CS370

Symbolic Programming
Declarative Programming

LECTURE 19: Game Playing

Jong C. Park

park@cs.kaist.ac.kr

Computer Science Department

Korea Advanced Institute of Science and Technology

http://nlp.kaist.ac.kr/~cs370

Game Playing

- **Two-person, perfect-information games**
- **The minimax principle**
- **The alpha-beta algorithm**
  - an efficient implementation of minimax
- **Minimax-based programs**
  - refinements and limitations

Two-person, perfect-information games

The setting

- Two players, us and them, make moves alternatively.
- Both players have the complete information of the current situation in the game.
- The game is over when a position is reached that qualifies as ‘terminal’ by the rules of the game.
- The rules determine outcome of the game: win and loss. (cf. if draw → win, not-win)

Examples

- Chess, checkers, and go

Two-person, perfect-information games

- **Concepts in games and AND/OR trees**
  - game positions ←→ problems
  - terminal won position ←→ goal node, trivially solved problem
  - terminal lost position ←→ unsolvable problem
  - won position ←→ solved problem
  - us-to-move position ←→ OR node
  - them-to-move position ←→ AND node
- **Many concepts from searching AND/OR trees can be adapted for searching game trees.**

Two-person, perfect-information games

- **A simple Prolog program to find whether an us-to-move position is won or not**

```
won(Pos) :-
  terminalwon(Pos).

won(Pos) :-
  not terminallost(Pos),
  move(Pos,Pos1),
  not (move(Pos1,Pos2),not won(Pos2)).
```

Two-person, perfect-information games

- **The combinatorial complexity makes the naive search algorithm completely infeasible.**
  - The search space of astronomical proportions of the chess game includes some 10^{120} positions.
  - It can be argued that equal positions in the tree of Figure 22.1 occur at different places.
  - But it has also been shown that the number of different positions is far beyond anything manageable by foreseeable computers.

