Algorithms for solving sequential (zero-sum) complete-information games

Tuomas Sandholm

CHESS,

MINIMAX SEARCH,

AND IMPROVEMENTS TO MINIMAX SEARCH

acm
ACM Chess Challenge
Garry Kasparov
vs
DEEP BLUE

1996

IBM
RS/6000 SP

1996

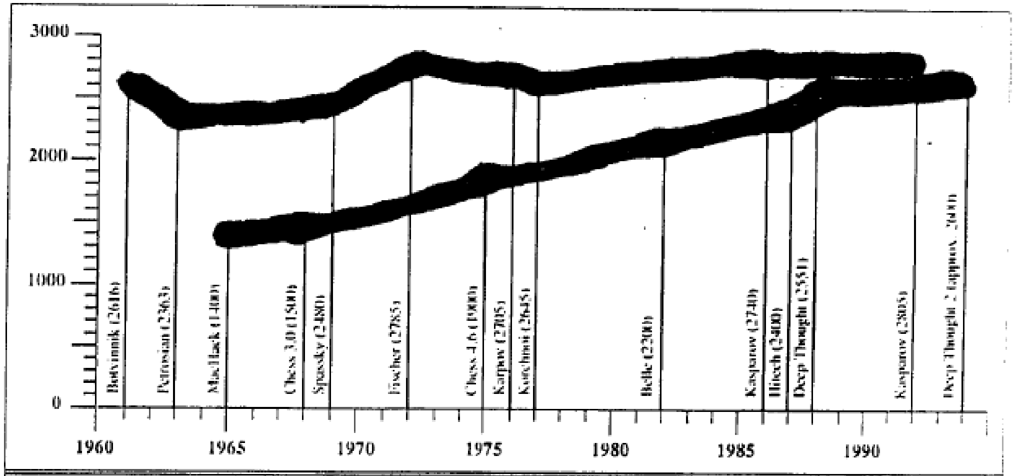


Figure 5.12 Ratings of human and machine chess champions.

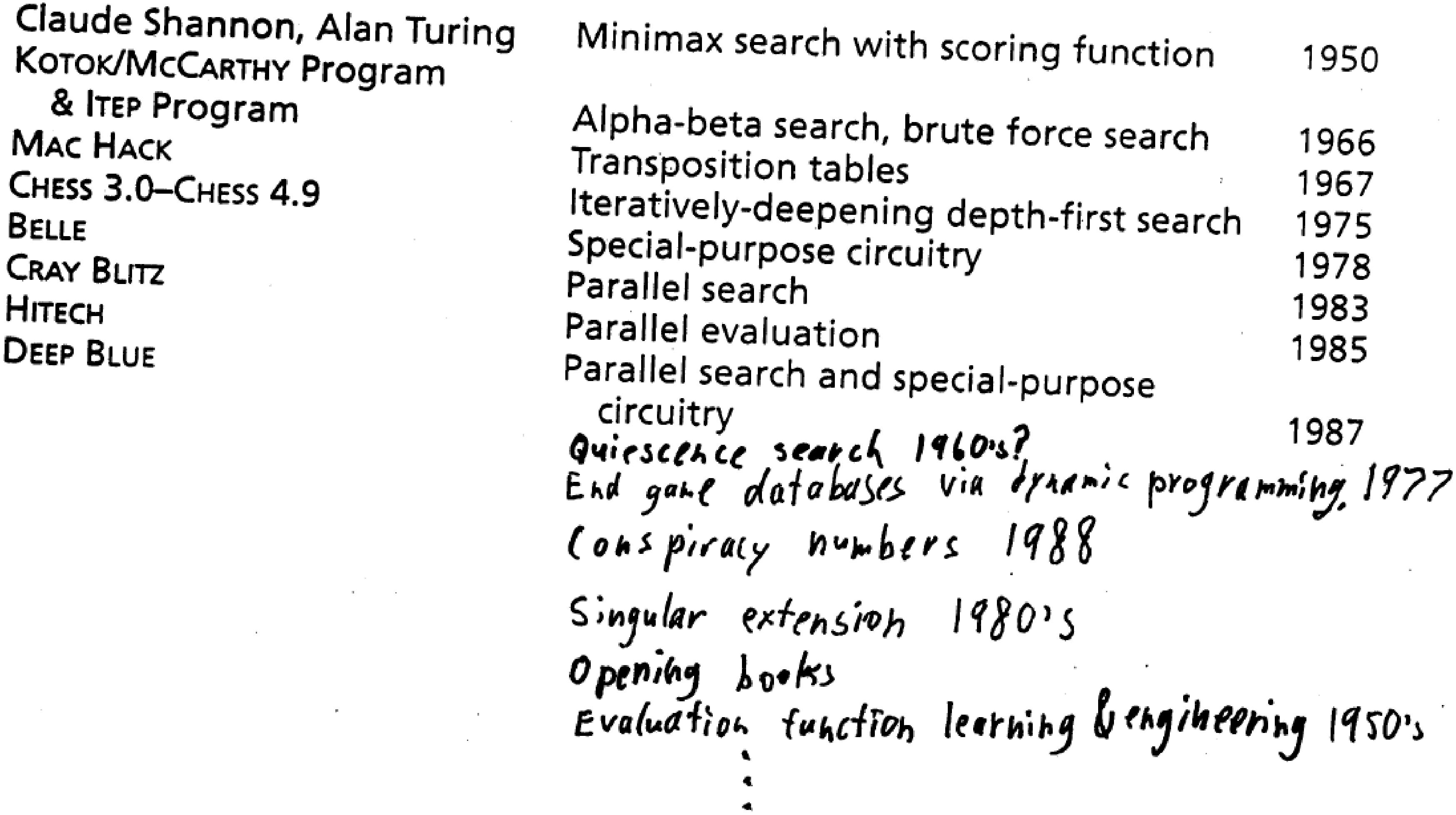
Deep Blue team. Front, left to right: Joel Benjamin, Chung-Jen Tan. Back, left to right: Jerry Brody, Murray Campbell, Feng-Hsiung Hsu, and Joe Hoane.

1997

3½ - 2½

Loss-win-draw-draw-draw-win

Rich history of cumulative ideas



Claude Shannon, Alan Turing	Minimax search with scoring function	1950
Kotok/McCarthy Program & Itep Program	Alpha-beta search, brute force search	1966
Mac Hack	Transposition tables	1967
Chess 3.0–Chess 4.9	Iteratively-deepening depth-first search	1975
Belle	Special-purpose circuitry	1978
Cray Blitz	Parallel search	1983
Hitech	Parallel evaluation	1985
Deep Blue	Parallel search and special-purpose circuitry	1987

Quiescence search 1960's?

End game databases via dynamic programming 1977

Conspiracy numbers 1988

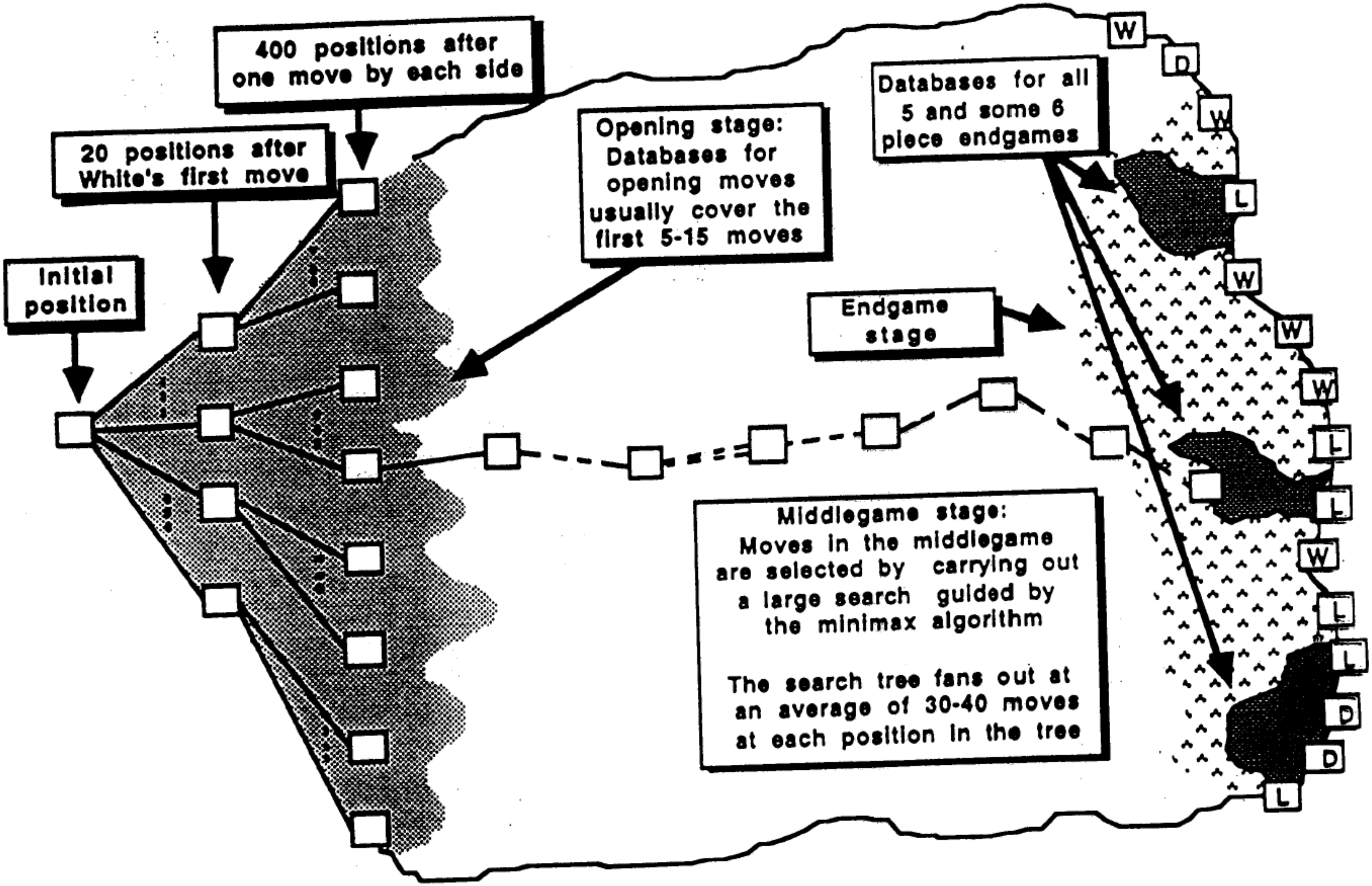
Singular extension 1980's

Opening books

Evaluation function learning & engineering 1950's

⋮

Chess game tree



400 positions after one move by each side

20 positions after White's first move

Initial position

Opening stage: Databases for opening moves usually cover the first 5-15 moves

Databases for all 5 and some 6 piece endgames

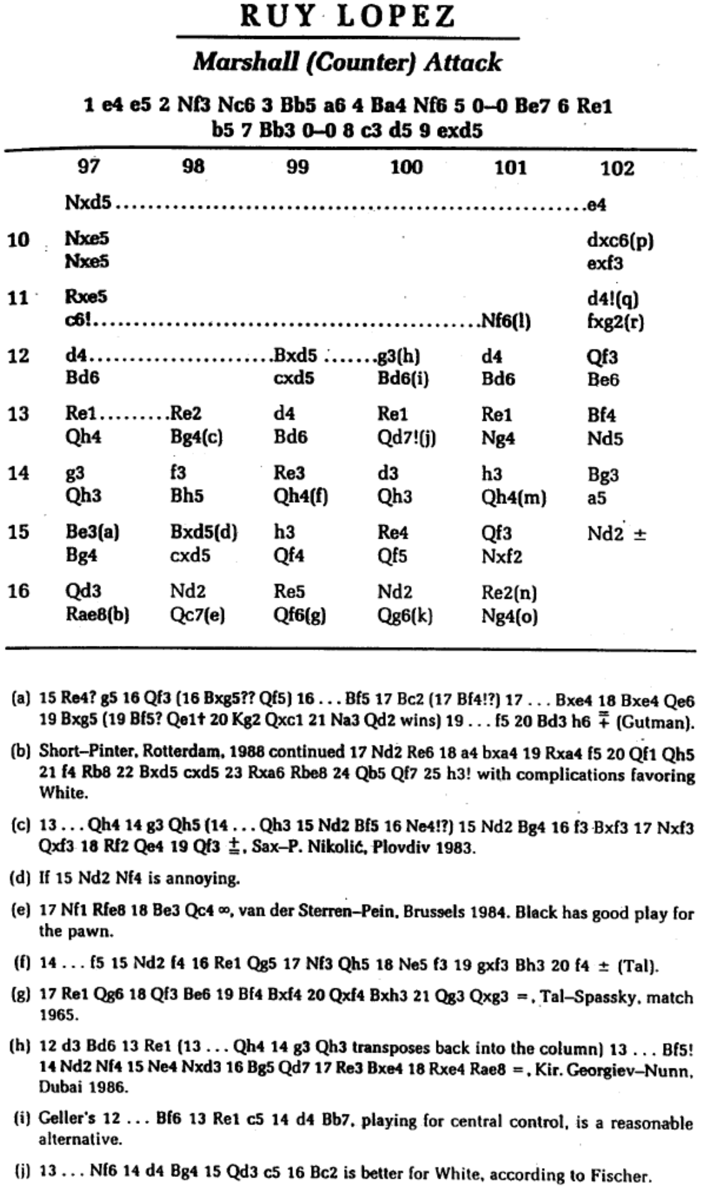
Endgame stage

Middlegame stage: Moves in the middlegame are selected by carrying out a large search guided by the minimax algorithm

The search tree fans out at an average of 30-40 moves at each position in the tree

Opening books (available electronically too)

Example opening where the book goes 16 moves (32 plies) deep



RUY LOPEZ

Marshall (Counter) Attack

1 e4 e5 2 Nf3 Nc6 3 Bb5 a6 4 Ba4 Nf6 5 0–0 Be7 6 Re1 b5 7 Bb3 0–0 8 c3 d5 9 exd5

	97	98	99	100	101	102
	Nxd5					.... e4
10	Nxe5 Nxe5					dxc6(p) exf3
11	Rxe5 c6!				 .... Nf6(l)	d4!(q) fxg2(r)

12	d4 ....
	Bd6
			Bxd5 ....
			cxd5
				g3(h)
				Bd6(i)
					d4
					Bd6
						Qf3
						Be6

13	Re1 .... Qh4	Re2 Bg4(c)	d4 Bd6	Re1 Qd7!(j)	Re1 Ng4	Bf4 Nd5
14	g3 Qh3	f3 Bh5	Re3 Qh4(f)	d3 Qh3	h3 Qh4(m)	Bg3 a5
15	Be3(a) Bg4	Bxd5(d) cxd5	h3 Qf4	Re4 Qf5	Qf3 Nxf2	Nd2 ±

16	Qd3
	Rae8(b)
		Nd2
		Qc7(e)
			Re5
			Qf6(g)
				Nd2
				Qg6(k)
					Re2(n)
					Ng4(o)	

(a) 15 Re4? g5 16 Qf3 (16 Bxg5?? Qf5) 16 . . . Bf5 17 Bc2 (17 Bf4!?) 17 . . . Bxe4 18 Bxe4 Qe6 19 Bxg5 (19 Bf5? Qe1† 20 Kg2 Qxc1 21 Na3 Qd2 wins) 19 . . . f5 20 Bd3 h6 ∓ (Gutman).

(b) Short–Pinter, Rotterdam, 1988 continued 17 Nd2 Re6 18 a4 bxa4 19 Rxa4 f5 20 Qf1 Qh5 21 f4 Rb8 22 Bxd5 cxd5 23 Rxa6 Rbe8 24 Qb5 Qf7 25 h3! with complications favoring White.

(c) 13 . . . Qh4 14 g3 Qh5 (14 . . . Qh3 15 Nd2 Bf5 16 Ne4!?) 15 Nd2 Bg4 16 f3 Bxf3 17 Nxf3 Qxf3 18 Rf2 Qe4 19 Qf3 ⩲, Sax–P. Nikolić, Plovdiv 1983.

(d) If 15 Nd2 Nf4 is annoying.

(e) 17 Nf1 Rfe8 18 Be3 Qc4 ∞, van der Sterren–Pein, Brussels 1984. Black has good play for the pawn.

(f) 14 . . . f5 15 Nd2 f4 16 Re1 Qg5 17 Nf3 Qh5 18 Ne5 f3 19 gxf3 Bh3 20 f4 ± (Tal).

