: Adversarial Search

Susmita Ghosh
Computer Science & Engineering Department
Jadavpur University
Kolkata
Email: susmitaghoshju@gmail.com


Kasparov vs. Deep Blue: The Rematch - Netscape
File Edit View Go Communicator Help
Bookmarks Location: http://www.chess.ibm.com/home/html/b.html
may 10th
game 6 : may 11 @ 3:00PM EDT | 19:00 GMT
kasparov 2.5 deep blue 2.5
Home
The match
The players
The technology
Community
Deep Blue Wins
3.5 to 2.5
the rematch
With a dramatic victory in Game 6, Deep Blue won its six-game rematch with Champion Garry Kasparov
OVERVIEW
EXPERT COVERAGE
MATCH NEWS
MAIN STORIES
Commentary
Commentary
Club Kasparov
Guest essays
Community
Clips from the rematch
Press room
Chess reference
Feedback
Site guide
IBM
Document Done

World champion Gary Kasparov was defeated by an IBM made chess program "DEEP BLUE" in 1997

Secrets of DEEP BLUE: high speed computation, **smart search strategies**, innovative techniques that partition the search problem for a multiprocessor system

----- hardware-software synergy

Games and Search Strategies

- Games often handle very large search spaces
  - Chess: 35^100 nodes; 10^40 legal states

- It is not possible to evaluate all possible nodes and then move a piece

- *Look ahead upto a certain depth*
- *Eliminate some questionable states*

Formulating Game Playing as Search

- Two-person, zero sum, perfect information game

  Zero-sum-- one player's loss is the other's gain

  Perfect information -- both players have access to complete information about the state of the game.
- Players move alternately
- No chance (e.g., using dice) factor
- Clear rules for legal moves (transition to no uncertain positions)
- Well defined outcomes (Win/Loss/Draw)

- Examples: Tic-Tac-Toe, Checkers, Chess, Go, Nim, Othello

and Not Ludo, Bridge, Solitaire, Backgammon, ...

How to play a game

- A way to play such a game is to:
  - Consider all the legal moves you can make
  - Each move leads to a new board configuration (position)
  - Evaluate each resulting position and determine the best move
  - Make that move
  - Wait for your opponent to move and repeat

  Iterative methods are applied (search space too large) –search will be done before each move in order to select the next best move

  Adversary methods are used
- Key problems are:
  - Representing the "board" configuration
  - Generating all legal next boards
  - Evaluating each board
  - **Look ahead**

Game Trees

- Problem spaces for typical games are represented as trees
- Root node represents the "board" configuration at which a decision must be made as to what is the **best single move to make next** (not necessarily the initial configuration)
- **Evaluation function, *f*,** rates a board configuration

  f(board) → a real number
- Arcs represent the possible legal moves for a player (no costs are associated with the arcs)
- Terminal nodes represent end-game configurations : the result must be "win"/"lose"/"draw"

AND-OR Tree

- Search processes studied so far represented by OR Graph

  - at any point of time only one alternative is processed

- But the problems can be of decomposable in nature and to obtain a solution, might be, all/some the nodes need to be processed ----- AND nodes

  ---- k-connectors

e.g., *To turn the light on, either hire an electrician or (get a screwdriver and get some wire); to get wire goto some hardware shop; to get screwdriver goto Sam's house.*

Game Trees

- Problem spaces for typical games are represented as trees
- Root node represents the "board" configuration at which a decision must be made as to what is the best single move to make next (not necessarily the initial configuration)
- Evaluation function, f, rates a board configuration

  $f(\text{board}) \rightarrow$ a real number
- Arcs represent the possible legal moves for a player (no costs are associated with the arcs)
- Terminal nodes represent end-game configurations : the result must be "win"/ "lose"/ "draw"

Game Tree : AND-OR Tree

- If it is my turn to move, then the root is labeled as "MAX" node; otherwise it is labeled as "MIN" node
- Each level of the tree has nodes that are all MAX or all MIN;
- Nodes at level *i* are of the opposite kind from those at level *i+1*
- *Nodes corresponding to MIN's next move have successors that are like AND nodes*
- *Nodes corresponding to MAX's next move have successors that are like OR nodes*

Look Ahead !