Two-person, perfect-information games

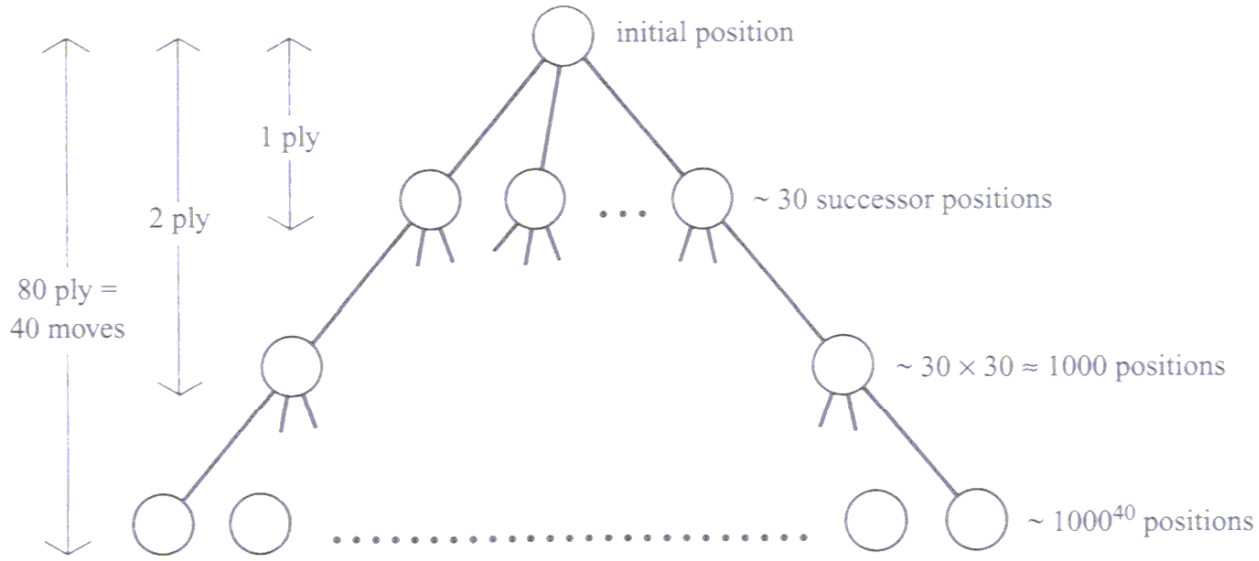


Figure 22.1 The complexity of game trees in chess. The estimates here are based on an approximation that there are about 30 legal moves from each chess position, and that terminal positions occur at a depth of 40 moves. One move is 2 plies (1 half-move by each side).

The minimax principle

⊙Game playing with the minimax principle

- A game tree is searched only up to a certain depth, typically a few moves, and then the tip nodes of the search tree are evaluated by some evaluation function.
  - The idea is to assess these terminal search position estimates without searching beyond them, thus saving time.
- The terminal position estimates propagate up the search tree according to the minimax principle.

The minimax principle

- The former yields position values for all the positions in the search tree.
- The move that leads from the initial, root position to its most promising successor is then actually played in the game.

The minimax principle

⊙Game tree vs. search tree

- A search tree is normally a part of the game tree (upper part) – that is, the part that is explicitly generated by the search process.
- Thus, terminal search positions do not have to be terminal positions of the game.