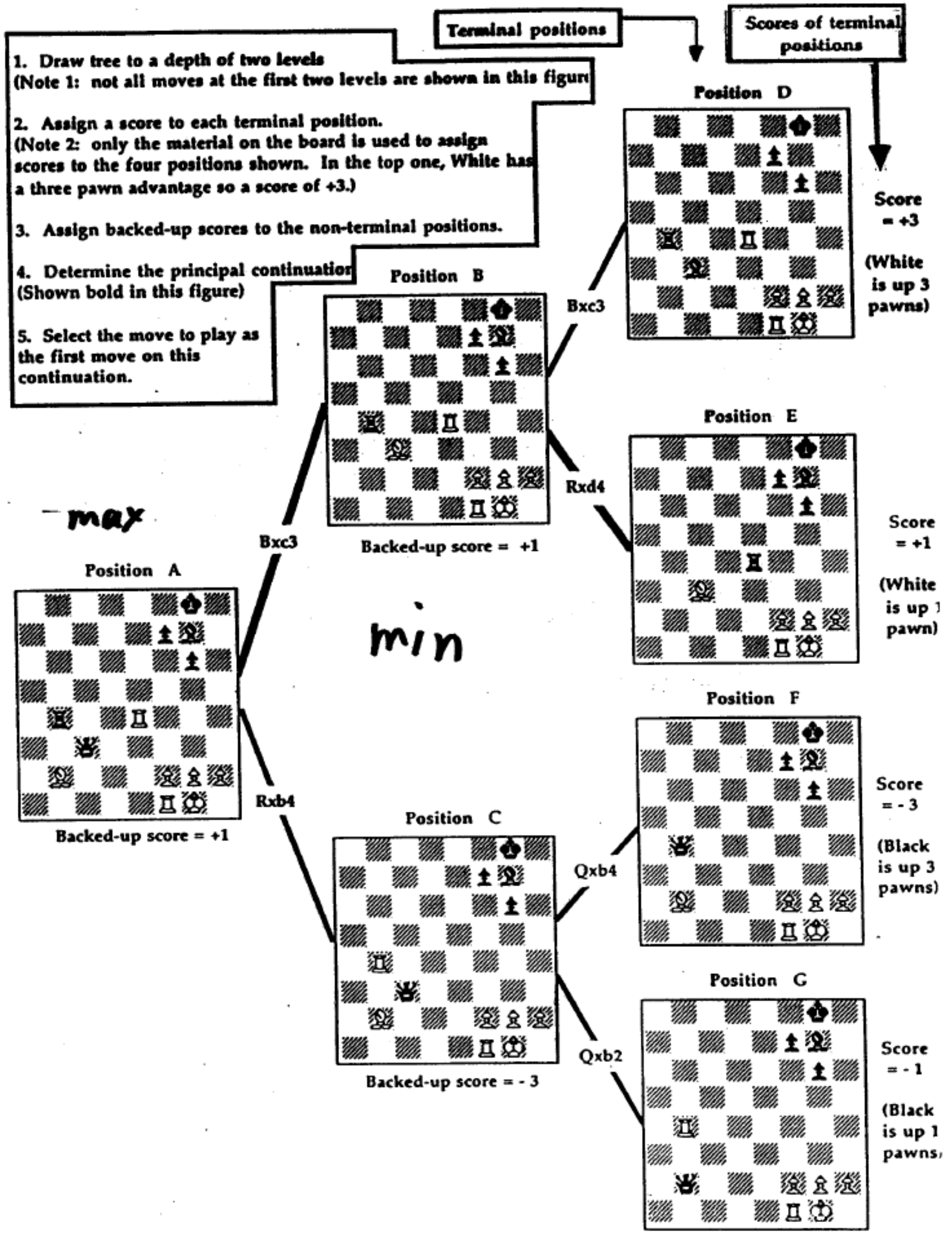
(g) 17 Re1 Qg6 18 Qf3 Be6 19 Bf4 Bxf4 20 Qxf4 Bxh3 21 Qg3 Qxg3 =, Tal–Spassky, match 1965.

(h) 12 d3 Bd6 13 Re1 (13 . . . Qh4 14 g3 Qh3 transposes back into the column) 13 . . . Bf5! 14 Nd2 Nf4 15 Ne4 Nxd3 16 Bg5 Qd7 17 Re3 Bxe4 18 Rxe4 Rae8 =, Kir. Georgiev–Nunn, Dubai 1986.

(i) Geller's 12 . . . Bf6 13 Re1 c5 14 d4 Bb7, playing for central control, is a reasonable alternative.

(j) 13 . . . Nf6 14 d4 Bg4 15 Qd3 c5 16 Bc2 is better for White, according to Fischer.

Minimax algorithm (not all branches are shown)



1. Draw tree to a depth of two levels
(Note 1: not all moves at the first two levels are shown in this figur

2. Assign a score to each terminal position.
(Note 2: only the material on the board is used to assign scores to the four positions shown. In the top one, White has a three pawn advantage so a score of +3.)

3. Assign backed-up scores to the non-terminal positions.

4. Determine the principal continuation
(Shown bold in this figure)

5. Select the move to play as the first move on this continuation.

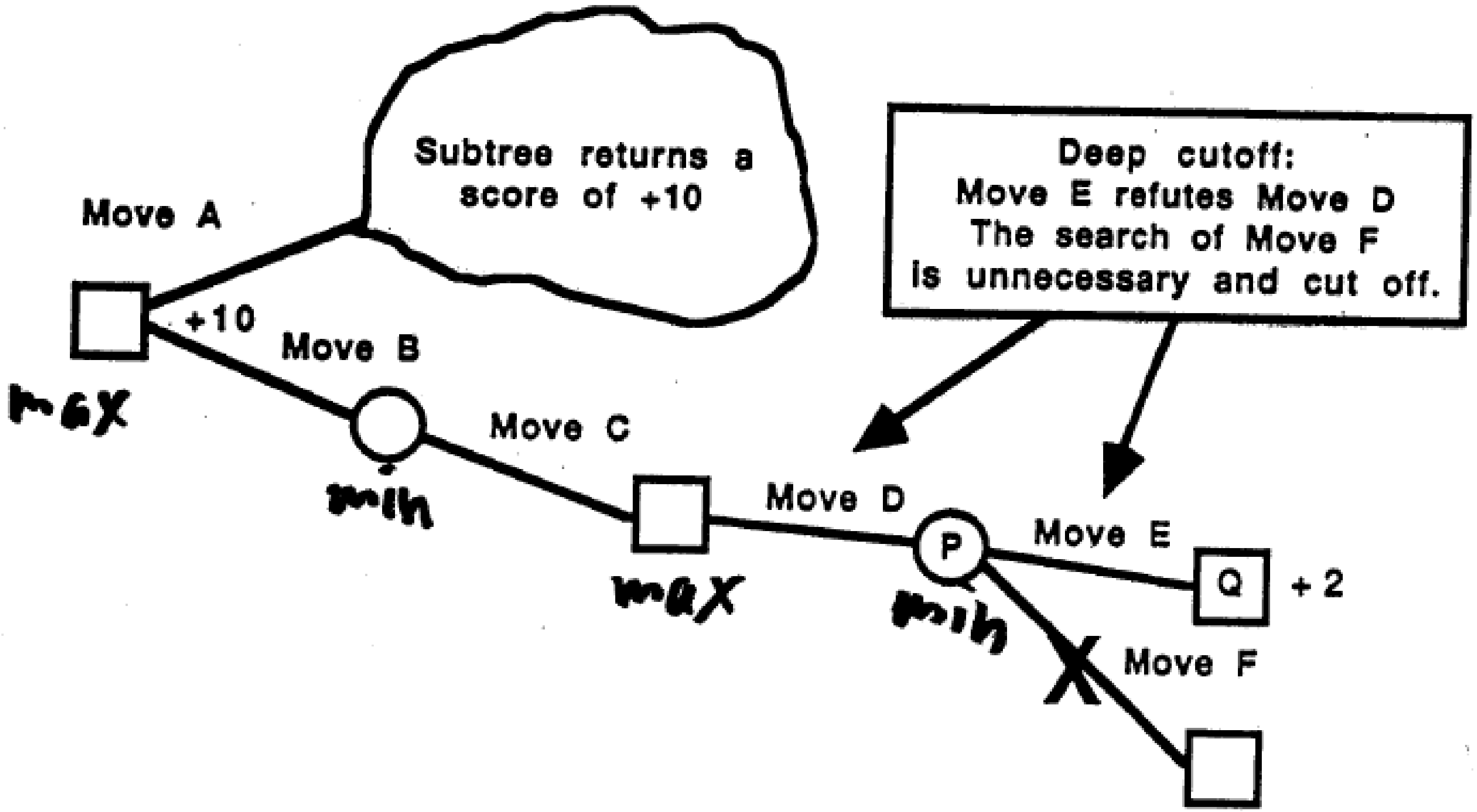
```
recursive function      MINIMAX(POSITION,DEPTH);
    {MINIMAX is the name of the process, which requires two inputs: a chess
    POSITION with white to move, and a number DEPTH indicating the ply
    level at which evaluation is to take place. The result of this process is the
    minimax value of the position}
if      DEPTH = 0
then        MINIMAX := EVAL(POSITION)
                {the function EVAL evaluates at the bottom level}
else
  begin
    MINIMAX := FINDMOVES(POSITION,MOVES,NMOVES)
        {the move generator finds all legal moves from POSITION; the
        value produced and stored in MINIMAX is that of a loss, say –100,
        or zero if stalemate (NMOVES = 0 and no check)}
  if      NMOVES > 0        {loop over legal moves}
  then for¯     i := 1 to NMOVES do
    NEWPOSITION := SWAPSIDES(MAKEMOVE(POSITION,MOVE(i)));
        {produces a new position, by making move i in POSITION, and then
        reversing Black and White sides}
    VALUE := -MINIMAX(NEWPOSITION,DEPTH–1);
        {here comes the magic: assuming that the MINIMAX function is
        available for use (not quite true at the time this line is written), it is
        called upon to produce a minimax value for NEWPOSITION (with
        depth decreased by 1); since this value is with respect to the Black
        side, its sign is reversed}
    if VALUE > MINIMAX then MINIMAX := VALUE
        {MINIMAX contains the largest value found up to now; in this
        example, no record is kept of the associated move}
  end do
  end
```

Folk wisdom for playing against computers:
Play open positions ⇒ increases the branching factor
⇒ reduces computer's lookahead.

Search depth pathology

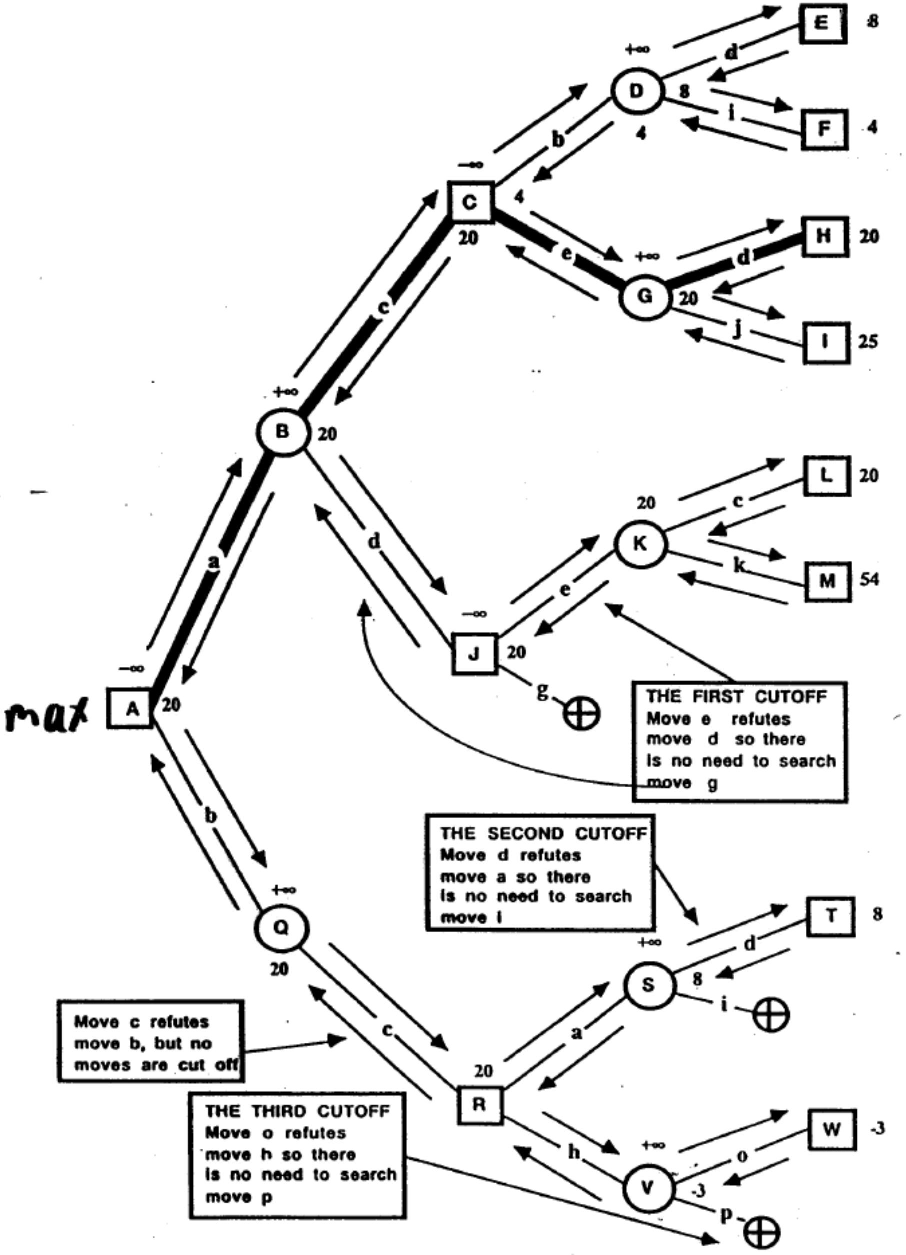
- Beal (1980) and Nau (1982, 83) analyzed whether values backed up by minimax search are more trustworthy than the heuristic values themselves. The analyses of the model showed that backed-up values are somewhat less trustworthy
- Anomaly goes away if sibling nodes' values are highly correlated [Beal 1982, Bratko & Gams 1982, Nau 1982]
- Pearl (1984) partly disagreed with this conclusion, and claimed that while strong dependencies between sibling nodes can eliminate the pathology, practical games like chess don't possess dependencies of sufficient strength.
  - He pointed out that few chess positions are so strong that they cannot be spoiled abruptly if one really tries hard to do so.
  - He concluded that success of minimax is "based on the fact that common games do not possess a uniform structure but are riddled with early terminal positions, colloquially named blunders, pitfalls or traps. Close ancestors of such traps carry more reliable evaluations than the rest of the nodes, and when more of these ancestors are exposed by the search, the decisions become more valid."
- Still not fully understood. For new results, see:
  - Sadikov, Bratko, Kononenko. (2003) Search versus Knowledge: An Empirical Study of Minimax on KRK, In: van den Herik, Iida and Heinz (eds.) Advances in Computer Games: Many Games, Many Challenges, Kluwer Academic Publishers, pp. 33-44
  - Understanding Sampling Style Adversarial Search Methods [PDF]. Raghuram Ramanujan, Ashish Sabharwal, Bart Selman. UAI-2010, pp 474-483.
  - On Adversarial Search Spaces and Sampling-Based Planning [PDF]. Raghuram Ramanujan, Ashish Sabharwal, Bart Selman. ICAPS-2010, pp 242-245.
- Also present in imperfect-information games when one party has limited lookahead [Kroer & Sandholm IJCAI-15; Kroer, Farina & Sandholm AAAI-18]

α-β -pruning



Partially drawn game tree showing deep alpha-beta cutoff

α-β -search on ongoing example



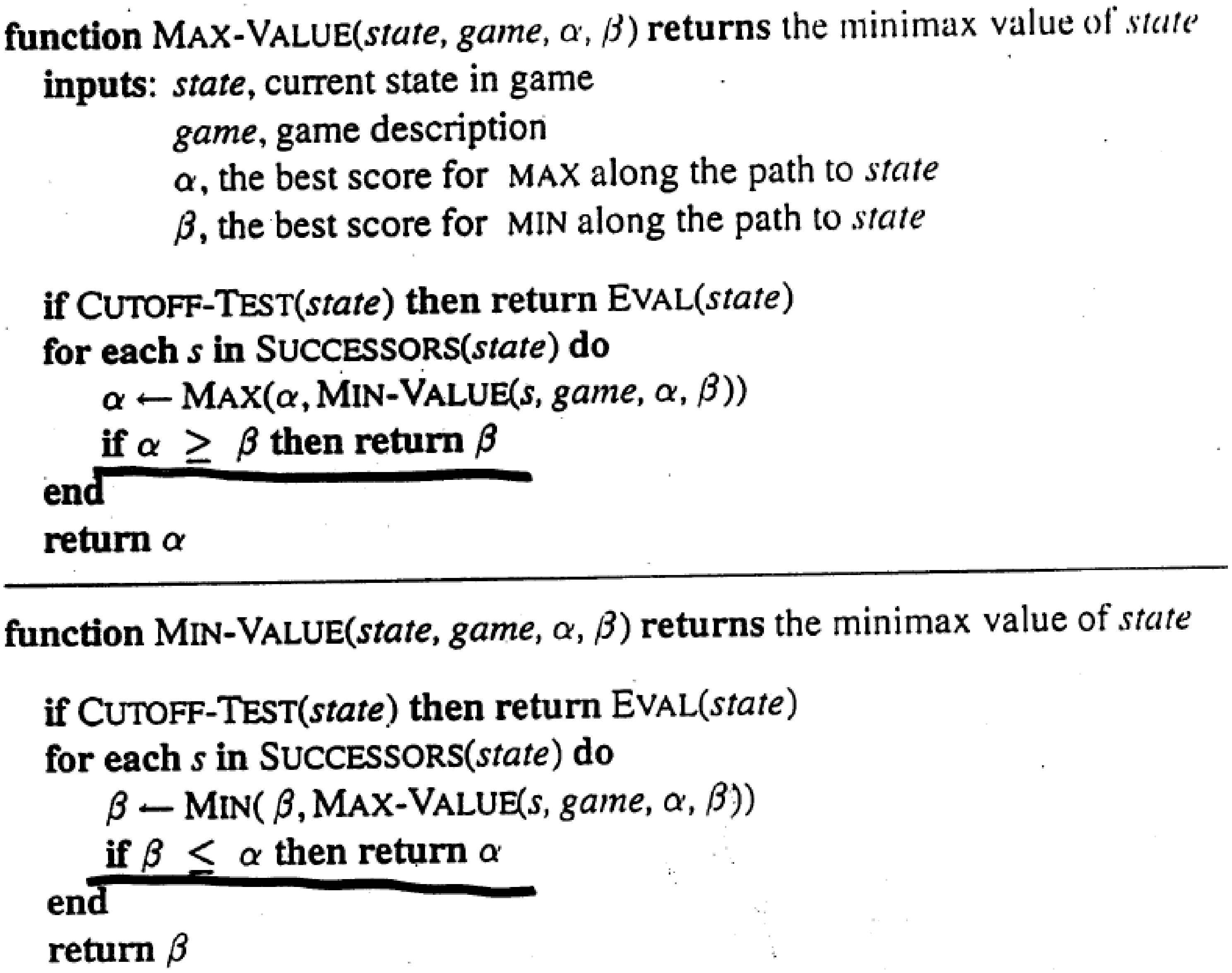
THE FIRST CUTOFF
Move e refutes
move d so there
is no need to search
move g

