Game Playing

Chapter 4

Outline

- Overview

- Minimax search

- Adding alpha-beta cutoffs

- Additional refinements

Overview

Old beliefs

Games provided a structured task in which it was very easy to measure success or failure.

Games did not obviously require large amounts of knowledge, thought to be solvable by straightforward search.

Overview

Chess

The average branching factor is around 35.

In an average game, each player might make 50 moves.

One would have to examine 35^{100} positions.

Overview

- Improve the generate procedure so that only good moves are generated.

- Improve the test procedure so that the best moves will be recognized and explored first.

Overview

- Improve the generate procedure so that only good moves are generated.

    plausible-moves vs. legal-moves

- Improve the test procedure so that the best moves will be recognized and explored first.

    less moves to be evaluated

Overview

- It is not usually possible to search until a goal state is found.

- It has to evaluate individual board positions by estimating how likely they are to lead to a win.

  Static evaluation function

- Credit assignment problem (Minsky, 1963).

Overview

- Good plausible-move generator.
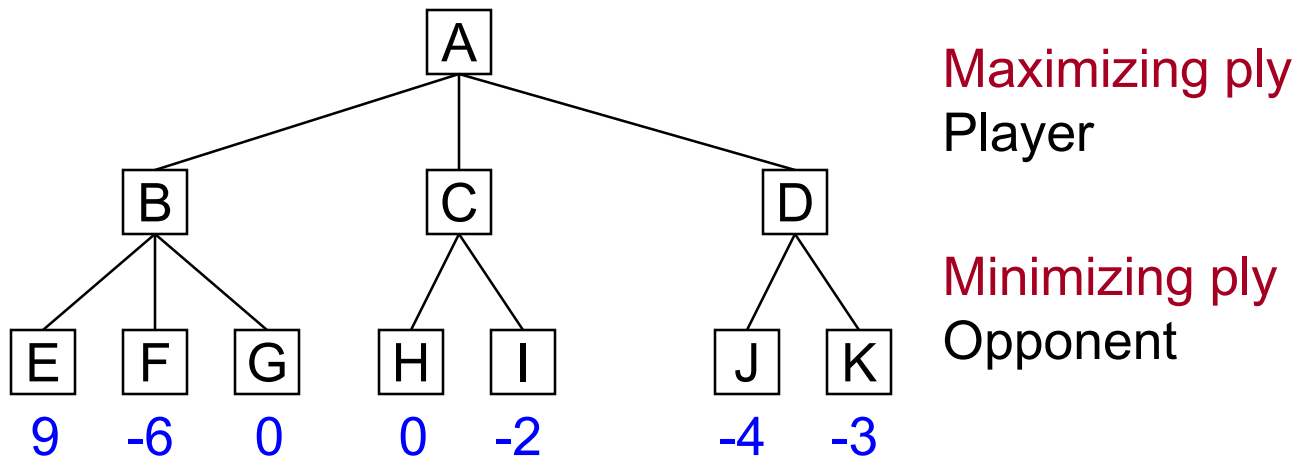
- Good static evaluation function.

Overview

- What search strategy should we use then?
  - One-person game
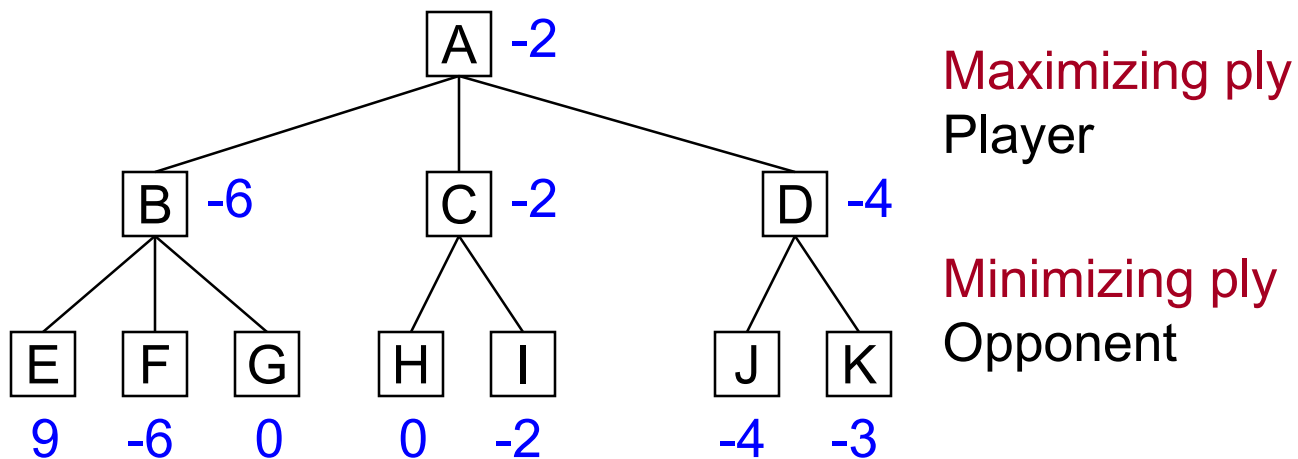 - Two-person game