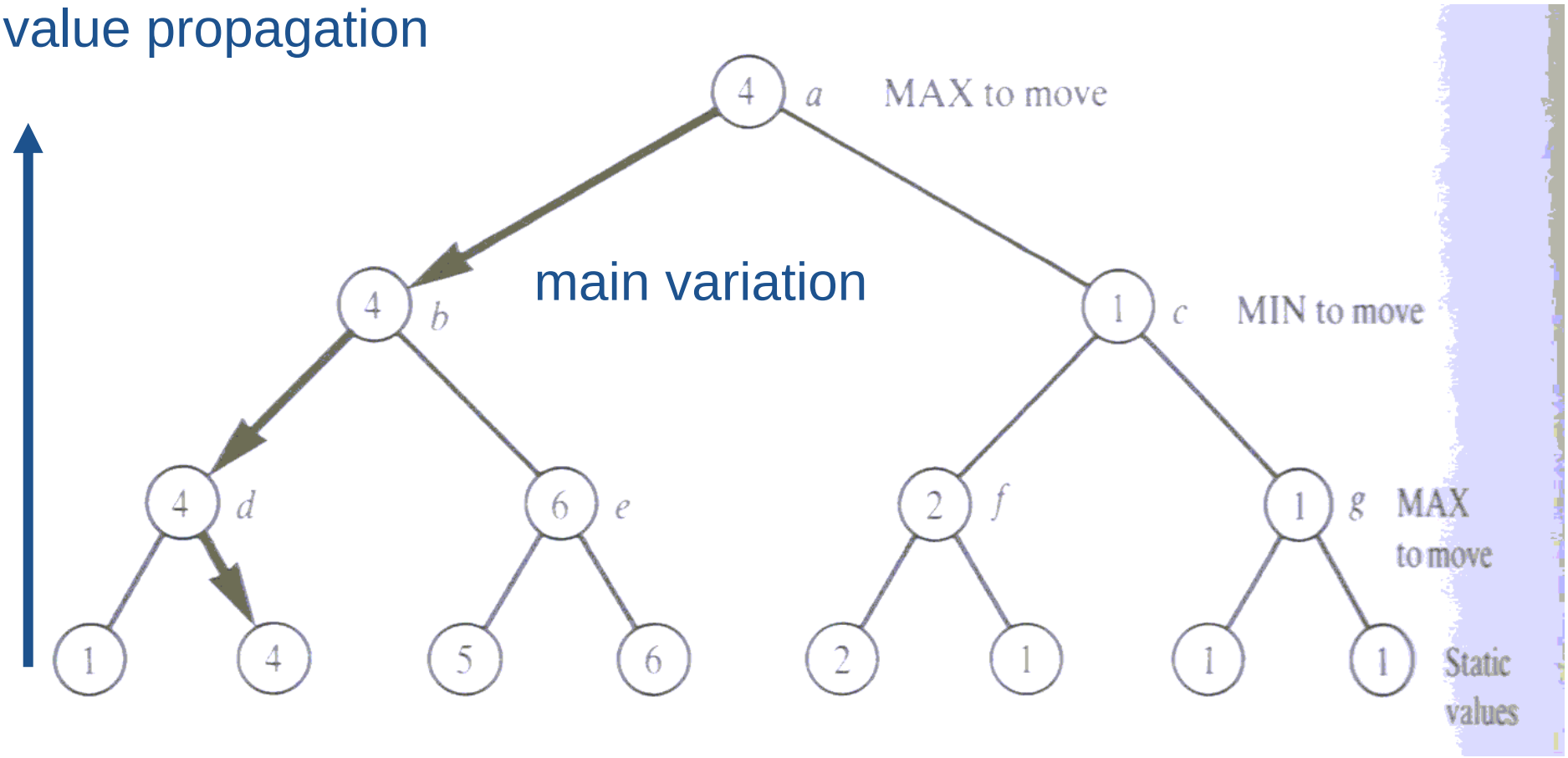
The minimax principle

The evaluation function

- It must be a heuristic estimator that estimates the winning chances from the point of view of one of the players.
- The higher the value the higher the player's chances are to win, and the lower the value the higher the opponent's chances are to win.
- Whenever MAX is to move, she will choose a move that maximizes the value; on the contrary, MIN will choose a move that minimizes the value.

The minimax principle

value propagation



The main variation defines the ‘minimax optimal’ play for both sides.

The minimax principle

⊙ The bottom-level values vs. the backed-up values

- The bottom-level values are static, since they are obtained by a ‘static’ evaluation function.
- The backed-up values are obtained dynamically by propagation of static values up the tree.

The minimax principle

The value propagation rules

- v(P): the static value of a position P
- V(P): the backed-up value of a position P
- P_1, ..., P_n: legal successor position of P
- The relation between static values and backed-up values:
  - V(P) = v(P) if P is a terminal position in a search tree (n = 0)
  - V(P) = $\max_i V(P_i)$ if P is a MAX-to-move position
  - V(P) = $\min_i V(P_i)$ if P is a MIN-to-move position

The minimax principle

- **minimax(Pos, BestSucc, Val)**

```
minimax(Pos, BestSucc, Val) :- moves(Pos, PosList), !,
   best(PosList, BestSucc, Val) ; staticval(Pos, Val).
best([Pos], Pos, Val) :- minimax(Pos, _, Val), !.
best([Pos1 | PosList], BestPos, BestVal) :-
   minimax(Pos1, _, Val1), best(PosList, Pos2, Val2),
   betterof(Pos1, Val1, Pos2, Val2, BestPos, BestVal).
betterof(Pos0, Val0, Pos1, Val1, Pos0, Val0) :-
   min_to_move(Pos0), Val0 > Val1, ! ;
   max_to_move(Pos0), Val0 < Val1, !.
betterof(Pos0, Val0, Pos1, Val1, Pos1, Val1).
```

The alpha-beta algorithm

Motivation

- Problems of the program in Figure 22.3
  - The program visits all the positions in the search tree, up to its terminal positions in a depth-first fashion, and statistically evaluates all the terminal positions of this tree.
  - Usually not all this work is necessary in order to correctly compute the minimax value of the root position.