THE SECOND CUTOFF
Move d refutes
move a so there
is no need to search
move i

Move c refutes
move b, but no
moves are cut off

THE THIRD CUTOFF
Move o refutes
move h so there
is no need to search
move p

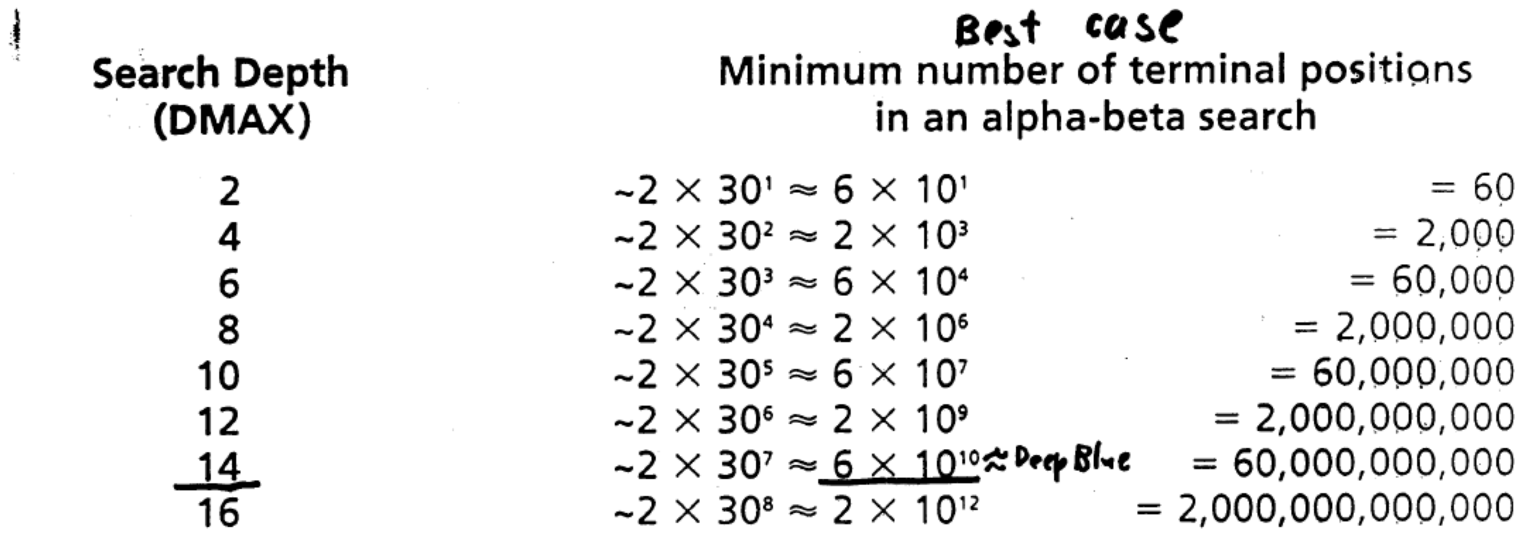
α-β -search



```
function MAX-VALUE(state, game, $\alpha$, $\beta$) returns the minimax value of state
   inputs: state, current state in game
           game, game description
           $\alpha$, the best score for MAX along the path to state
           $\beta$, the best score for MIN along the path to state

   if CUTOFF-TEST(state) then return EVAL(state)
   for each s in SUCCESSORS(state) do
       $\alpha \leftarrow$ MAX($\alpha$, MIN-VALUE(s, game, $\alpha$, $\beta$))
       if $\alpha \geq \beta$ then return $\beta$
   end
   return $\alpha$
```

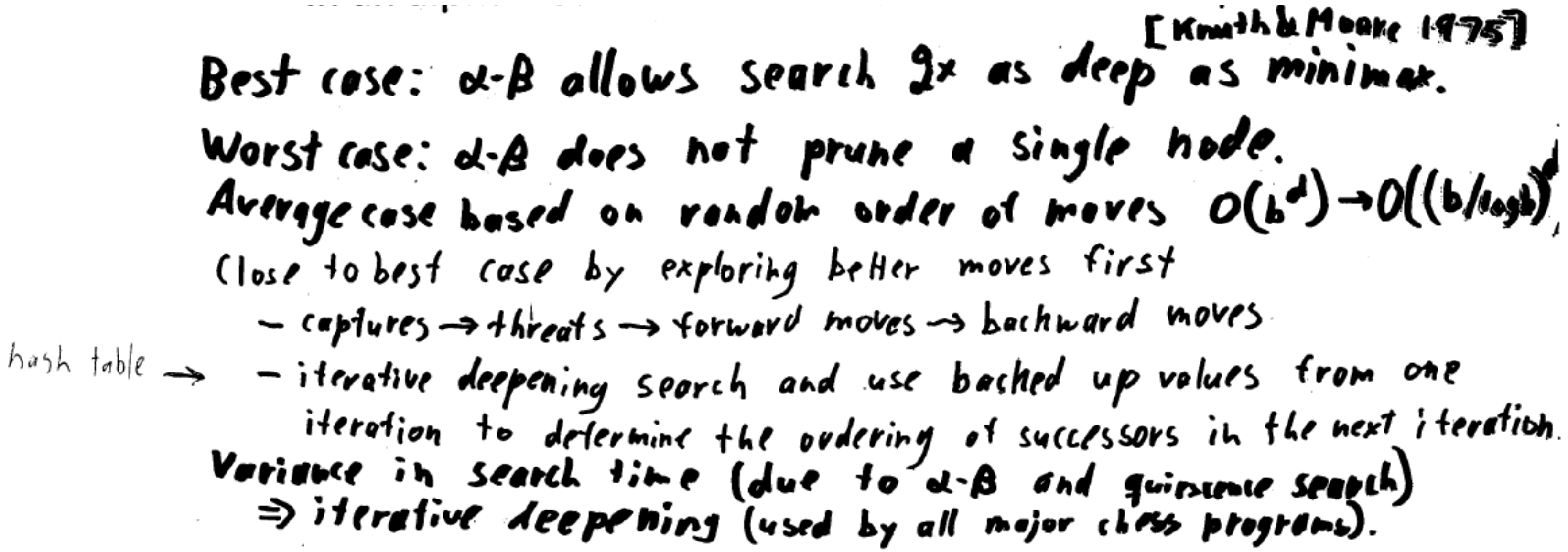
```
function MIN-VALUE(state, game, $\alpha$, $\beta$) returns the minimax value of state

   if CUTOFF-TEST(state) then return EVAL(state)
   for each s in SUCCESSORS(state) do
       $\beta \leftarrow$ MIN( $\beta$, MAX-VALUE(s, game, $\alpha$, $\beta$))
       if $\beta \leq \alpha$ then return $\alpha$
   end
   return $\beta$
```

Complexity of α-β -search



Best case

Search Depth (DMAX)	Minimum number of terminal positions in an alpha-beta search	
2	$\sim 2 \times 30^1 \approx 6 \times 10^1$	= 60
4	$\sim 2 \times 30^2 \approx 2 \times 10^3$	= 2,000
6	$\sim 2 \times 30^3 \approx 6 \times 10^4$	= 60,000
8	$\sim 2 \times 30^4 \approx 2 \times 10^6$	= 2,000,000
10	$\sim 2 \times 30^5 \approx 6 \times 10^7$	= 60,000,000
12	$\sim 2 \times 30^6 \approx 2 \times 10^9$	= 2,000,000,000
$\underline{14}$	$\sim 2 \times 30^7 \approx \underline{6 \times 10^{10}}$ ≈ Deep Blue	= 60,000,000,000
16	$\sim 2 \times 30^8 \approx 2 \times 10^{12}$	= 2,000,000,000,000



[Knuth & Moore 1975]

Best case: α-β allows search 2x as deep as minimax.

Worst case: α-β does not prune a single node.

Average case based on random order of moves $O(b^d) \rightarrow O((b/\log b)^d)$

Close to best case by exploring better moves first

- captures → threats → forward moves → backward moves

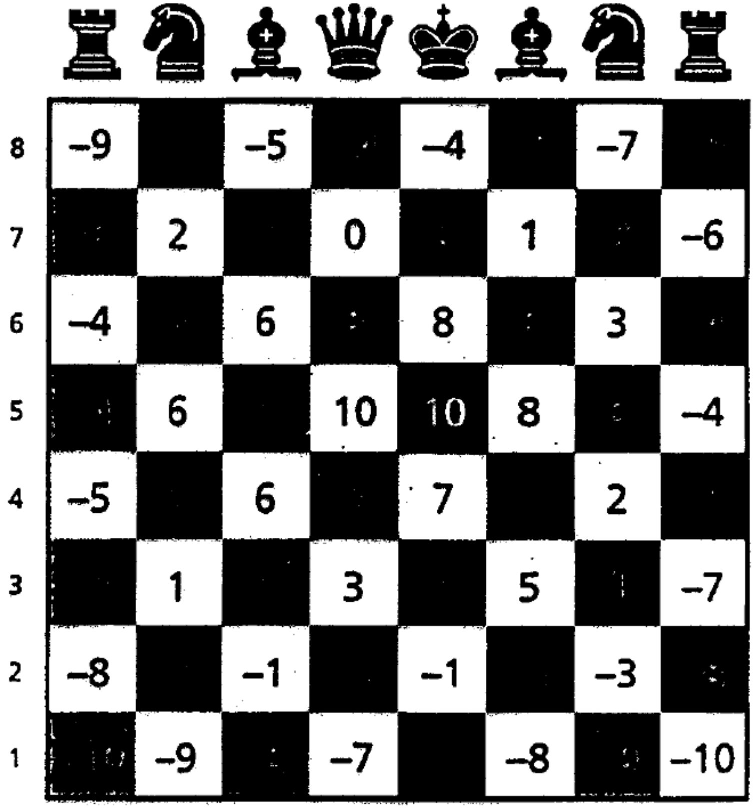
hash table → - iterative deepening search and use backed up values from one iteration to determine the ordering of successors in the next iteration.

Variance in search time (due to α-β and quiescence search)

⇒ iterative deepening (used by all major chess programs).

Evaluation function

- **Difference (between player and opponent) of**
  - **Material**
  - **Mobility**
  - **King position**
  - **Bishop pair**
  - **Rook pair**
  - **Open rook files**
  - **Control of center (piecewise)**
  - **Others**



8	–9		–5		–4		–7	
7		2		0		1		–6
6	–4		6		8		3	
5		6		10	10	8		–4
4	–5		6		7		2	
3		1		3		5		–7
2	–8		–1		–1		–3	
1		–9		–7		–8		–10

Player to move

Values of knight's position in Deep Blue

Evaluation function...

- Deep Blue used ~6,000 different features in its evaluation function (in hardware)
- A different weighting of these features is downloaded to the chips after every real world move (based on current situation on the board)
  - Contributed to strong positional play
- Acquiring the weights for Deep Blue
  - Weight learning based on a database of 900 grand master games (~120 features)
    - Alter weight of one feature => 5-6 ply search => if matches better with grand master play, then alter that parameter in the same direction further
    - Least-squares with no search
  - Manually: Grand master Joel Benjamin played take-back chess. At possible errors, the evaluation was broken down, visualized, and weighting possibly changed

    Deep Blue ~~is~~ brute force

    Smart search and knowledge engineered evaluation

  - Other learning is possible, e.g., Tesauro's Backgammon programs
    - Neurogammon [1989]
      - Taught using supervised learning on 400 games
      - Level: intermediate human player
    - TD-Gammon [1992]: Reinforcement learning; Level: world-class human tournament player

Databases of expert games

- Deep Blue does not use these during play
- Deep Blue uses them offline to learn evaluation f.

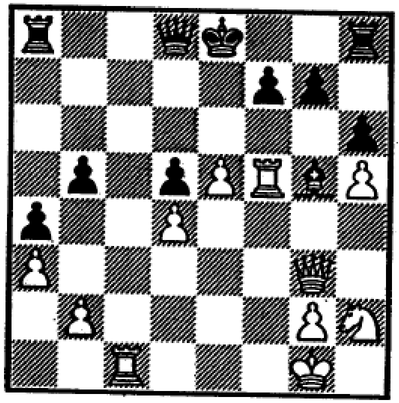
332.* **C 02**

KUPREJČIK 2520 — VLADO KOVAČEVIĆ 2545

Ljubljana/Rogaška Slatina 1989

1. e4 e6 2. d4 d5 3. e5 c5 4. c3 ♘e7 5. ♘f3 ♘ec6 6. ♗e3!? N [6. h4 — 46/343; RR 6. ♗d3 N b6 7. ♗g5 ♕d7 8. 0—0 ♗a6 9. dc5 bc5 10. ♗a6 ♘a6 11. c4 h6 12. ♗h4 ♘c7 13. ♘c3 ♗e7 14. ♗e7 ♘e7 15. ♖c1 ♖c8 16. ♕e2 0—0 17. ♖fd1 ♕c6 18. b3± Svešnikov 2435 — Lputjan 2610, Moskva (GMA) 1989] **♘d7** [6... b6] **7. ♗d3 a5** [7... ♗e7] **8. ♘bd2** [8. ♘g5!? cd4 9. cd4 ♗e7 (9... h6?! 10. ♕h5 hg5 11. ♕h8 ♘b4 12. ♕h7 g6 13. ♗g6+—) 10. h4!? (10. ♕h5? ♗g5! 11. ♗g5 ♕b6∓) ♕b6 (10... h6 11. ♕h5) 11. ♘c3±] **cd4 9. cd4 a4 10. a3** [10. ♘g5!] **♗e7 11. h4** [11. 0—0] **h6 12. h5 ♘b6∞ 13. ♘h2 ♘a5 14. ♕g4 ♗f8** [14... ♔f8 15. ♖c1 △ 0—0, f4-f5↑] **15. ♖c1** [⌓ 15. ♕e2 ♗d7 16. f4] **♗d7 16. 0—0 ♘bc4! 17. ♘c4 ♘c4 18. ♕e2** [18. ♗c4 dc4 19. d5 ed5 20. ♕d4 ♗f5! 21. g4 ♗d3∓; 19. f4!?] **b5** [18... ♖c8!?] **19. f4 ♗e7 20. f5!?** [20. ♗c4 dc4 (20... bc4 21. g4↑) 21. f5!? (21. d5 ed5 22. f5 d4! 23. ♗d4 ♗f5∓; 22. ♗d4!?∞̄) ef5 22. d5∞] **ef5** [20... ♗g5? 21. ♗c4 bc4 (21... dc4 22. d5↑) 22. ♗g5 ♕g5 23. f6±] **21. ♗f5 ♘e3 22. ♕e3 ♗g5 23. ♕g3 ♗f5 24. ♖f5**

(diagram)



24... ♖c8? [24... ♗c1! 25. ♕g7 ♖f8 *a)* 26. ♘g4 ♖a6 (26... ♗g5 27. e6 ♖a7 28. ♘c5 ♕d6 29. ef7 ♔d8 30. ♘c6±) 27. ♘f6 ♖f6 (27... ♔e7 28. ♘g8 ♔d7 29. ♖f7 ♖f7 30. ♕f7 ♔c8 31. e6 ♗e3 32. ♔f1∞) 28. ef6 ♕d6□ 29. ♖e5 ♔d8 30. ♖e7 ♖e8 31. ♖e8 ♔e8 32. ♕g8 ♕f8 33. ♕g3! ♔d7 34. ♕h3 ♔d8 35. ♕g3=; *b)* 26. e6!? ♕d6! 27. ef7!? (27. ♖f7 0-0-0 28. e7 ♖f7 29. ♕f7 ♗b2! 30. e8♕ ♗d4 31. ♔h1 ♖e8 32. ♕e8 ♔c7∓) ♔d7 28. ♘f3 (28. ♘g4!? △ ♖d5) ♔c7 29. ♖f6! ♕e7 30. ♕g6∞] **25. ♖cf1 0—0 26. e6!± ♕c7** [26... f6 27. ♕f3±] **27. ♕e1! ♕e7** [27... ♗f6 28. ♖f6 gf6 29. ♘g4 fe6 30. ♕e6 ♔g7 31. ♖f6±; 27... f6 28. ♖d5±] **28. ♖f7 ♖f7 29. ♖f7 ♖c1** [29... ♕d6 30. ♖d7 ♕b6 31. ♕e5 ♗f6 32. ♕d5+—] **30. ♕c1 ♕e6 31. ♖f4**

1 : 0 **[Kuprejčik]**

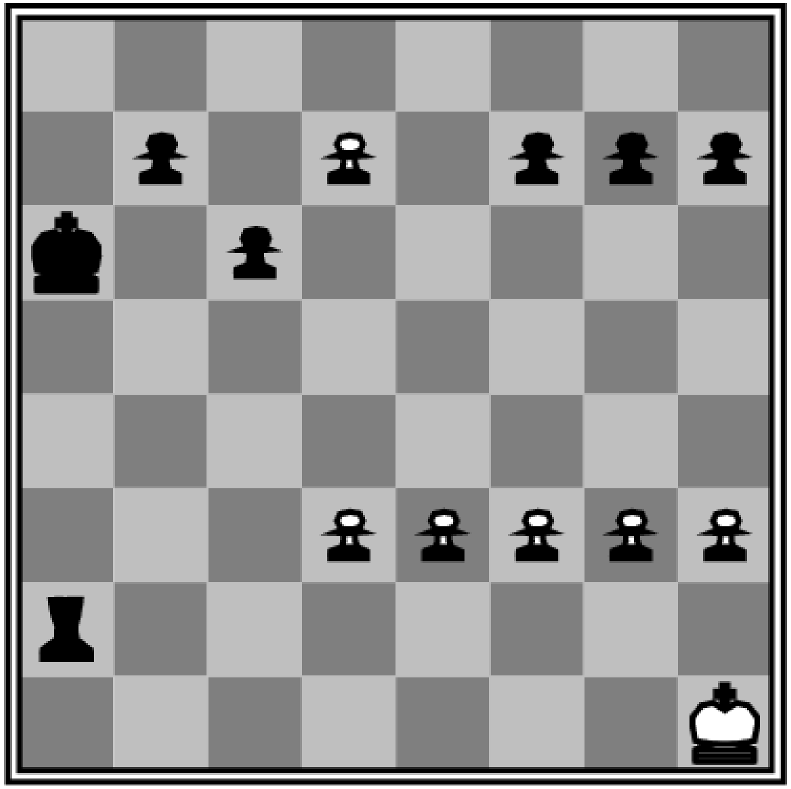
333.** **C 02**

KUPREJČIK 2520 — KOSTEN 2505

Torcy 1989

1. e4 e6 2. d4 d5 3. e5 c5 4. c3 ♘c6 5. ♘f3 ♗d7 6. ♗e2 [RR 6. ♗d3 ♘ge7 7. 0—0 cd4 8. cd4 ♘c8 N 9. ♘c3 ♗e7 10.