Minimax Search

- Depth-first and depth-limited search.

- At the player choice, maximize the static evaluation of the next position.

- At the opponent choice, minimize the static evaluation of the next position.

Minimax Search

Maximizing ply
Player

Minimizing ply
Opponent

A

B   C   D

E   F   G   H   I   J   K

9   -6   0   0   -2   -4   -3

Two-ply search

11

Minimax Search

A -2

Maximizing ply
Player

B -6   C -2   D -4

Minimizing ply
Opponent

E F G   H I   J K

9 -6 0   0 -2   -4 -3

Two-ply search

Minimax Search

Player(Position, Depth):

for each S $\in$ SUCCESSORS(Position) do

RESULT = Opponent(S, Depth + 1)

NEW-VALUE = VALUE(RESULT)

if NEW-VALUE > MAX-SCORE, then

MAX-SCORE = NEW-VALUE

BEST-PATH = PATH(RESULT) + S

return

VALUE = MAX-SCORE

PATH = BEST-PATH

Minimax Search

Opponent(Position, Depth):

for each S $\in$ SUCCESSORS(Position) do

RESULT = Player(S, Depth + 1)

NEW-VALUE = VALUE(RESULT)

if NEW-VALUE < MIN-SCORE, then

MIN-SCORE = NEW-VALUE

BEST-PATH = PATH(RESULT) + S

return

VALUE = MIN-SCORE

PATH = BEST-PATH

Minimax Search

Any-Player(Position, Depth):

for each S $\in$ SUCCESSORS(Position) do

RESULT = Any-Player(S, Depth + 1)

NEW-VALUE = $-$ VALUE(RESULT)

if NEW-VALUE > BEST-SCORE, then
    BEST-SCORE = NEW-VALUE
    BEST-PATH = PATH(RESULT) + S

return

VALUE = BEST-SCORE

PATH = BEST-PATH

Minimax Search

MINIMAX(Position, Depth, Player):

- MOVE-GEN(Position, Player).

- STATIC(Position, Player).

- DEEP-ENOUGH(Position, Depth)

Minimax Search

1. if DEEP-ENOUGH(Position, Depth), then return:

   VALUE = STATIC(Position, Player)
   PATH = nil

2. SUCCESSORS = MOVE-GEN(Position, Player)

3. if SUCCESSORS is empty, then do as in Step 1

Minimax Search

4.   if SUCCESSORS is not empty:

RESULT-SUCC = MINIMAX(SUCC, Depth+1, Opp(Player))

NEW-VALUE = − VALUE(RESULT-SUCC)

if NEW-VALUE > BEST-SCORE, then:

BEST-SCORE = NEW-VALUE

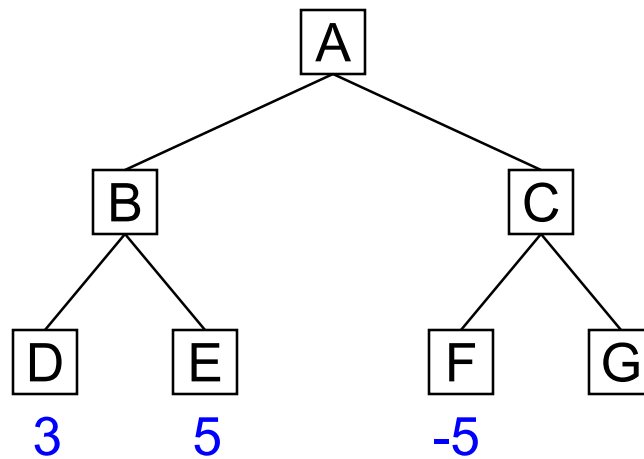BEST-PATH = PATH(RESULT-SUCC) + SUCC

5.   Return:

VALUE = BEST-SCORE

PATH = BEST-PATH

Adding Alpha-Beta Cutoffs

- Depth-first and depth-limited search.

    branch-and-bound

- At the player choice, maximize the static evaluation of the next position.