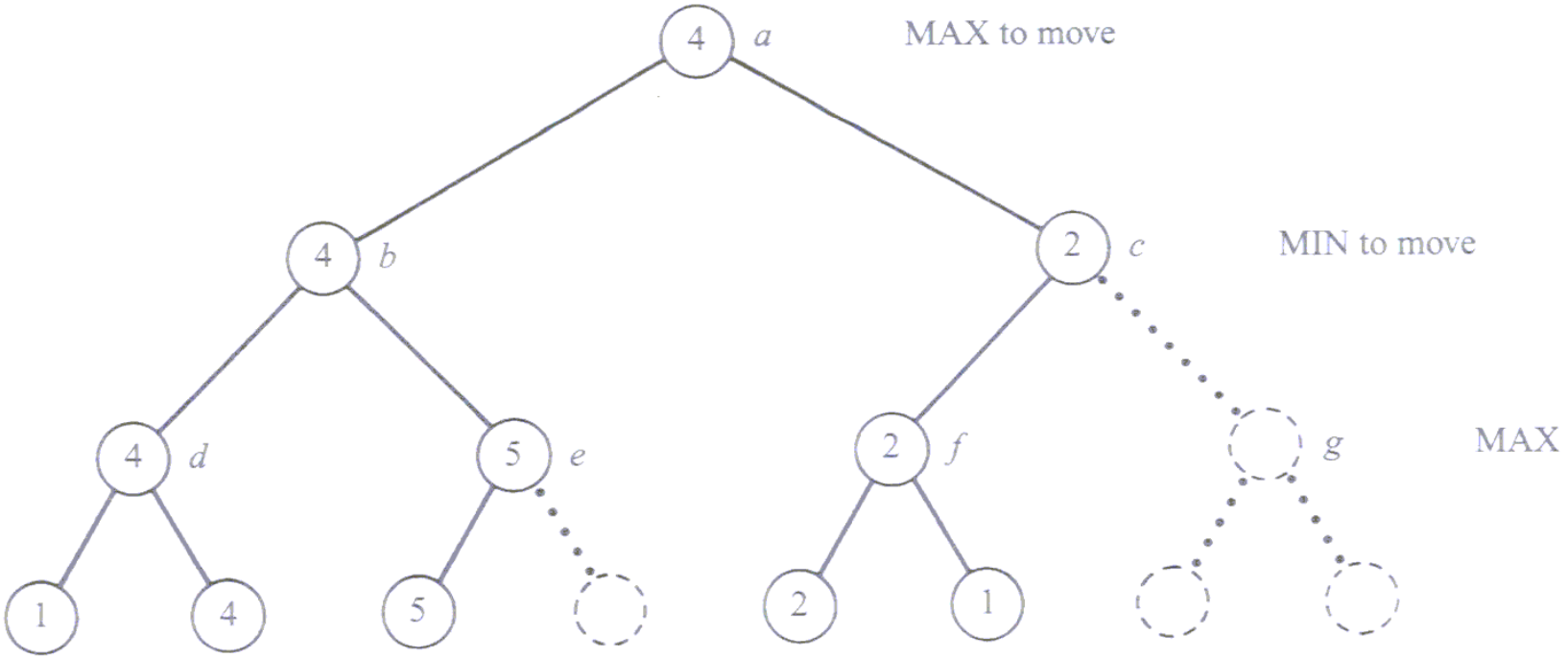
The alpha-beta algorithm

Idea

- Suppose that there are two alternative moves; once one of them has been shown to be clearly inferior to the other, it is not necessary to know *exactly* how much inferior it is for making the correct decision.

The alpha-beta algorithm

⊙Example



a → b → d → e → c → f

The alpha-beta algorithm

Alpha-beta pruning

- The key idea is to find a ‘good enough’ move, not necessarily the best, that is sufficiently good to make the correct decision.
- We use two bounds, *Alpha* and *Beta*, on the backed-up value of a position.
  - *Alpha* is the minimal value that MAX is already guaranteed to achieve.
  - *Beta* is the maximal value that MAX can hope to achieve.
  - From MIN’s point of view, *Beta* is the worst value for MIN that MIN is guaranteed to achieve.