Horizon problem



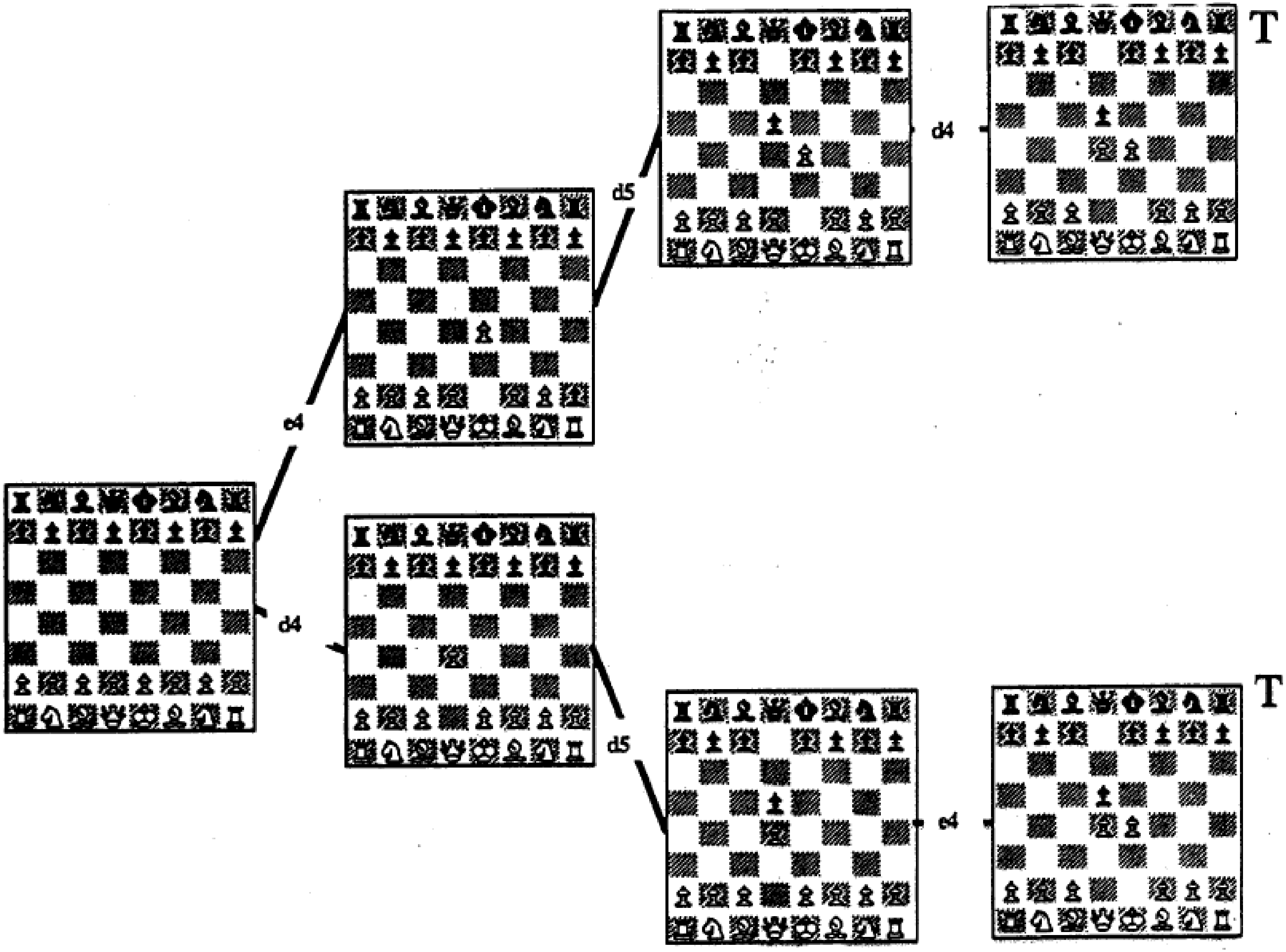
Black to move

A series of checks by the black rook forces the inevitable queening move by white "over the horizon" and makes this position look like a slight advantage for black, when it is really a sure win for white.

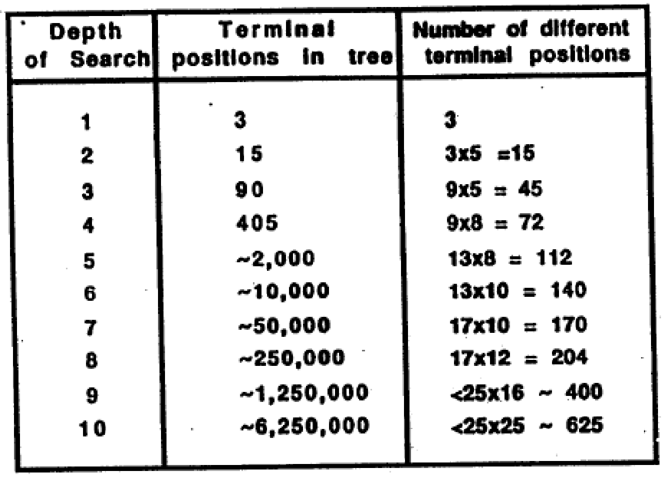
Ways to tame the horizon problem

- **Quiescence search**
  - **Evaluation function (domain specific) returns another number in addition to evaluation: stability**
    - **Threats**
    - **Other**
  - **Continue search (beyond normal horizon) if position is unstable**
  - **Introduces variance in search time**
- **Singular extension**
  - **Domain independent**
  - **A node is searched deeper if its value is much better than its siblings’**
  - **Even 30-40 ply**
  - **A variant is used by Deep Blue**

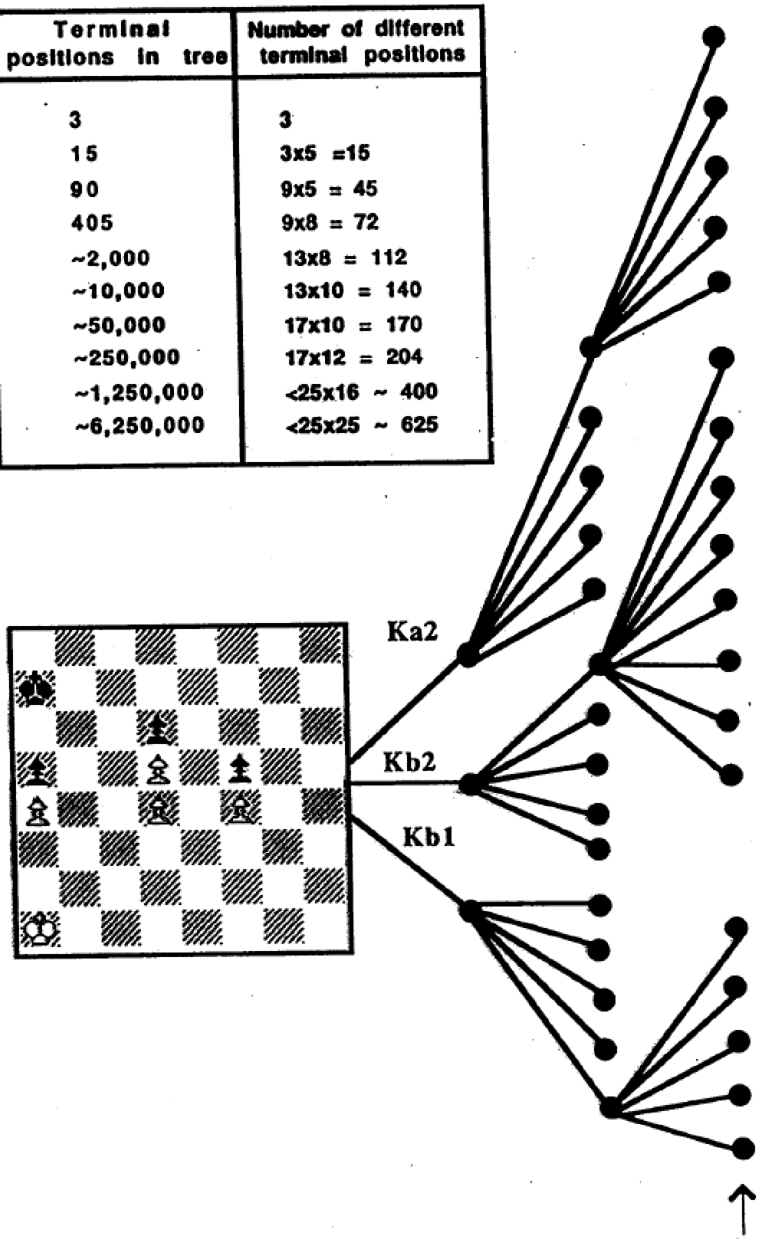
Transpositions



Transpositions are important



Depth of Search	Terminal positions in tree	Number of different terminal positions
1	3	3
2	15	3x5 =15
3	90	9x5 = 45
4	405	9x8 = 72
5	~2,000	13x8 = 112
6	~10,000	13x10 = 140
7	~50,000	17x10 = 170
8	~250,000	17x12 = 204
9	~1,250,000	<25x16 ~ 400
10	~6,250,000	<25x25 ~ 625

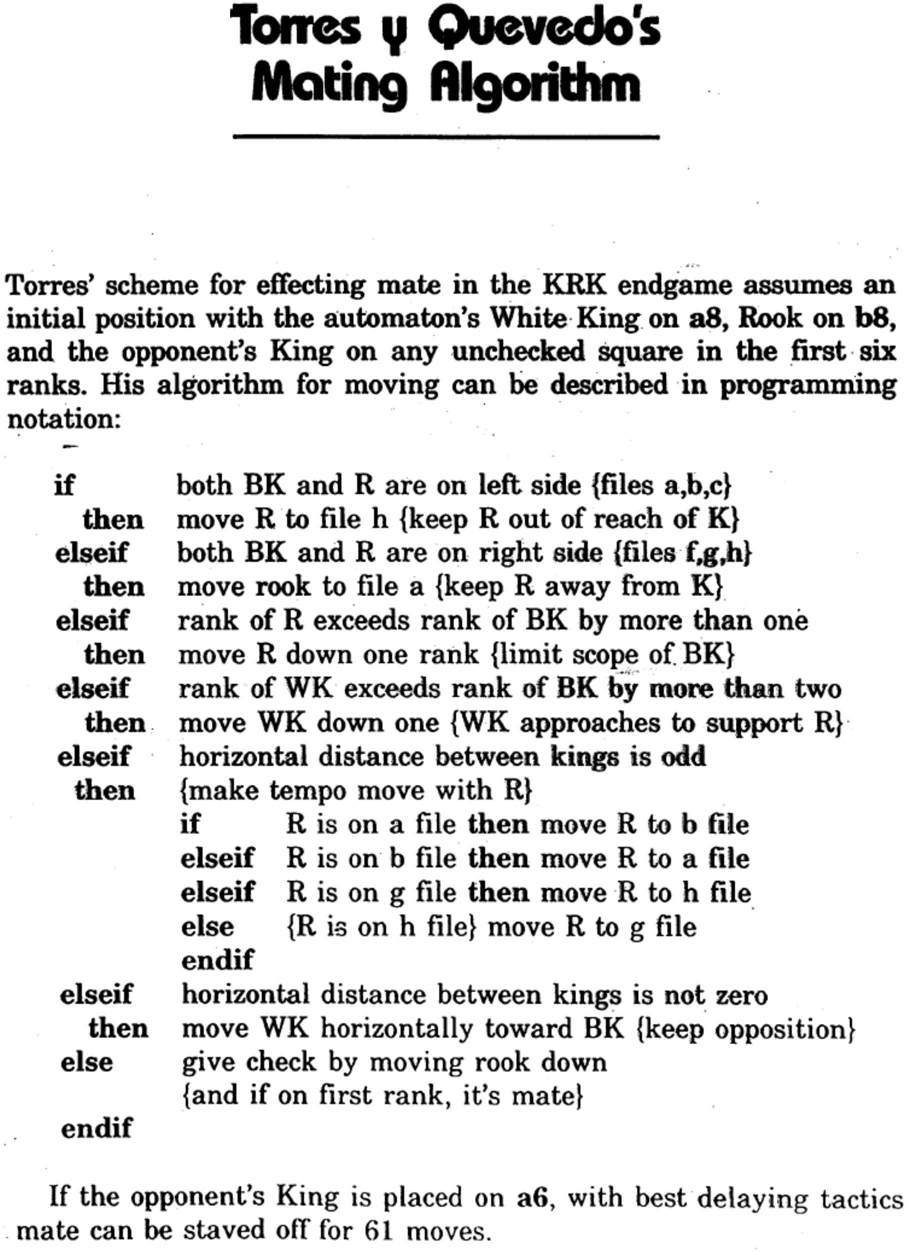


90 terminal nodes
45 different positions

Transposition table

- Store millions of positions in a hash table to avoid searching them again
  - Position
  - Hash code
  - Score
  - Exact / upper bound / lower bound
  - Depth of searched tree rooted at the position
  - Best move to make at the position
- Algorithm
  - When a position P is arrived at, the hash table is probed
  - If there is a match, and
    - new_depth(P) ≥ stored_depth(P), and
    - score in the table is exact, or the bound on the score is sufficient to cause the move leading to P to be inferior to some other choice
  - then P is assigned the attributes from the table
  - else computer scores (by direct evaluation or search (old best move searched first)) P and stores the new attributes in the table
- Fills up => replacement strategies
  - Keep positions with greater searched tree depth under them
  - Keep positions with more searched nodes under them

End game databases



Torres y Quevedo's Mating Algorithm

Torres' scheme for effecting mate in the KRK endgame assumes an initial position with the automaton's White King on **a8**, Rook on **b8**, and the opponent's King on any unchecked square in the first six ranks. His algorithm for moving can be described in programming notation:

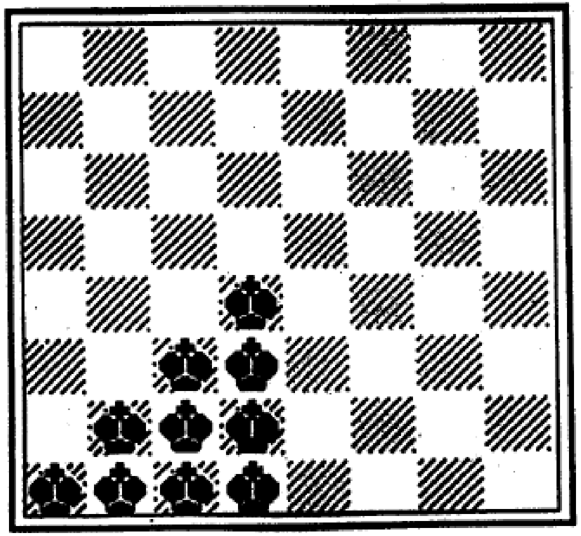
```
if        both BK and R are on left side {files a,b,c}
  then    move R to file h {keep R out of reach of K}
elseif    both BK and R are on right side {files f,g,h}
  then    move rook to file a {keep R away from K}
elseif    rank of R exceeds rank of BK by more than one
  then    move R down one rank {limit scope of BK}
elseif    rank of WK exceeds rank of BK by more than two
  then    move WK down one {WK approaches to support R}
elseif    horizontal distance between kings is odd
 then     {make tempo move with R}
          if        R is on a file then move R to b file
          elseif    R is on b file then move R to a file
          elseif    R is on g file then move R to h file
          else      {R is on h file} move R to g file
          endif
elseif    horizontal distance between kings is not zero
  then    move WK horizontally toward BK {keep opposition}
else      give check by moving rook down
          {and if on first rank, it's mate}
endif
```

If the opponent's King is placed on **a6**, with best delaying tactics mate can be staved off for 61 moves.