    $> \alpha$ threshold

- At the opponent choice, minimize the static evaluation of the next position.
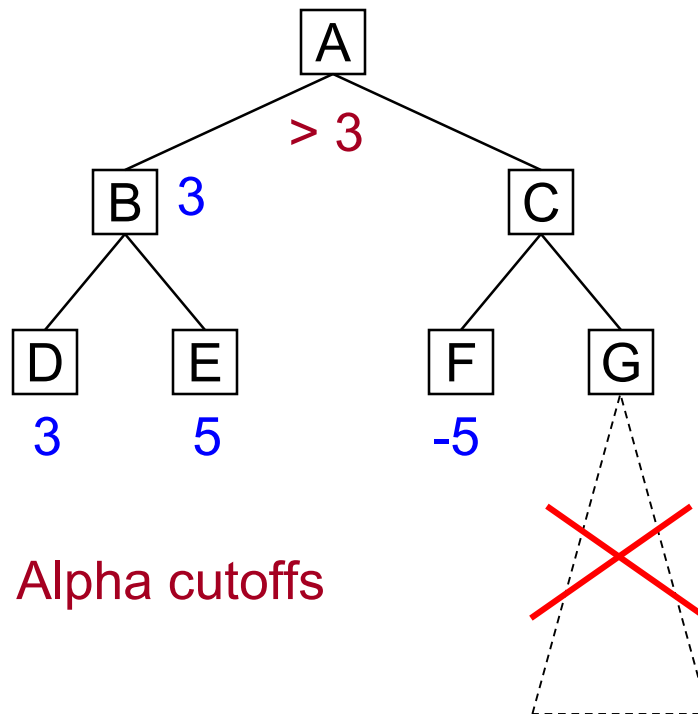
    $< \beta$ threshold