- Complete game tree: includes all configurations that can be generated from the root by legal moves (leaves are terminal nodes)
- Incomplete game tree: includes all configurations that can be generated from the root by legal moves to a given depth (looking ahead/ ply upto a certain depth)

Evaluation Function

- Evaluates "goodness" of a game position – *estimates* board quality in leading to a win for one player
  - Contrast with heuristic search where evaluation function was a non-negative estimate of the cost from start node to a goal and passing through the given node.
- The zero-sum assumption allows us to use a single evaluation function to describe the goodness of a board with respect to both the players.
  - **f(n) > 0**: position n is good for me and bad for you
  - **f(n) < 0**: position n is bad for me and good for you
  - **f(n) near 0**: position n is neutral.
  - **f(n)** $+\infty$: win for me.
  - **f(n)** $-\infty$: win for you.

Evaluation Function : Examples

- Evaluation function is heuristic one ---- domain experts' knowledge plays a key role.

- Example of an Evaluation Function for Tic-Tac-Toe:

  f(n) = [# of 3-lengths open for me] - [# of 3-lengths open for you]

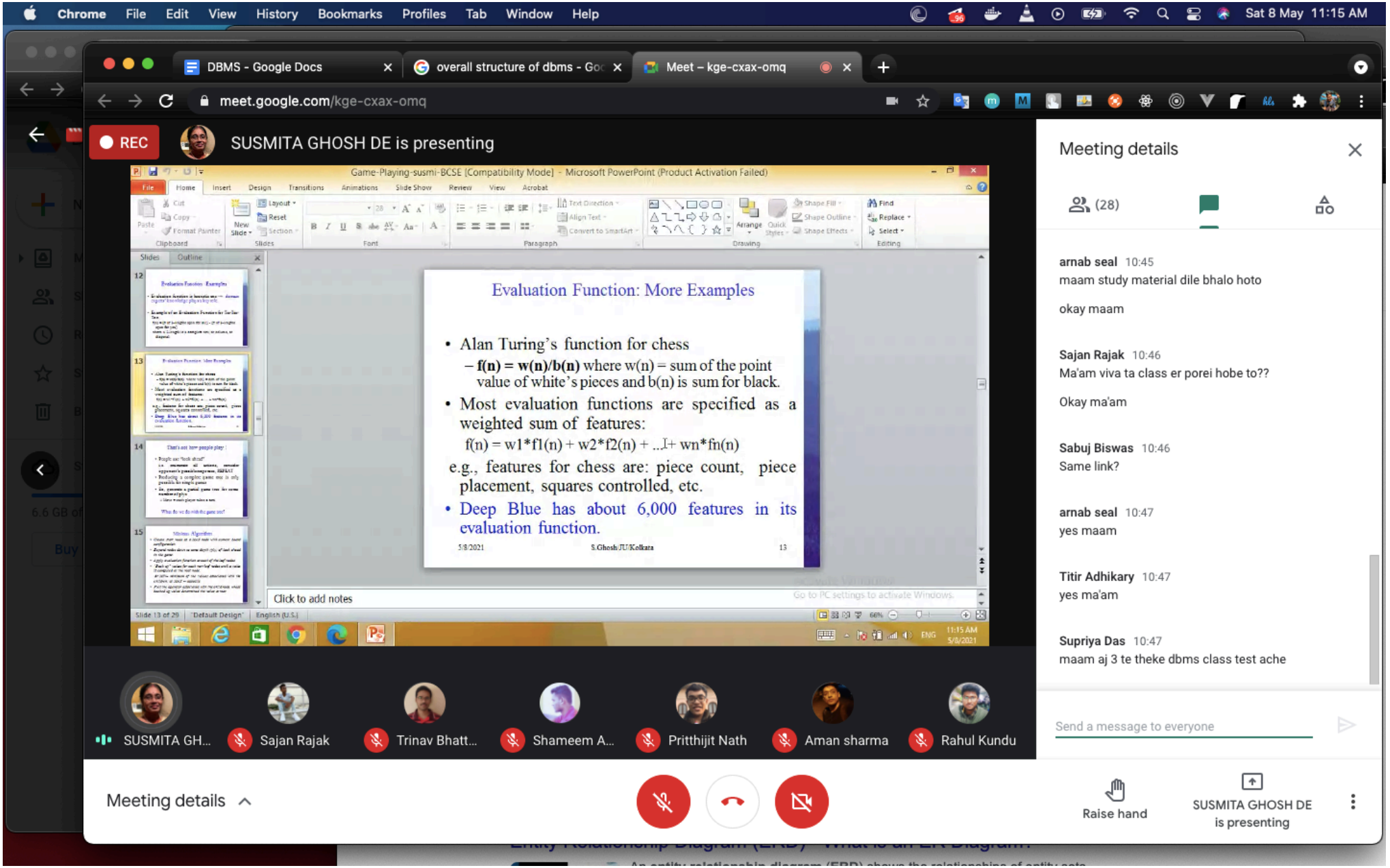
  where a 3-length is a complete row, or column, or diagonal.