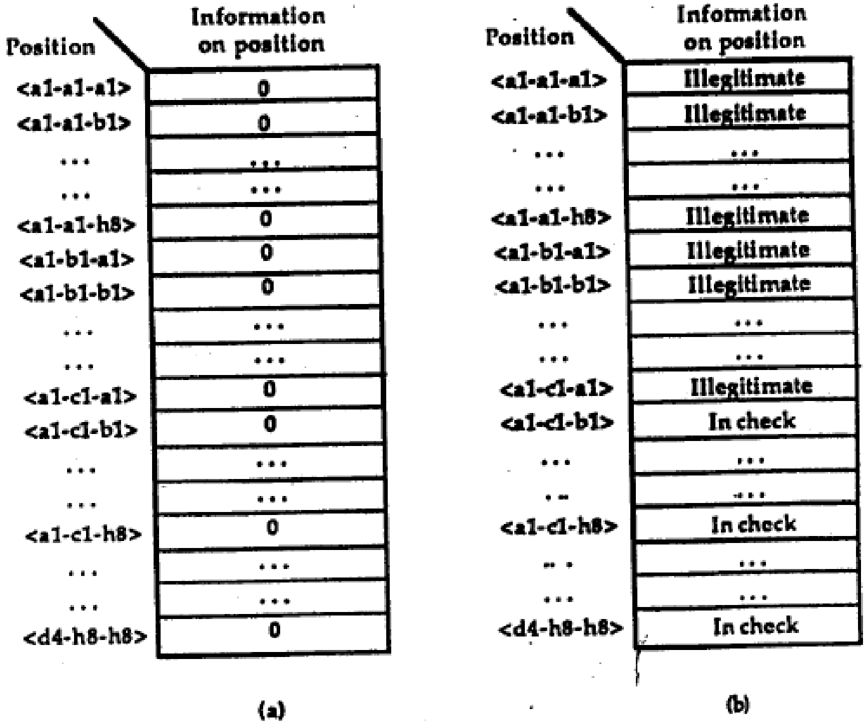
Generating databases for solvable subgames

- State space = {WTM, BTM} x {all possible configurations of remaining pieces}
- BTM table, WTM table, legal moves connect states between these
- Start at terminal positions: mate, stalemate, immediate capture without compensation (=reduction). Mark white’s wins by won-in-0
- Mark unclassified WTM positions that allow a move to a won-in-0 by won-in-1 (store the associated move)
- Mark unclassified BTM positions as won-in-2 if forced moved to won-in-1 position
- Repeat this until no more labelings occurred
- Do the same for black
- Remaining positions are draws

Compact representation methods to help endgame database representation & generation



Squares for Black's king that must be considered in KRK database.



Position	Information on position
<a1-a1-a1>	0
<a1-a1-b1>	0
...	...
...	...
<a1-a1-h8>	0
<a1-b1-a1>	0
<a1-b1-b1>	0
...	...
...	...
<a1-c1-a1>	0
<a1-c1-b1>	0
...	...
...	...
<a1-c1-h8>	0
...	...
...	...
<d4-h8-h8>	0

(a)

Position	Information on position
<a1-a1-a1>	Illegitimate
<a1-a1-b1>	Illegitimate
...	...
...	...
<a1-a1-h8>	Illegitimate
<a1-b1-a1>	Illegitimate
<a1-b1-b1>	Illegitimate
...	...
...	...
<a1-c1-a1>	Illegitimate
<a1-c1-b1>	In check
...	...
...	...
<a1-c1-h8>	In check
...	...
...	...
<d4-h8-h8>	In check

(b)

Building a KQK database: (a) initial contents of database, and (b) contents after performing the first step.

Endgame databases…

1977

Game 1 [Ken Thompson]

White: Walter Browne *Black:* BELLE

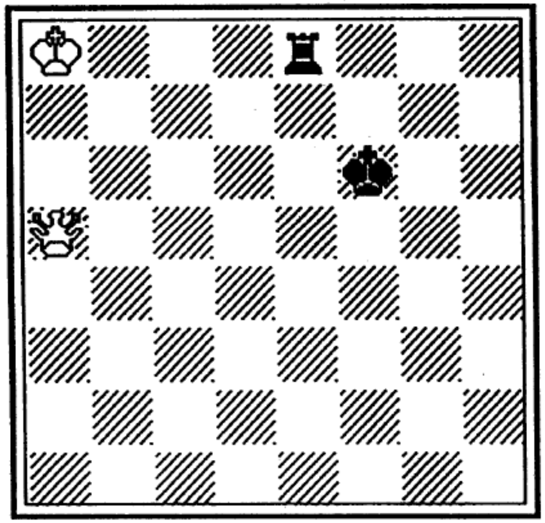


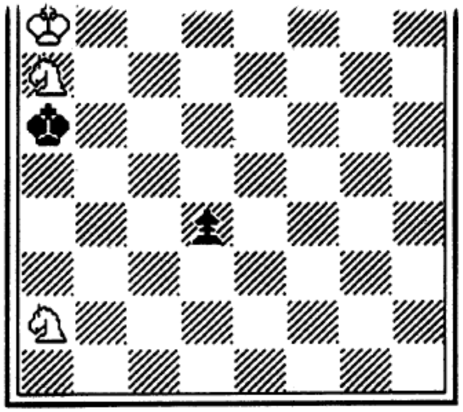
Figure 6.17. Position from BELLE's database: White to play and win in thirty moves.

Computer could hold a lost position against IM Hans Berliner.

Separated rook & King.

Folk wisdom of playing open positions?

Endgame databases…



KNNKP(d4) endgame with White to play and win

1 Nb4+ Kb6 2 Nd3 Kc7 3 Nb5+ Kc6 4 Na3 Kb6 5 Kb8 (5 Nc4+ or 5 Nc2) Kc6
6 Nc4 (6 Nc2) Kb5 7 Nce5 Kb6 8 Kc8 Ka6 (8 . . . Ka5 or 8 . . . Kb5) 9 Kc7 (9 Kd7) Kb5
10 Kd6 Ka4 11 Kc5 Kb3 12 Kb5 Kc3 13 Ka4 Kc2 14 Kb4 Kd1 15 Kb3 Kd2 16 Kb2 Kd1
17 Nc4 Ke2 18 Kc2 Kf3 19 Kd2 (19 Kd1) Kg3 (19 . . . Ke4) 20 Ke2 (20 Nce5) Kg2
21 Nce5 Kg3 22 Kf1 Kh4 23 Kg2 (23 Kf2) Kg5 24 Kf3 Kf5 25 Nc4 Kf6 26 Kf4 Ke6
27 Ke4 Kf6 28 Kd5 Ke7 29 Ke5 Kf7 30 Kd6 Kf6 31 Nd2 Kf5 32 Ke7 Kg6 33 Ke6 Kg7
(33 . . . Kg5) 34 Ne4 Kg6 35 Ke5 Kg7 36 Kd6 Kh7 (36 . . . Kh6) 37 Nd2 (37 Nef2) Kg7
38 Ke6 Kf8 39 Ne4 (39 Nc4) Ke8 40 Nf6+ (40 Nd6+) Kf8 (40 . . . Kd8) 41 Nh5 Ke8
42 Ng7+ Kd8 43 Kd6 Kc8 44 Ne6 Kb8 (44 . . . Kb7) 45 Kc5 Ka7 46 Kc6 Ka6 47 Nec5+
(47 Ng5) Ka5 48 Nb3+ (48 Ne4) Ka4 49 Nd2 Ka5 50 Kc5 Ka6 51 Nc4 Kb7 52 Kd6 Kc8
53 Na5 Kd8 54 Nb7+ Ke8 55 Ke6 Kf8 56 Nd6 Kg7 57 Kf5 Kh6 58 Kf6 Kh5 59 Nf7
(59 Ne4) Kg4 60 Ng5 Kh4 61 Kf5 Kg3 62 Ke4 Kg4 63 Nf7 Kh5 (63 . . . Kg3) 64 Kf5 Kh4
65 Nfe5 Kh5 66 Ng4 Kh4 67 Nf6 Kh3 68 Ke5 Kg3 69 Ke4 Kh3 70 Kf3 Kh4 71 Kf4 Kh3
72 Ne8 (72 Ne4 or 72 Nh5) Kh4 73 Ng7 Kh3 74 Nf5 Kg2 (74 . . . Kh2) 75 Kg4 Kh2
(75 . . . Kf1 or 75 . . . Kg1 or 75 . . . Kh1) 76 Nd6 (76 Ng3) Kg2 (76 . . . Kg1 or 76 . . . Kh1)
77 Nc4 (77 Ne4) Kh2 (77 . . . Kg1) 78 Nd2 Kg2 79 Kh4 Kh2 (79 . . . Kg1) 80 Nf4
(80 Ne1) Kg1 81 Kg3 Kh1 82 Nf3 (82 Ne2 or 82 Nh3) d3 followed by 83 Nh3 d2
84 Nf2#.

How end game databases changed chess

- All 5 piece endgames solved (can have > 10^8 states) & many 6 piece
  - KRBKNN (~10^{11} states): longest path-to-reduction 223
- Rule changes
  - Max number of moves from capture/pawn move to completion
- Chess knowledge
  - Splitting rook from king in KRKQ
  - KRKN game was thought to be a draw, but
    - White wins in 51% of WTM
    - White wins in 87% of BTM

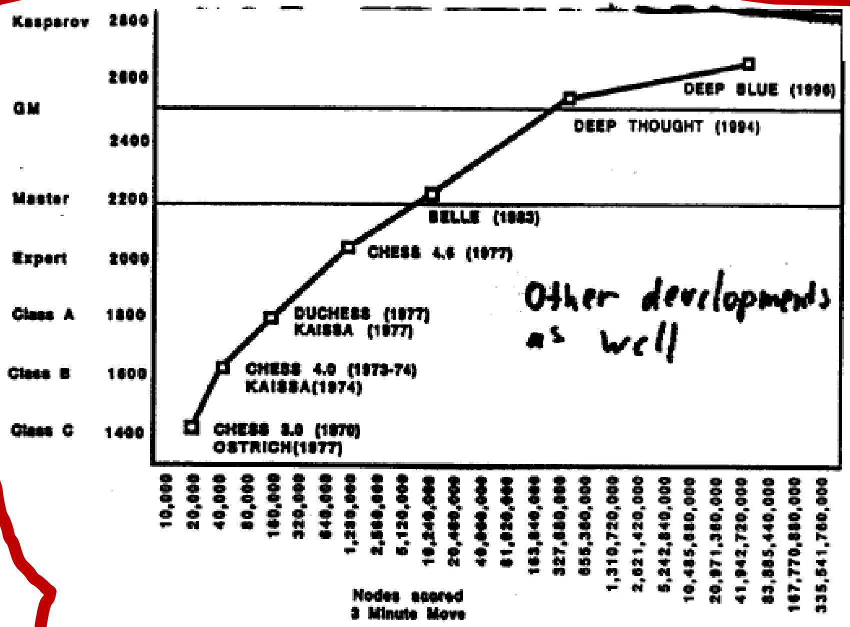
Deep Blue’s search

- ~200 million moves / second = 3.6 * 10^{10} moves in 3 minutes
- 3 min corresponds to
  - ~7 plies of uniform depth minimax search
  - 10-14 plies of uniform depth alpha-beta search
- 1 sec corresponds to 380 years of human thinking time
- Software searches first
  - Selective and singular extensions
- Specialized hardware searches last 5 ply

Deep Blue’s hardware

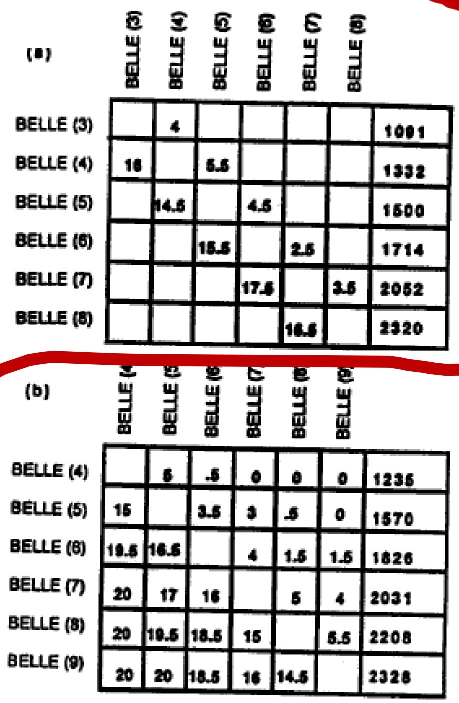
- 32-node RS6000 SP multicomputer
- Each node had
  - 1 IBM Power2 Super Chip (P2SC)
  - 16 chess chips
    - Move generation (often takes 40-50% of time)
    - Evaluation
    - Some endgame heuristics & small endgame databases
- 32 Gbyte opening & endgame database

Role of computing power



Other developments as well

Figure 6.23. Relationship between the level of play by chess programs and the size of the tree searched during a three minute move.



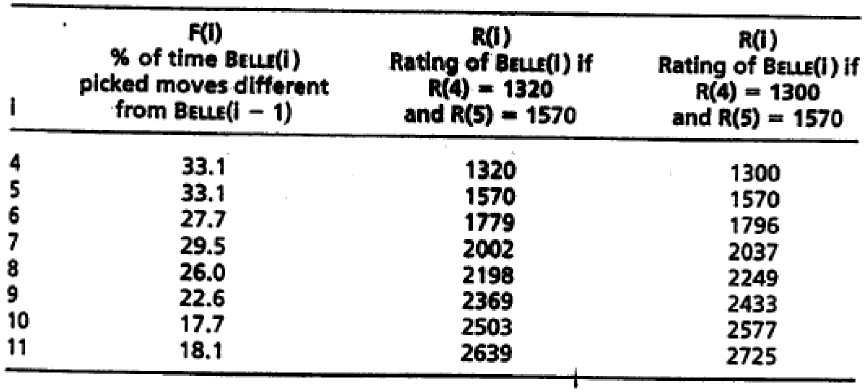
(a)

	BELLE (3)	BELLE (4)	BELLE (5)	BELLE (6)	BELLE (7)	BELLE (8)	
BELLE (3)		4					1091
BELLE (4)	16		5.5				1332
BELLE (5)		14.5		4.5			1500
BELLE (6)			15.5		2.5		1714
BELLE (7)				17.5		3.5	2052
BELLE (8)					16.5		2320

(b)

	BELLE (4)	BELLE (5)	BELLE (6)	BELLE (7)	BELLE (8)	BELLE (9)	
BELLE (4)		5	.5	0	0	0	1235
BELLE (5)	15		3.5	3	.5	0	1570
BELLE (6)	19.5	16.5		4	1.5	1.5	1826
BELLE (7)	20	17	16		5	4	2031
BELLE (8)	20	19.5	18.5	15		5.5	2208
BELLE (9)	20	20	18.5	16	14.5		2328

Figure 6.24. Results of Thompson's two experiments: (a) first experiment, (b) second experiment. Entries in the tables indicate the number of games won by the program heading the row against the program heading the column.



i	F(i) % of time BELLE(i) picked moves different from BELLE(i − 1)	R(i) Rating of BELLE(i) if R(4) = 1320 and R(5) = 1570	R(i) Rating of BELLE(i) if R(4) = 1300 and R(5) = 1570
4	33.1	1320	1300
5	33.1	1570	1570
6	27.7	1779	1796
7	29.5	2002	2037
8	26.0	2198	2249
9	22.6	2369	2433
10	17.7	2503	2577
11	18.1	2639	2725

Figure 6.25. Percentage of time BELLE(i) picked different moves from BELLE(i − 1) and the corresponding predicted ratings based on expression (1) for two cases: (1) R(4) = 1320 and R(5) = 1570, and (2) R(4) = 1300 and R(5) = 1570.

Diminishing returns to computation power.

Interestingly… “Freestyle Chess” = centaurs

- Hybrid human-AI chess players were stronger **for a while** than humans or AI alone

AlphaGo and AlphaZero

LEE SEDOL
00:01:00

MCTS Overview

- Iteratively building partial search tree
- Iteration
  - Most urgent node
    - Tree policy
    - Exploration/exploitation
  - Simulation
    - Add child node
    - Default policy
  - Update weights

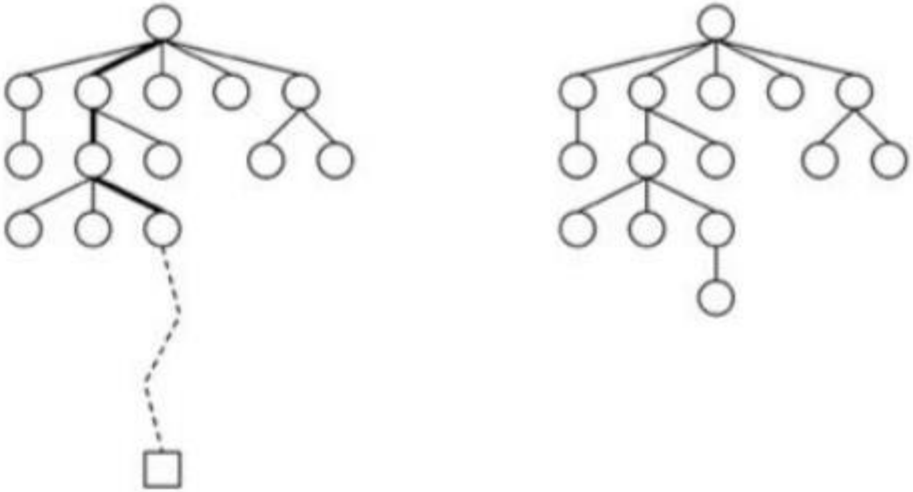


Fig. 1. The basic MCTS process [17].