Adding Alpha-Beta Cutoffs

Maximizing ply
Player

Minimizing ply
Opponent

Adding Alpha-Beta Cutoffs

A

> 3

B 3

C

D   E

3   5

F   G

-5

Maximizing ply
Player

Minimizing ply
Opponent

Alpha cutoffs

21

Adding Alpha-Beta Cutoffs

Maximizing ply
Player

Minimizing ply
Opponent

Maximizing ply
Player

Minimizing ply
Opponent

Alpha and Beta
cutoffs

22

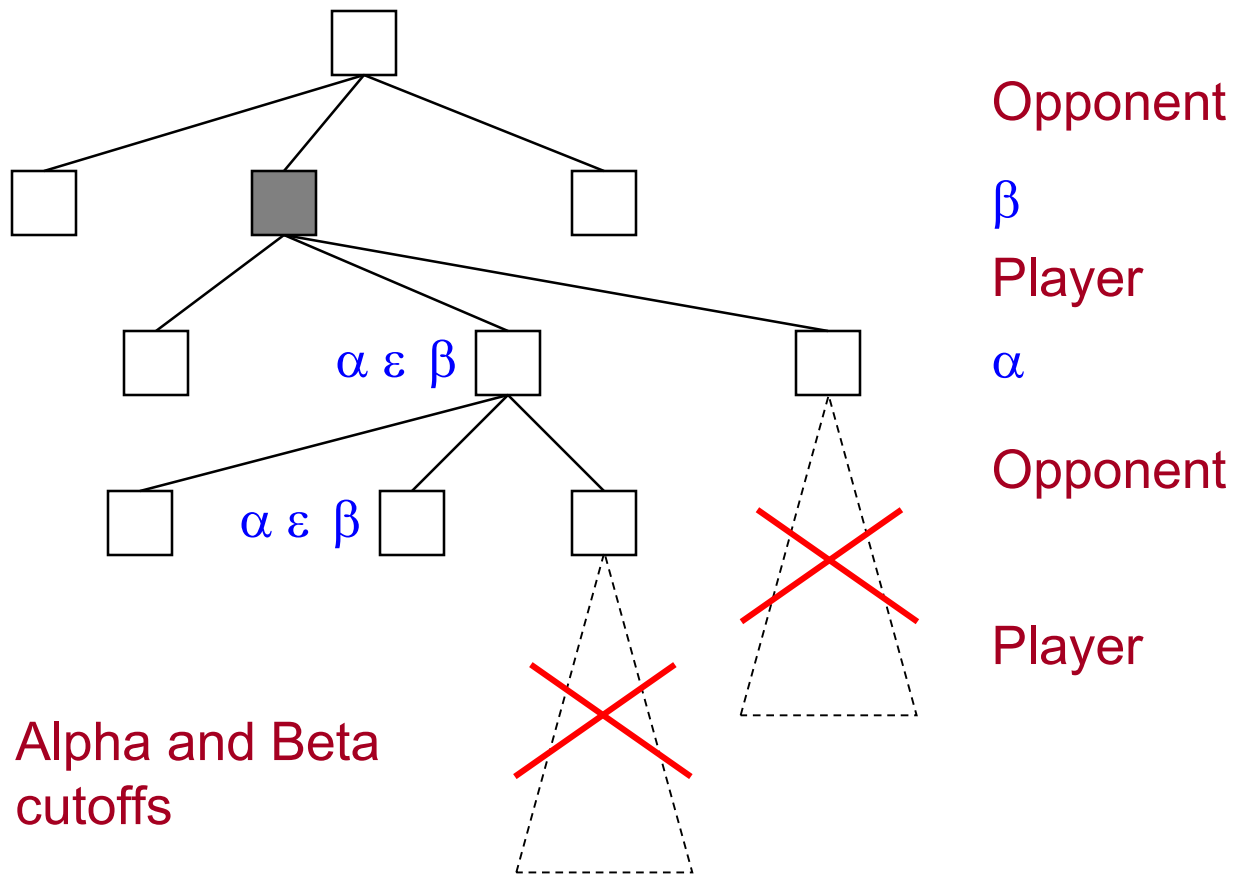# Adding Alpha-Beta Cutoffs

A