Evaluation Function: More Examples

- Alan Turing's function for chess
  - **f(n) = w(n)/b(n)** where w(n) = sum of the point value of white's pieces and b(n) is sum for black.
- Most evaluation functions are specified as a weighted sum of features:

  $f(n) = w1*f1(n) + w2*f2(n) + ... + wn*fn(n)$

  e.g., features for chess are: piece count, piece placement, squares controlled, etc.
- Deep Blue has about 6,000 features in its evaluation function.


Chrome File Edit View History Bookmarks Profiles Tab Window Help
Sat 8 May 11:15 AM
DBMS - Google Docs
overall structure of dbms - Goo
Meet – kge-cxax-omq
meet.google.com/kge-cxax-omq
REC
SUSMITA GHOSH DE is presenting
Game-Playing-susmi-BCSE [Compatibility Mode] - Microsoft PowerPoint (Product Activation Failed)
Evaluation Function: More Examples
• Alan Turing's function for chess
– f(n) = w(n)/b(n) where w(n) = sum of the point value of white's pieces and b(n) is sum for black.
• Most evaluation functions are specified as a weighted sum of features:
f(n) = w1*f1(n) + w2*f2(n) + ...I+ wn*fn(n)
e.g., features for chess are: piece count, piece placement, squares controlled, etc.
• Deep Blue has about 6,000 features in its evaluation function.
5/8/2021 S.Ghosh/JU/Kolkata 13
Click to add notes
Meeting details
(28)
arnab seal 10:45
maam study material dile bhalo hoto
okay maam
Sajan Rajak 10:46
Ma'am viva ta class er porei hobe to??
Okay ma'am
Sabuj Biswas 10:46
Same link?
arnab seal 10:47
yes maam
Titir Adhikary 10:47
yes ma'am
Supriya Das 10:47
maam aj 3 te theke dbms class test ache
Send a message to everyone
SUSMITA GH...
Sajan Rajak
Trinav Bhatt...
Shameem A...
Pritthijit Nath
Aman sharma
Rahul Kundu
Meeting details
Raise hand
SUSMITA GHOSH DE is presenting

That's not how people play !

- People use "look ahead"

  i.e. enumerate all actions, consider opponent's possible responses, REPEAT
- Producing a complete **game tree** is only possible for simple games
- So, generate a partial game tree for some number of plys
  - Move = each player takes a turn

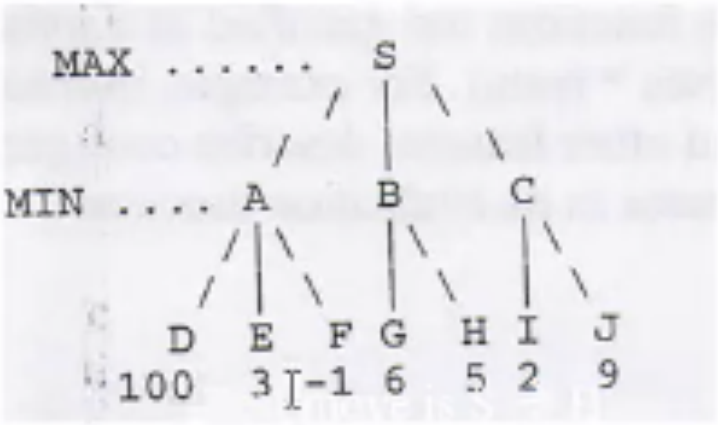
What do we do with the game tree?

Minimax Algorithm

- *Create start node as a MAX node with current board configuration*
- *Expand nodes down to some depth (ply) of look ahead in the game.*
- *Apply evaluation function at each of the leaf nodes*
- *"Back up" values for each non-leaf nodes until a value is computed at the root node.*

  At MIN– minimum of the values associated with the children; at MAX -- opposite
- *Pick the operator associated with the child node whose backed up value determined the value at root*

Example

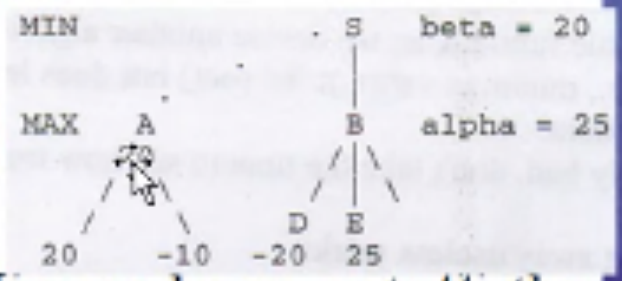


Alpha-Beta Pruning

- **Beta cutoff**: Given a Max node n, cutoff the search below n (i.e., do not generate or examine any more of n's children) if alpha(n) >= beta(i) for some Min node ancestor i of n