Algorithm Overview

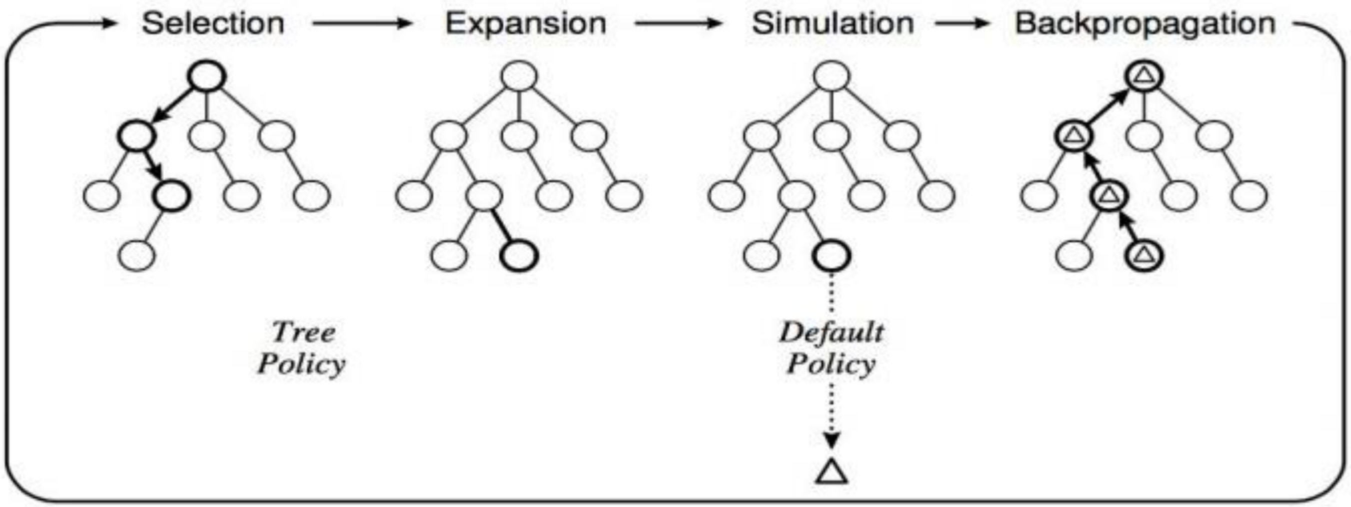


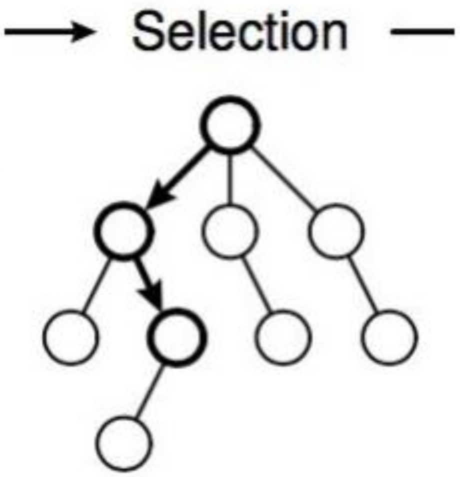
Fig. 2. One iteration of the general MCTS approach.

Policies

- Policies are crucial for how MCTS operates
- Tree policy
    - Used to determine how children are selected
- Default policy
    - Used to determine how simulations are run (ex. randomized)
    - Result of simulation used to update values

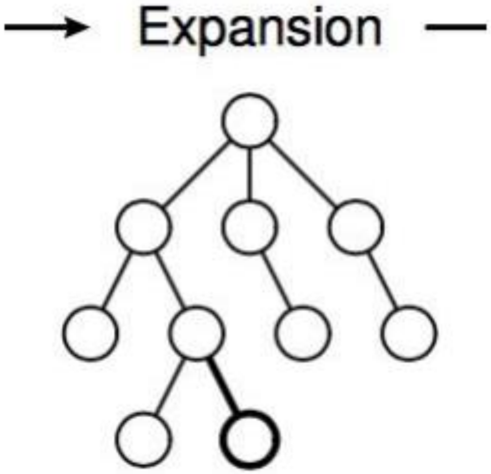
Selection

- Start at root node
- Based on Tree Policy select child
- Apply recursively - descend through tree
    - Stop when expandable node is reached
    - *Expandable* -
        - Node that is non-terminal and has unexplored children



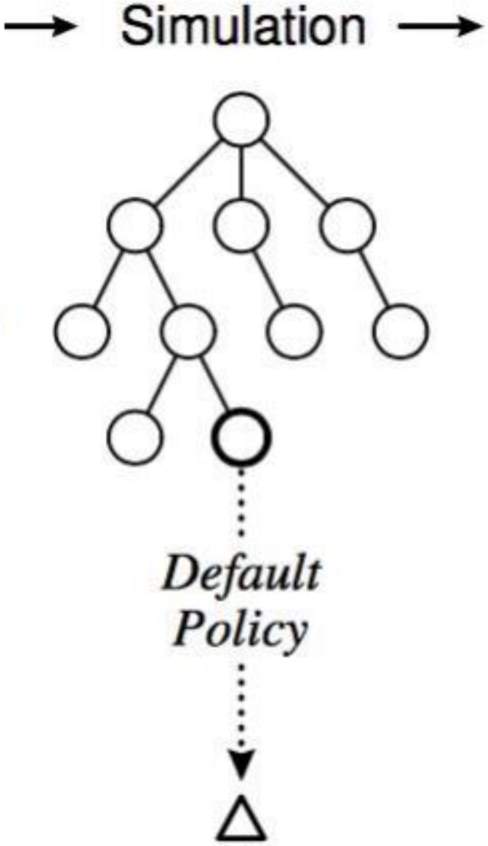
Expansion

- Add one or more child nodes to tree
  - Depends on what actions are available for the current position
  - Method in which this is done depends on Tree Policy



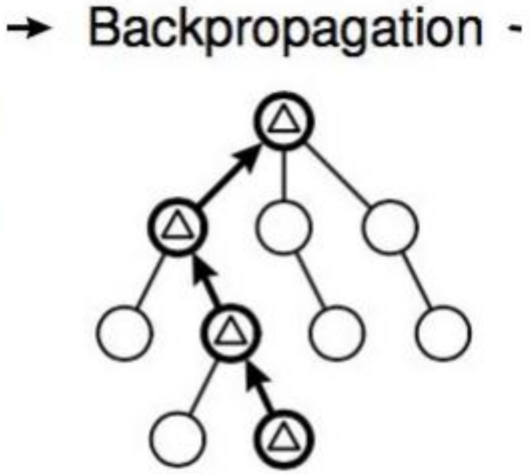
Simulation

- Runs simulation of path that was selected
- Get position at end of simulation
- Default Policy determines how simulation is run
- Board outcome determines value

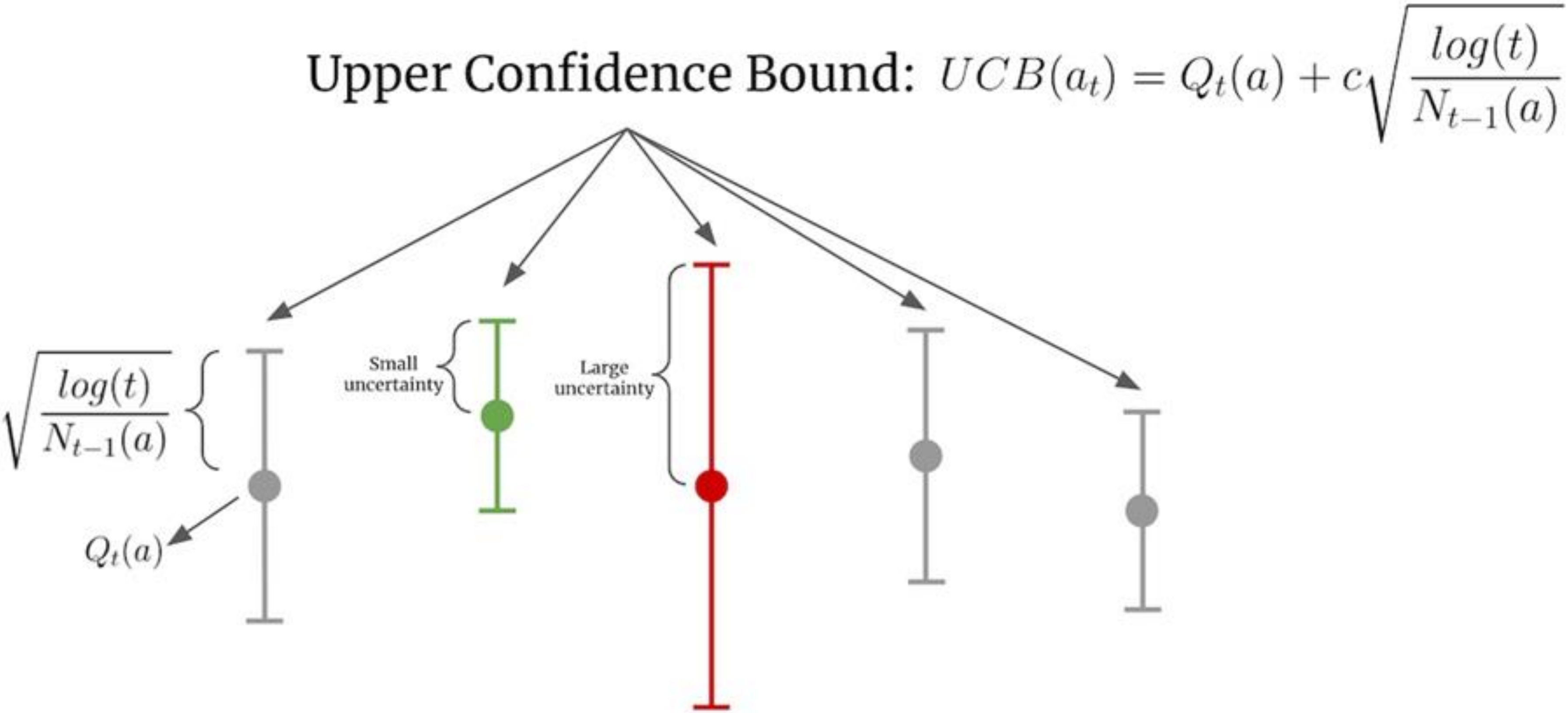


Backpropagation

- Moves backward through saved path
- Value of Node
  - representative of benefit of going down that path from parent
- Values are updated dependent on board outcome
  - Based on how the simulated game ends, values are updated



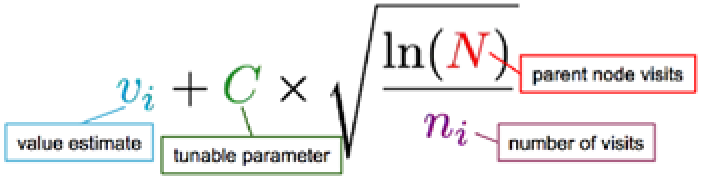
UCB in Bandits



Upper Confidence Bound: $UCB(a_t) = Q_t(a) + c\sqrt{\frac{log(t)}{N_{t-1}(a)}}$

UCT Algorithm

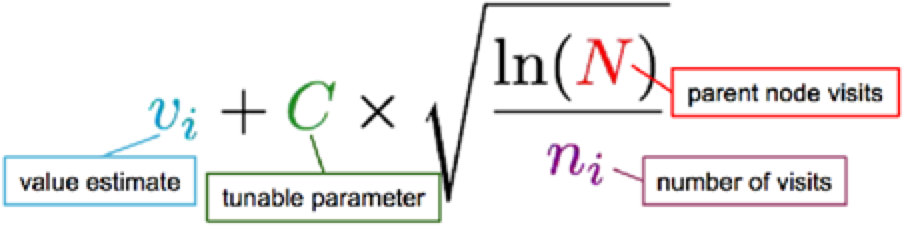
- Selecting child node: multi-armed bandit problem
- UCB for child selection
- UCT



$$v_i + C \times \sqrt{\frac{\ln(N)}{n_i}}$$

- v: value estimate
- C: exploration parameter
- N: number of parent node visits
- n: number of visits

UCT Algorithm



$$v_i + C \times \sqrt{\frac{\ln(N)}{n_i}}$$

- n = 0 means infinite weight
  - Guarantees we explore each child at least once
- Each child has non-zero probability of selection
- Adjust C to change explore-exploit tradeoff

Theorem. MCTS with UCT action selection in the Selection phase finds an optimal policy. [Kocsis and Szepesvári. ECML ‘06]

Example - The Game of Othello

$$v_i + C \times \sqrt{\frac{\ln(N)}{n_i}}$$

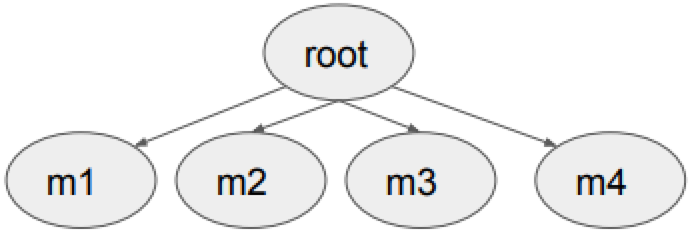
value estimate

tunable parameter

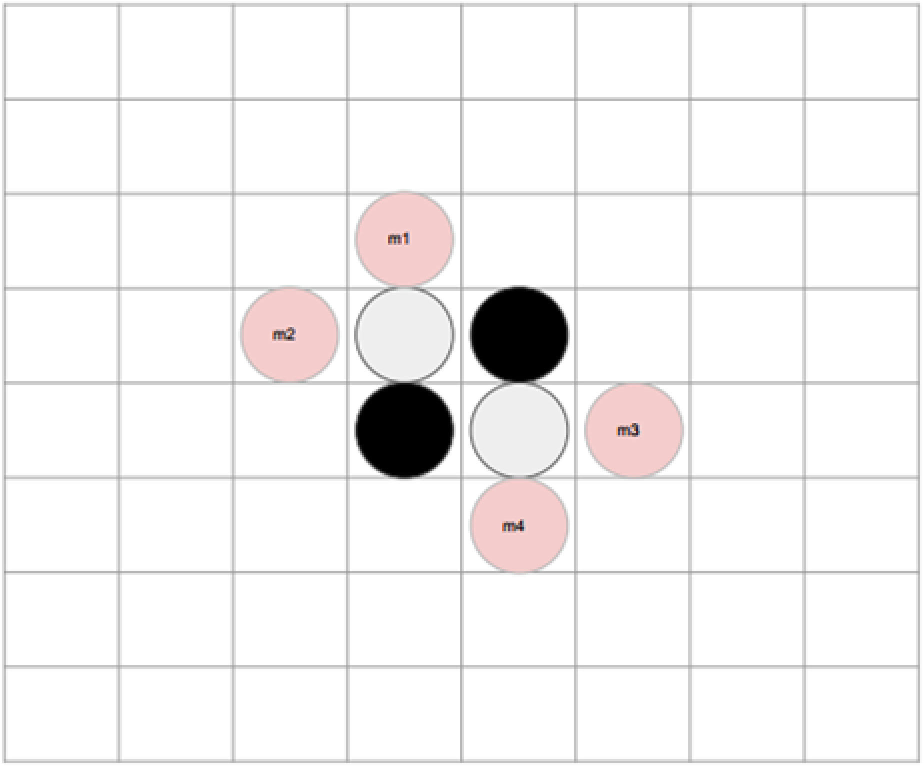
parent node visits

number of visits

(X_j, n, n_j) - (Mean Value, Parent Visits, Child Visits)

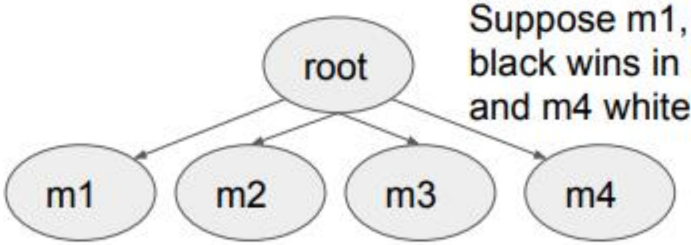


- n_j - initially 0
  - all weights are initially infinity
- n - initially 0
- C_p - some constant > 0
  - For this example
  - C = (1 / 2√2)
- X_j - mean reward of selecting this position
  - [0, 1]
  - Initially N/A



Example - The Game of Othello cont.