> 3

B   C

< 5

D   E   F   G   H

3   5

I   J   M   N

5   7

Alpha and Beta
cutoffs

K   L

0

Maximizing ply
Player

Minimizing ply
Opponent

Maximizing ply
Player

Minimizing ply
Opponent

23

Adding Alpha-Beta Cutoffs

Opponent

β

Player

α

Opponent

Player

$\alpha \ \varepsilon \ \beta$

$\alpha \ \varepsilon \ \beta$
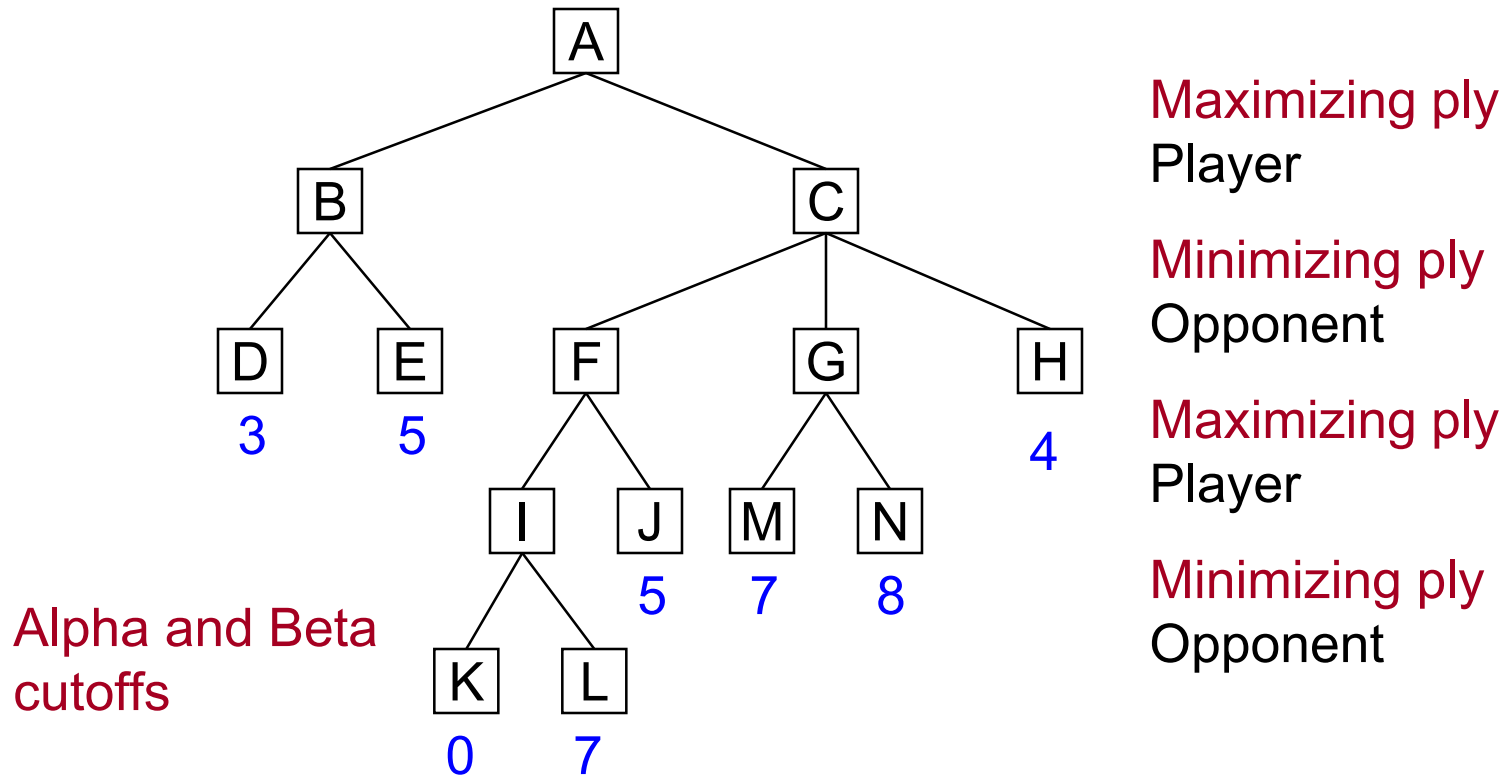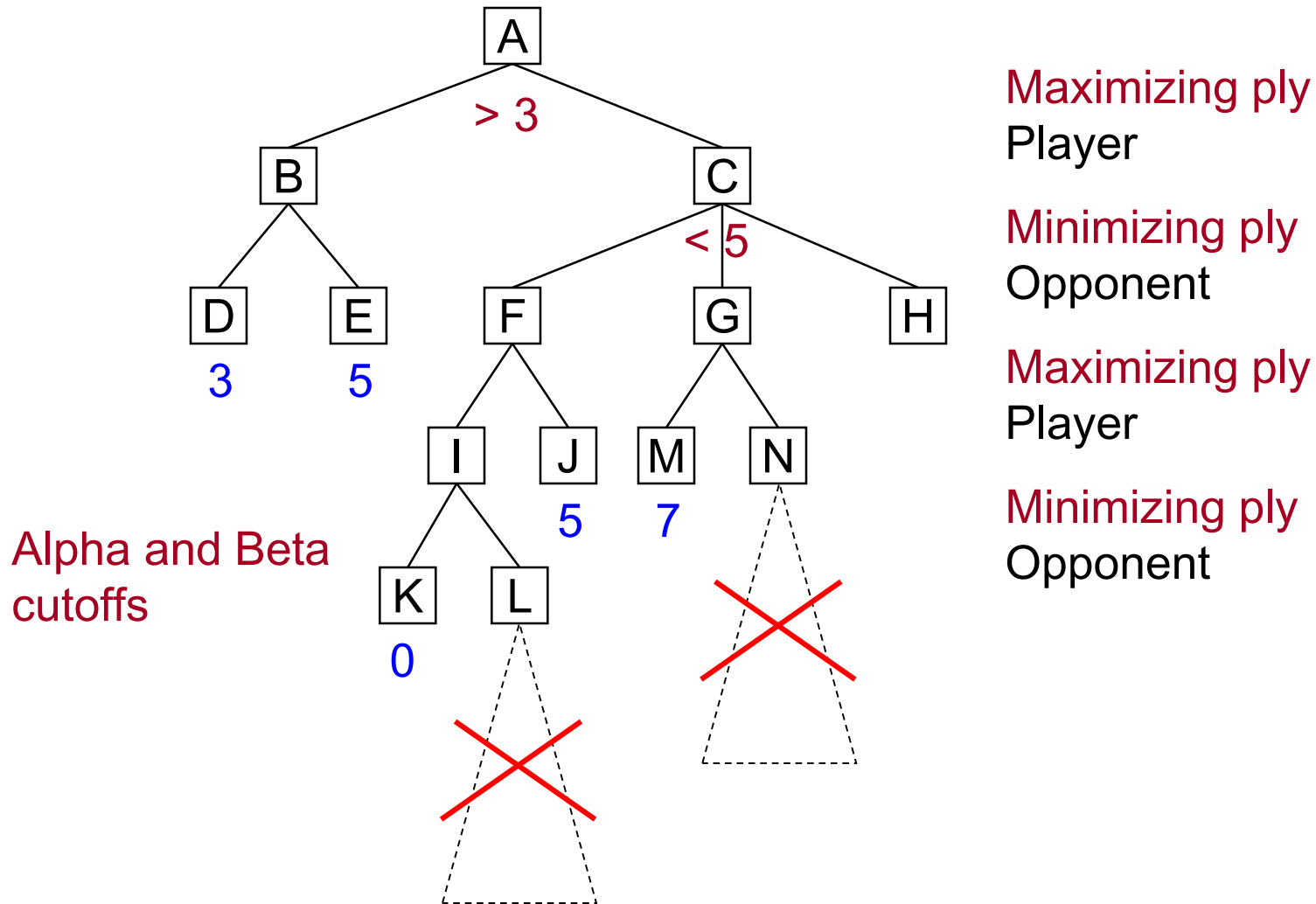
Alpha and Beta cutoffs