  beta cutoff at B

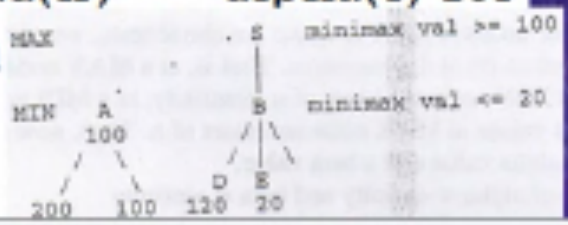
  alpha(B) (25) < beta(S) (20)



- **Alpha cutoff.:** Given a Min node n, cutoff the search below n (i.e., do not generate or examine any more of n's children) if beta(n) <= alpha(i) for some Max node ancestor i of n

  alpha cutoff at B

  beta (B) (20) < alpha(S) (100)



Why can't we use MiniMax/ Alpha-beta for Stochastic Games?

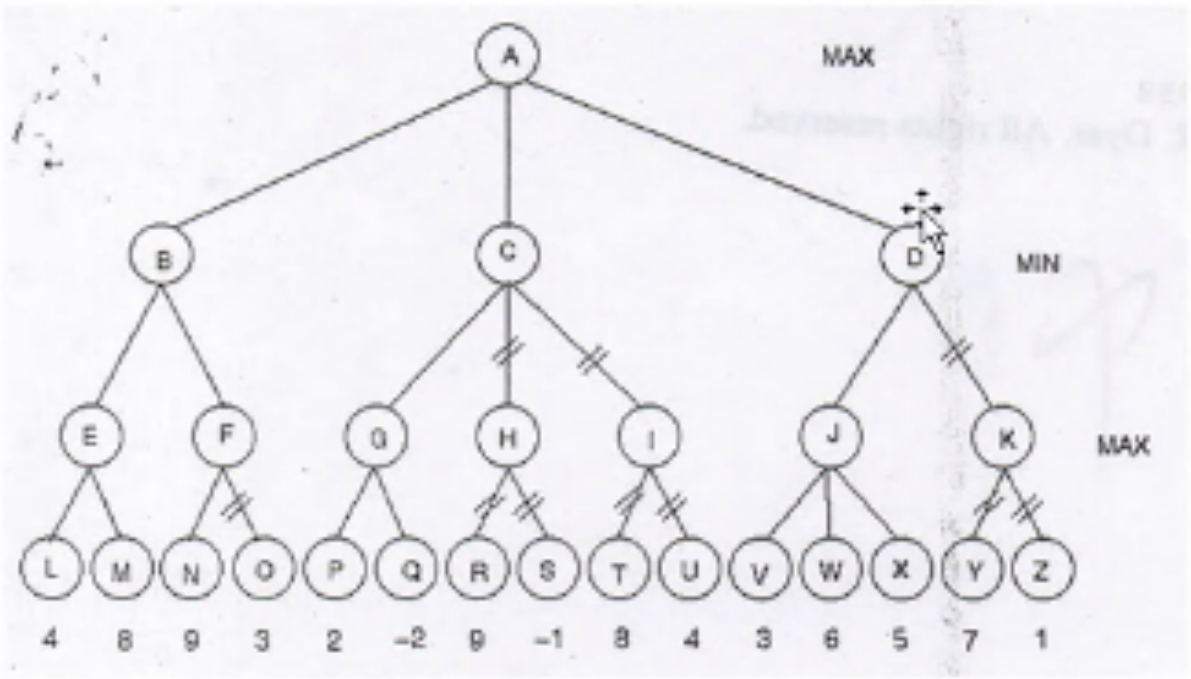
- Before a player chooses her move, she rolls dice and then knows exactly what they are
- And the immediate outcome of each move is also known
- But she does not know what moves will be available for her opponent to choose from (since chance involved)

We need to adapt MiniMax / alpha-beta to handle this

Effectiveness of Alpha-Beta Pruning

- Alpha-Beta Pruning algorithm is guaranteed to compute the same value for the root node as computed by Minimax.
- **Worst case:** NO pruning, examine O(b^d) leaf nodes.
- **Best case:** examine only O(2b^(d/2)) leaf nodes.
  - You can search twice as deep as Minimax!
- **Best case occurs** when each player's best move is the leftmost alternative, i.e. at MAX nodes the child with the largest value generated first, and at MIN nodes the opposite
- In Deep Blue, the average branching factor at each node was about 6 instead of about 35-40

Example



see examples