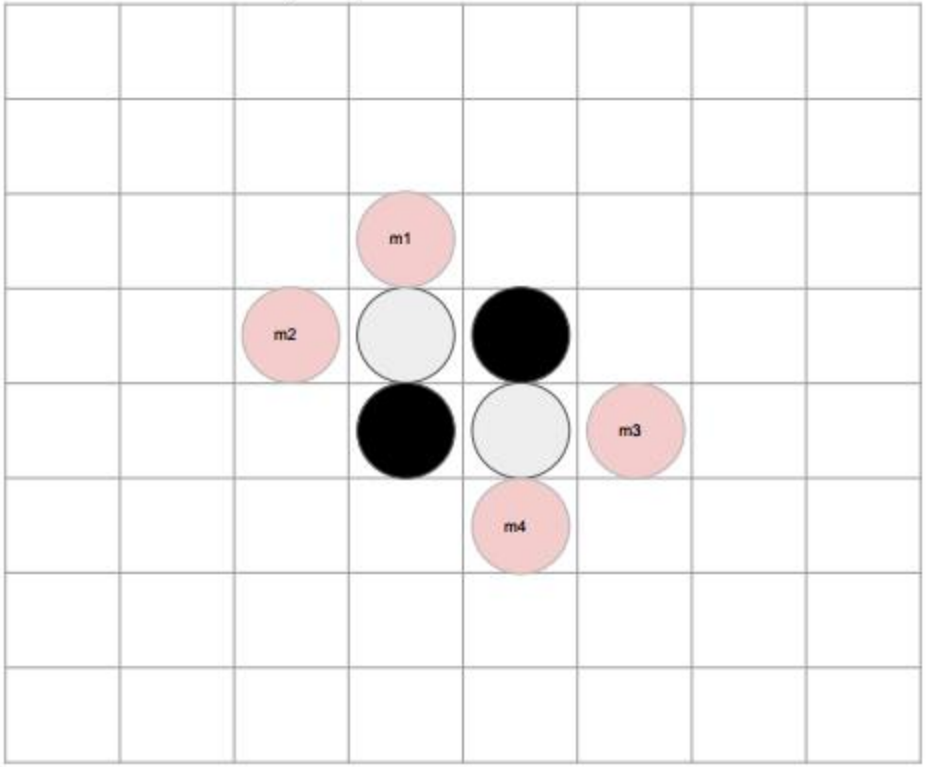
$$\underbrace{v_i}_{\text{value estimate}} + \underbrace{C}_{\text{tunable parameter}} \times \sqrt{\frac{\ln(\overset{\text{parent node visits}}{N})}{\underset{\text{number of visits}}{n_i}}}$$



After first 4 iterations:
Suppose m1, m2, m3 black wins in simulation and m4 white wins

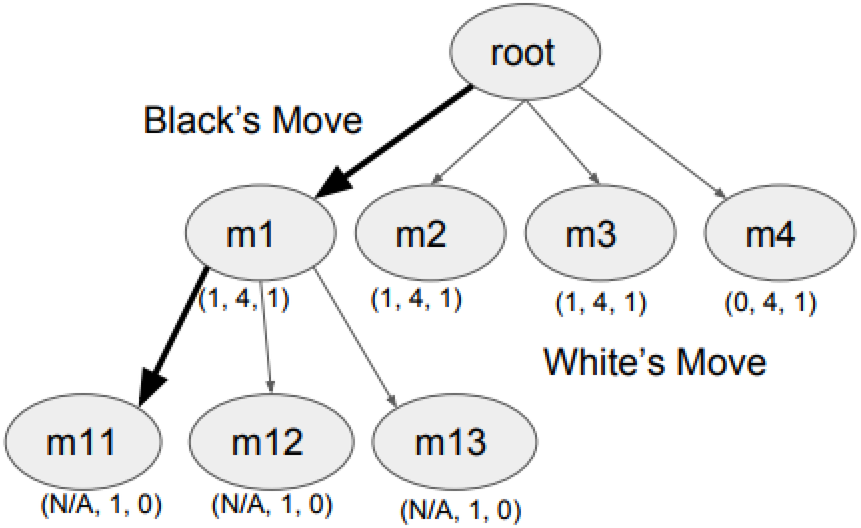
	X_j	n	n_j
m1	1	4	1
m2	1	4	1
m3	1	4	1
m4	0	4	1

(X_j, n, n_j) - (Mean Value, Parent Visits, Child Visits)

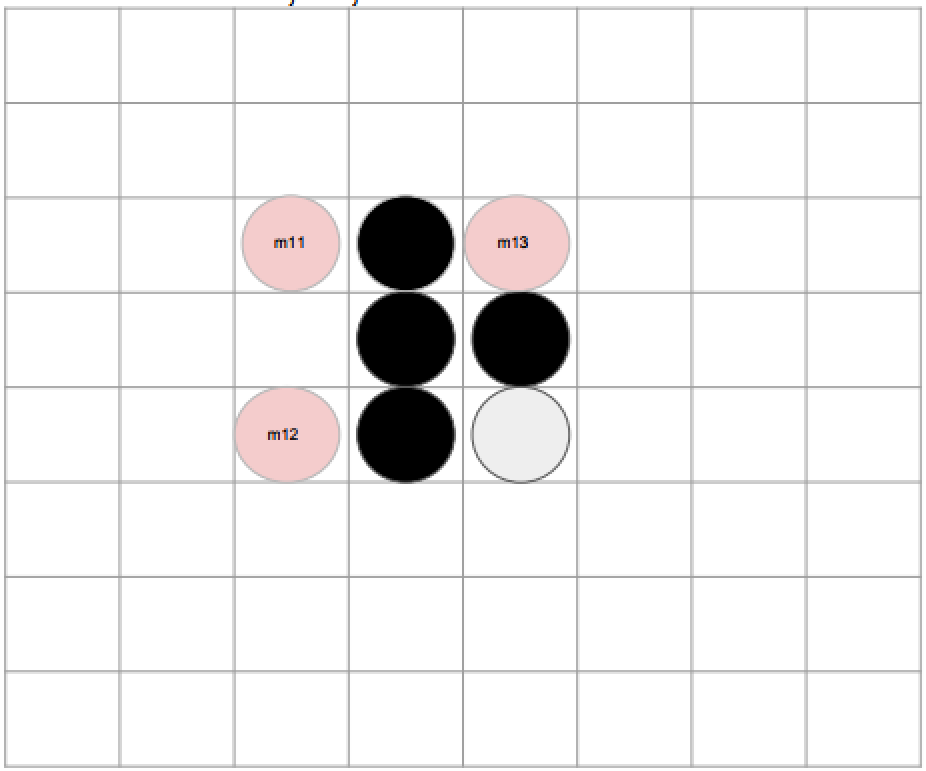


Example - The Game of Othello Iter #5

$$\underbrace{v_i}_{\text{value estimate}} + \underbrace{C}_{\text{tunable parameter}} \times \sqrt{\frac{\ln(\underbrace{N}_{\text{parent node visits}})}{\underbrace{n_i}_{\text{number of visits}}}}$$



(X_j, n, n_j) - *(Mean Value, Parent Visits, Child Visits)*



- First selection picks m1
- Second selection picks m11

Example - The Game of Othello Iter #5

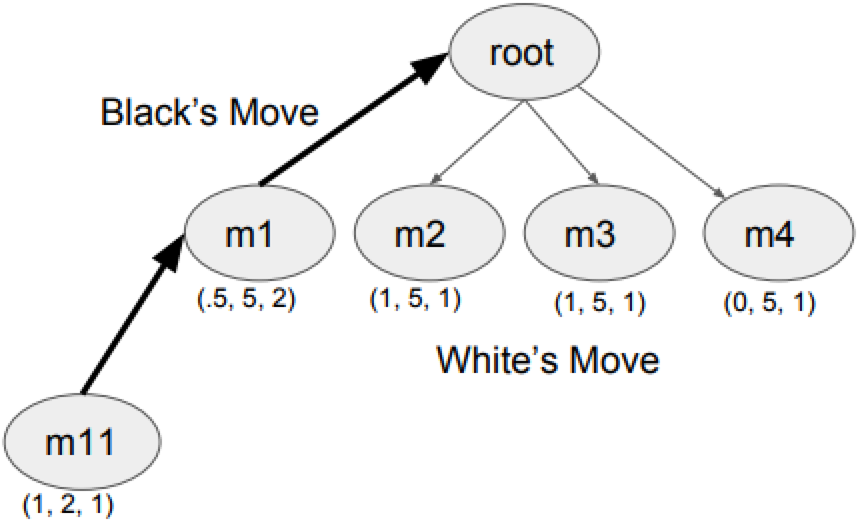
$$v_i + C \times \sqrt{\frac{\ln(N)}{n_i}}$$

value estimate

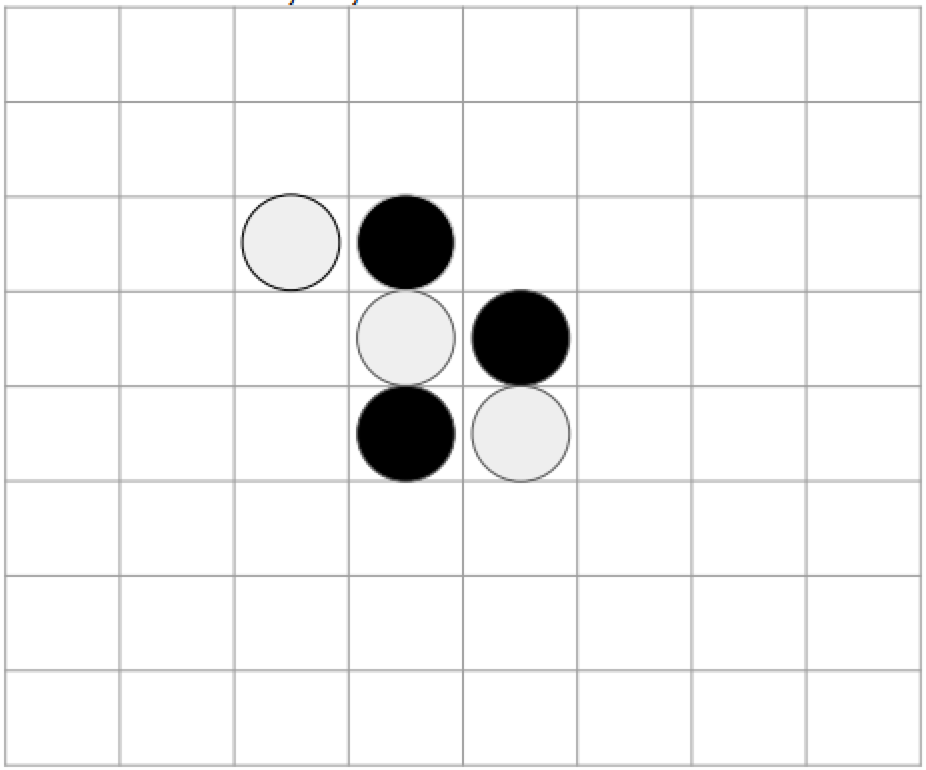
tunable parameter

parent node visits

number of visits



(X_i, n, n_i) - (Mean Value, Parent Visits, Child Visits)



- Run a simulation
- White Wins
- Backtrack, and update mean scores accordingly.

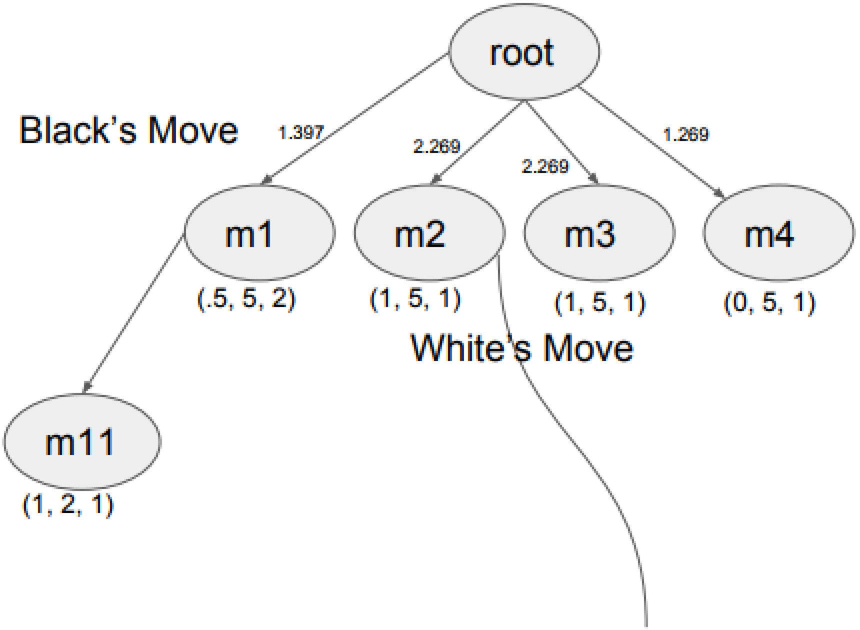
Example - The Game of Othello Iter #6

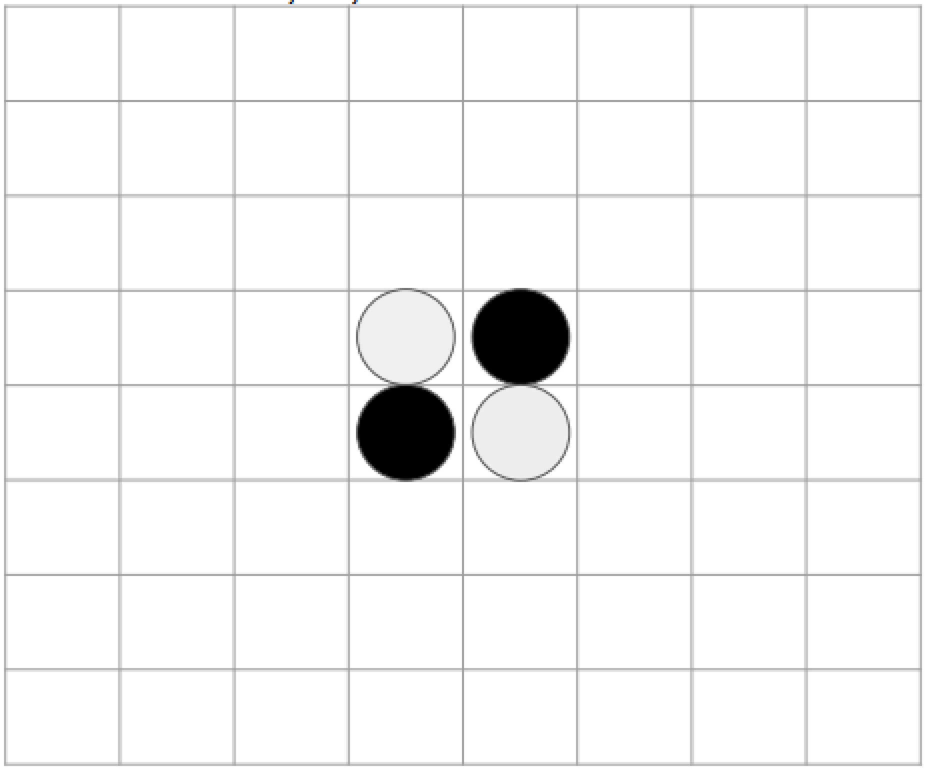
$$\underbrace{v_i}_{\text{value estimate}} + \underbrace{C}_{\text{tunable parameter}} \times \sqrt{\frac{\ln(N)}{n_i}}$$

parent node visits

number of visits

(X_j, n, n_j) - (Mean Value, Parent Visits, Child Visits)





- Suppose we first select m2

Example - The Game of Othello Iter #6

$$v_i + C \times \sqrt{\frac{\ln(N)}{n_i}}$$

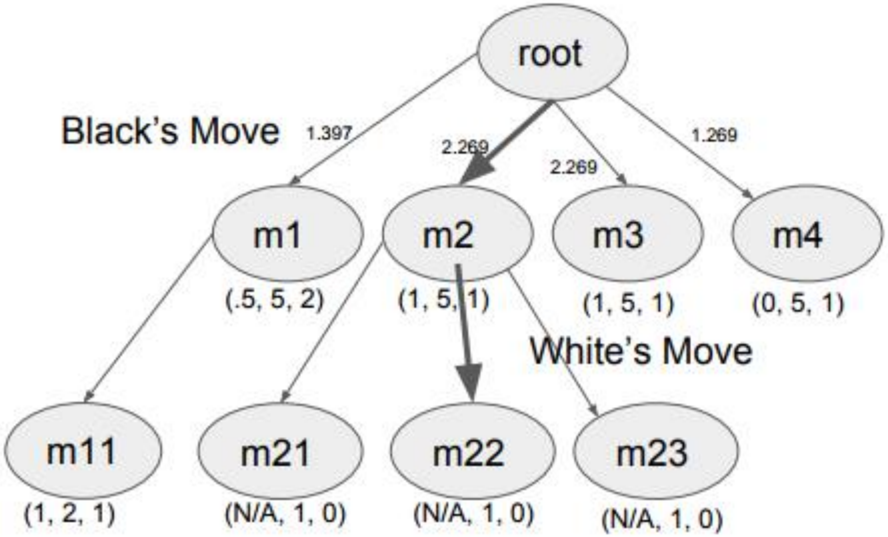
value estimate

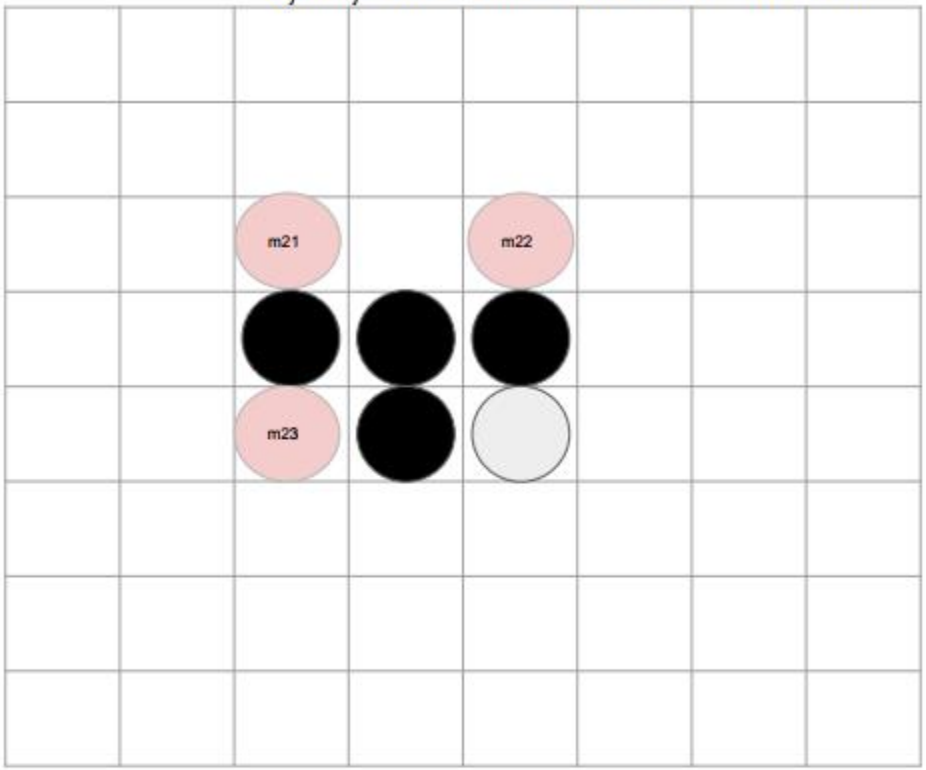
tunable parameter

parent node visits

number of visits

(X_j, n, n_j) - *(Mean Value, Parent Visits, Child Visits)*





- Suppose we pick m22

Example - The Game of Othello Iter #6

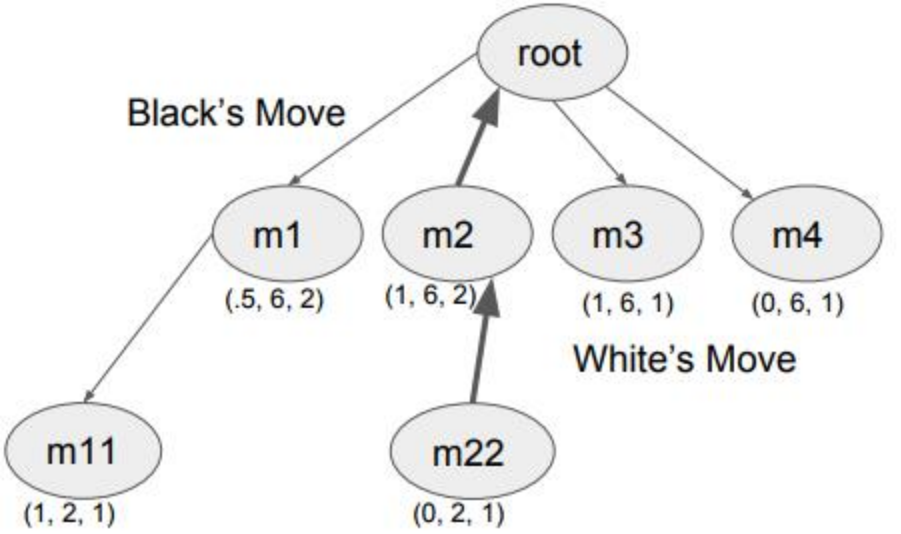
$$v_i + C \times \sqrt{\frac{\ln(N)}{n_i}}$$