Player(Position, Depth, α, β):

for each S $\in$ SUCCESSORS(Position) do

RESULT = Opponent(S, Depth + 1, α, β)

NEW-VALUE = VALUE(RESULT)

if NEW-VALUE $>$ α, then

α = NEW-VALUE

BEST-PATH = PATH(RESULT) + S

if $\alpha \geq \beta$ then return

VALUE = α

PATH = BEST-PATH

return

VALUE = α

PATH = BEST-PATH

Opponent(Position, Depth, α, β):

for each S $\in$ SUCCESSORS(Position) do

　　　RESULT = Player(S, Depth + 1, α, β)

　　　NEW-VALUE = VALUE(RESULT)

　　　if NEW-VALUE $<$ β, then

　　　　　β = NEW-VALUE

　　　　　BEST-PATH = PATH(RESULT) + S

　　　if $\beta \leq \alpha$ then return

　　　　　VALUE = β

　　　　　PATH = BEST-PATH

return

　　　VALUE = β

　　　PATH = BEST-PATH

Any-Player(Position, Depth, α, β):

for each S $\in$ SUCCESSORS(Position) do

    RESULT = Any-Player(S, Depth + 1, $-\beta$, $-\alpha$)

    NEW-VALUE = $-$ VALUE(RESULT)

    if NEW-VALUE $> \alpha$, then

        α = NEW-VALUE

        BEST-PATH = PATH(RESULT) + S

    if $\alpha \geq \beta$ then return

        VALUE = α

        PATH = BEST-PATH

return

    VALUE = α

    PATH = BEST-PATH

Adding Alpha-Beta Cutoffs

MINIMAX-A-B(Position, Depth, Player, UseTd, PassTd):

• UseTd: checked for cutoffs.

• PassTd: current best value

Adding Alpha-Beta Cutoffs

1. if DEEP-ENOUGH(Position, Depth), then return:

   VALUE = STATIC(Position, Player)
   PATH = nil

2. SUCCESSORS = MOVE-GEN(Position, Player)

3. if SUCCESSORS is empty, then do as in Step 1

Adding Alpha-Beta Cutoffs

4. if SUCCESSORS is not empty:
    RESULT-SUCC = MINIMAX-A-B(SUCC, Depth + 1,
                          Opp(Player), – PassTd, – UseTd)

    NEW-VALUE = – VALUE(RESULT-SUCC)

    if NEW-VALUE > PassTd, then:
            PassTd = NEW-VALUE

            BEST-PATH = PATH(RESULT-SUCC) + SUCC

    if PassTd ≥ UseTd, then return:
            VALUE = PassTd

            PATH = BEST-PATH

5. Return:
    VALUE = PassTd

    PATH = BEST-PATH

Additional Refinements

- Futility cutoffs

- Waiting for quiescence

- Secondary search

- Using book moves

- Not assuming opponent's optimal move

Homework

Exercises   1-7, 9 (Chapter 12)