The alpha-beta algorithm

Alpha-beta pruning

- If a position has been shown to have a value that lies outside the *Alpha-Beta* interval, then this is sufficient to know that this position is not in the main variation, without knowing the exact value of this position.
- 'Good enough' backed-up value V(P, *Alpha*, *Beta*) of a position P, with respect to *Alpha* and *Beta*, is defined to be as any value that satisfies:
  - V(P,Alpha,Beta) < Alpha if V(P) < Alpha
  - V(P,Alpha,Beta) = V(P) if Alpha <= V(P) <= Beta
  - V(P,Alpha,Beta) > Beta if V(P) > Beta
- If P is the root, V(P, -infinity, +infinity) = V(P).

The alpha-beta algorithm

⊙ The alpha-beta algorithm (Figure 22.5)

```
alphabeta(Pos, Alpha, Beta, GoodPos, Val) :-
    moves(Pos, PosList), !,
    boundedbest(Poslist, Alpha, Beta, GoodPos, Val)
    ;
    staticval(Pos, Val).
boundedbest([Pos|PosList], Alpha, Beta, GoodPos,
  GoodVal) :-
    alphabeta(Pos, Alpha, Beta, _, Val),
    goodenough(PosList, Alpha, Beta, Pos, Val, GoodPos,
    GoodVal).
```

The alpha-beta algorithm

⊙ The alpha-beta algorithm (Figure 22.5)

```
goodenough([ ], _, _, Pos, Val, Pos, Val) :- !.
goodenough(_, Alpha, Beta, Pos, Val, Pos, Val) :-
   min_to_move(Pos), Val > Beta, !;
   max_to_move(Pos), Val < Alpha, !.
```

Minimax-based programs

Background

- The minimax principle, together with the alpha-beta algorithm, is the basis of many successful game-playing programs, most notably chess programs.
- The general scheme:
  - Perform the alpha-beta search on the current position in the game, up to some fixed depth limit (dictated by the time constraints), using a game-specific evaluation function for evaluating the terminal positions of the search.
  - Then execute the best move on the play board, accept the opponent's reply, and start the same cycle again.

Minimax-based programs

⊙ Refinements [1]

- Distinguish turbulent positions from quiescent positions in setting the depth limit.
  - We should use the static evaluation only in quiescent positions.
  - The standard trick is to extend the search in turbulent positions beyond the depth limit until a quiescent position is reached.

Minimax-based programs

⊙ Refinements [2]

- Use heuristic pruning.
  - Achieve a greater depth limit by disregarding some less promising continuations.
  - Prune branches in addition to those that are pruned by the alpha-beta technique itself.
  - This entails the risk of overlooking some good continuation and incorrectly computing the minimax value.

Minimax-based programs

⊙ Refinements [3]

- Use progressive deepening.
  - The program repeatedly executes the alpha-beta search, first to some shallow depth, and then increases the depth limit on each iteration.
  - The process stops when the time limit has been reached.
  - The best move according to the deepest search is then played.
  - Advantages
    - Enables the time control.
    - The minimax values of the previous iteration can be used for preliminary ordering of positions on the next iteration, helping the alpha-beta algorithm to search strong moves first.

Minimax-based programs

⊙ Refinements

- Deal with the horizon effect.
  - Imagine a chess position in which the program's side inevitably loses a knight.
  - But the loss of the knight can be delayed at the cost of a lesser sacrifice, say a pawn.
  - This intermediate sacrifice may push the actual loss of the knight beyond the search limit (beyond the program's 'horizon').
  - Not seeing the eventual loss of the knight, the program will then prefer this variation to the quick death of the knight, eventually losing both the pawn and the knight.
  - Solution?

Minimax-based programs

⊙Limitations

- Differences of human (master) players vs. computer programs
  - The use of domain knowledge
    - evaluation function
    - tree-pruning heuristics
    - quiescence heuristics

Summary

- Two-person, perfect-information games
- The minimax principle
- The alpha-beta algorithm
- Minimax-based programs