value estimate

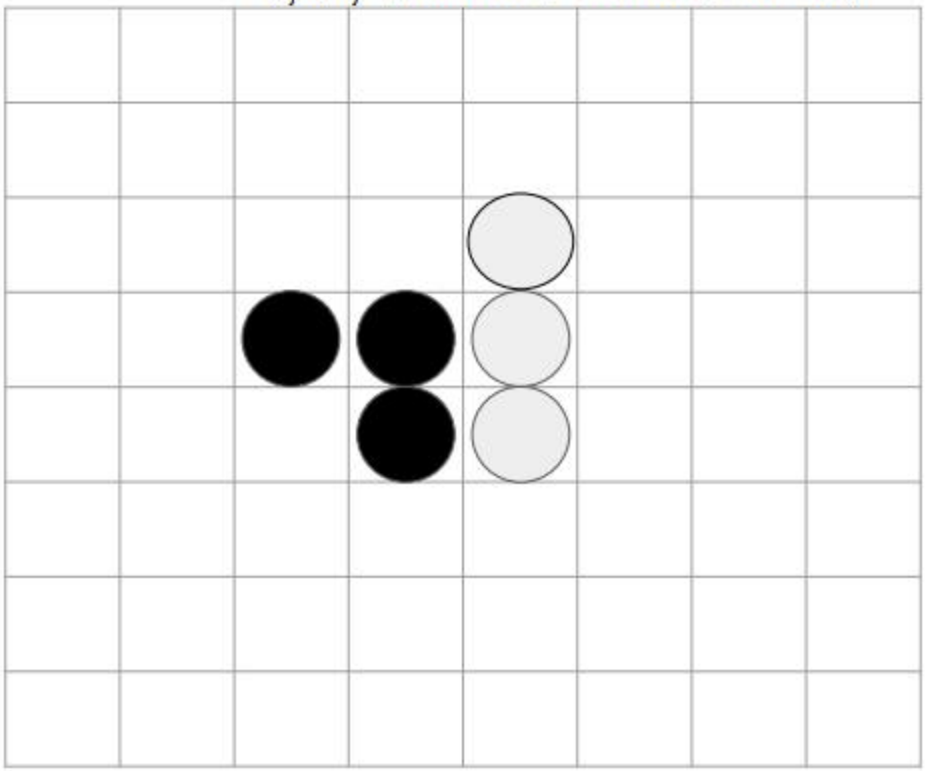
tunable parameter

parent node visits

number of visits



(X_j, n, n_j) - (Mean Value, Parent Visits, Child Visits)



- Run simulated game from this position.
- Suppose black wins the simulated game.
- Backtrack and update values

Example - The Game of Othello Iter #6

$$v_i + C \times \sqrt{\frac{\ln(N)}{n_i}}$$

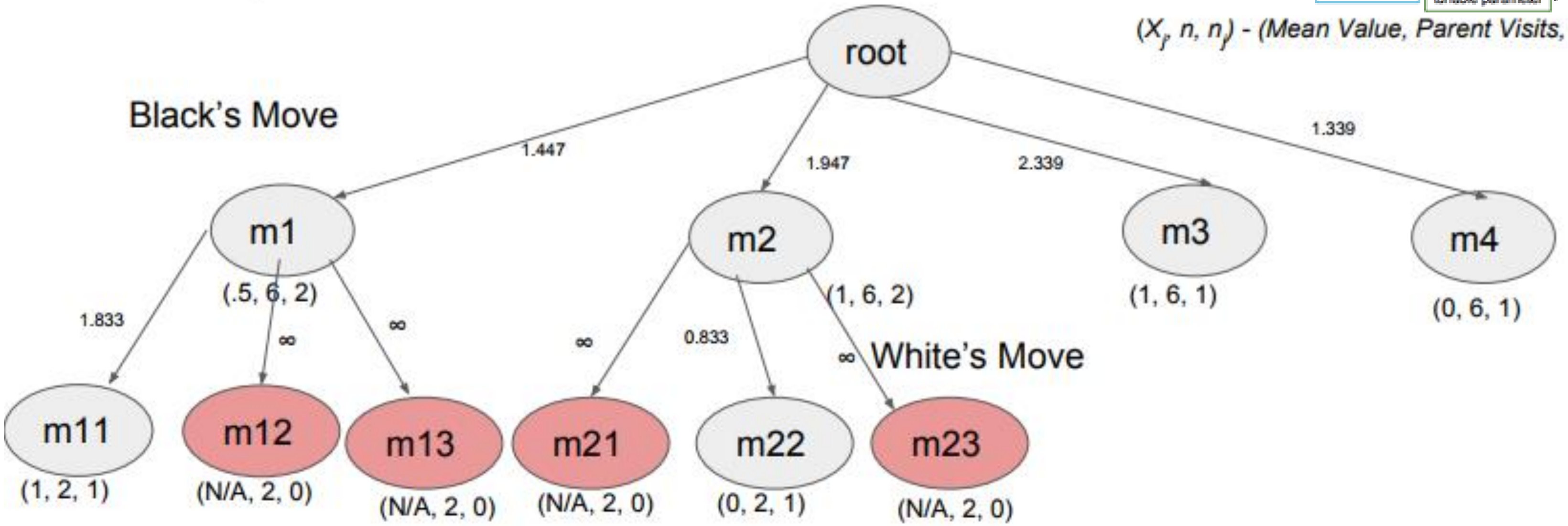
value estimate

tunable parameter

parent node visits

number of visits

(X_j, n, n_j) - (Mean Value, Parent Visits, Child Visits)

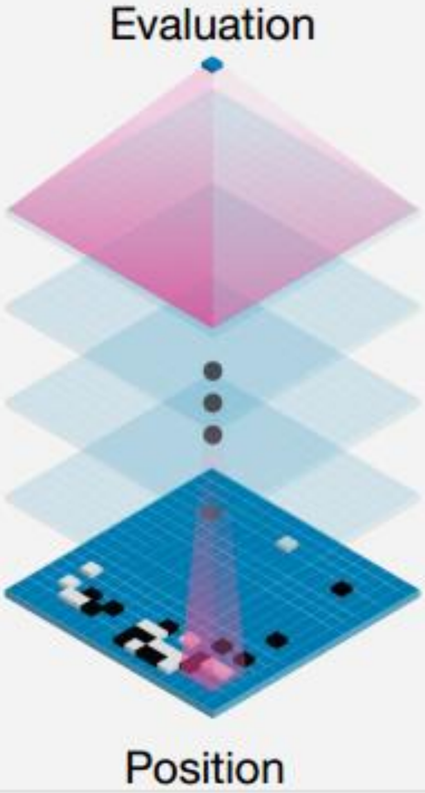


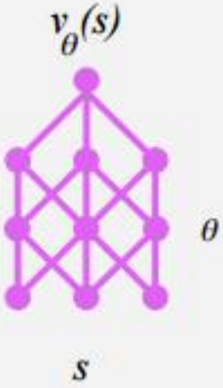
- This is how our tree looks after 6 iterations.
- Red Nodes not actually in tree
- Now given a tree, actual moves can be made using max, robust, max-robust, or other child selection policies.
- Only care about subtree after moves have been made

AlphaGo

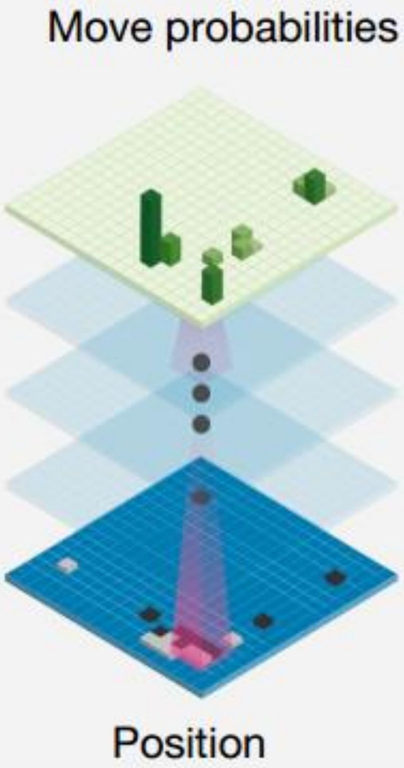
- Use value network and policy network to augment MCTS
- Trained on professional Go games

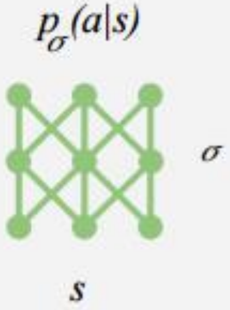
Value network

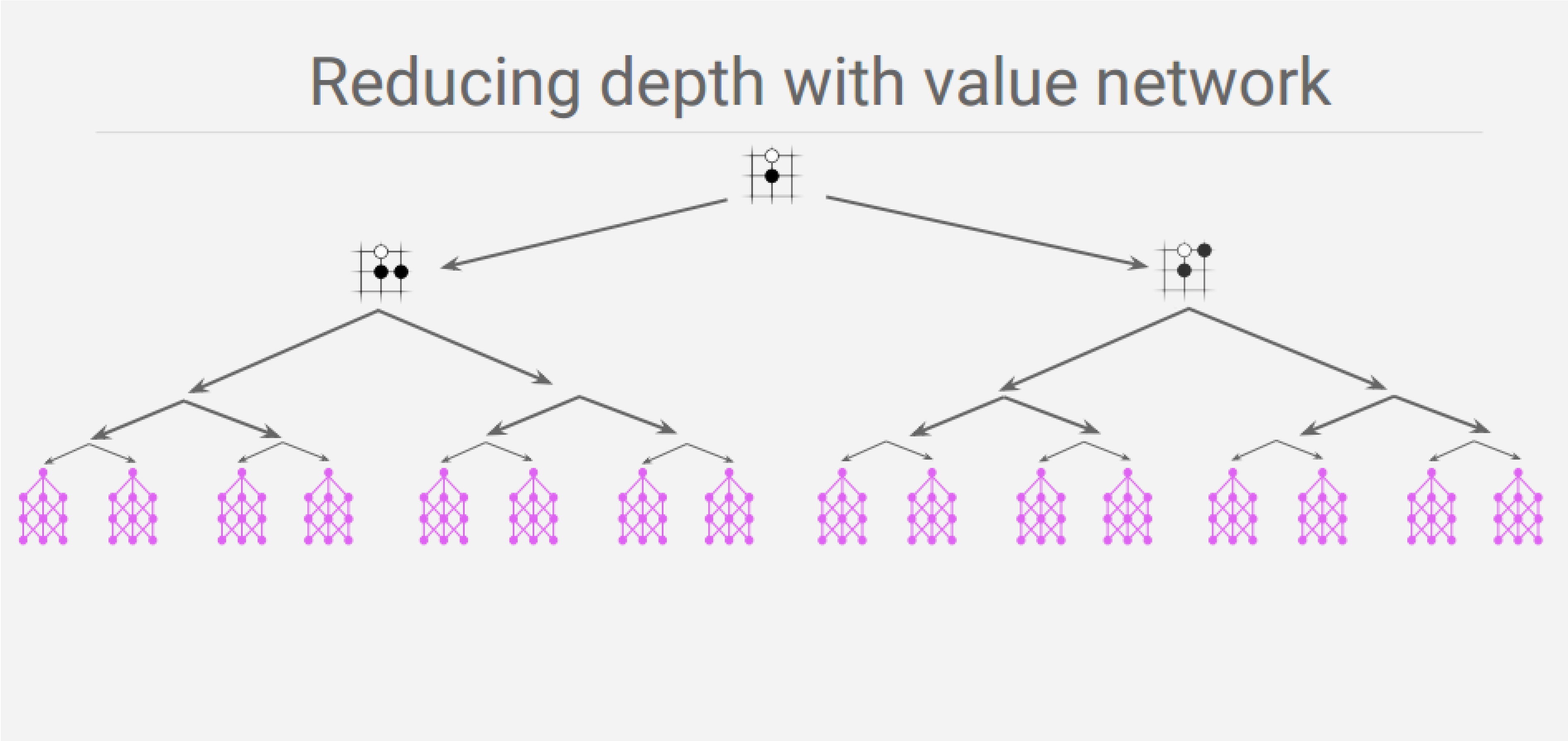




Policy network

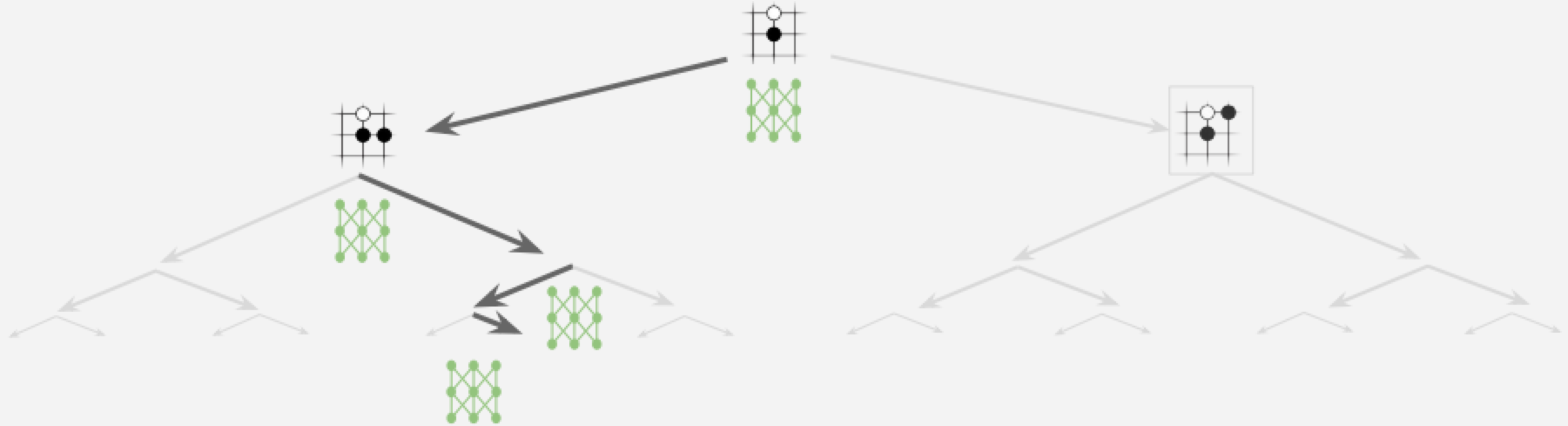


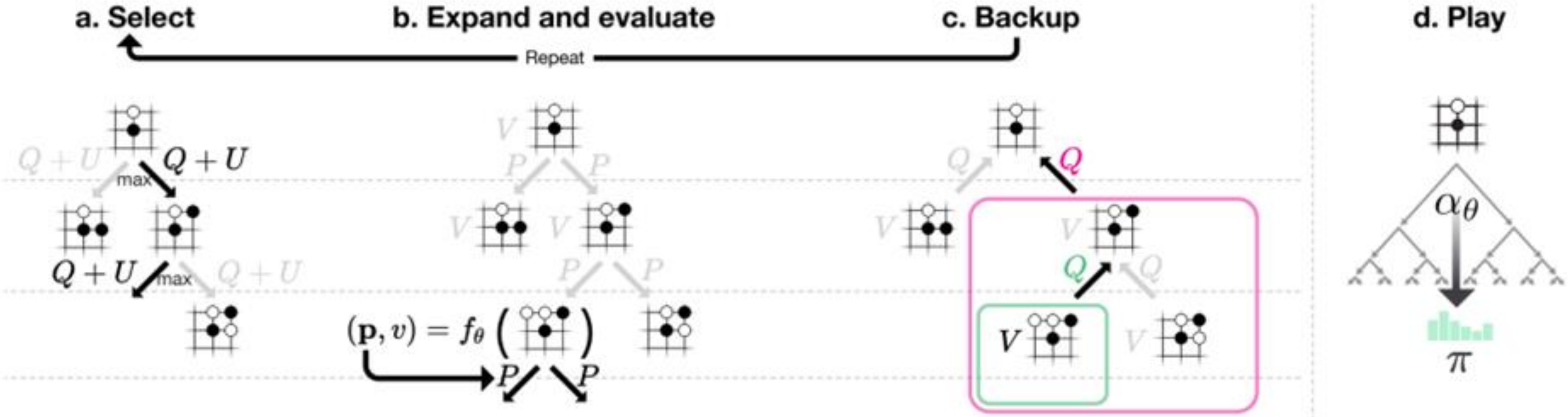




Reducing depth with value network

Reducing breadth with policy network





a. Select

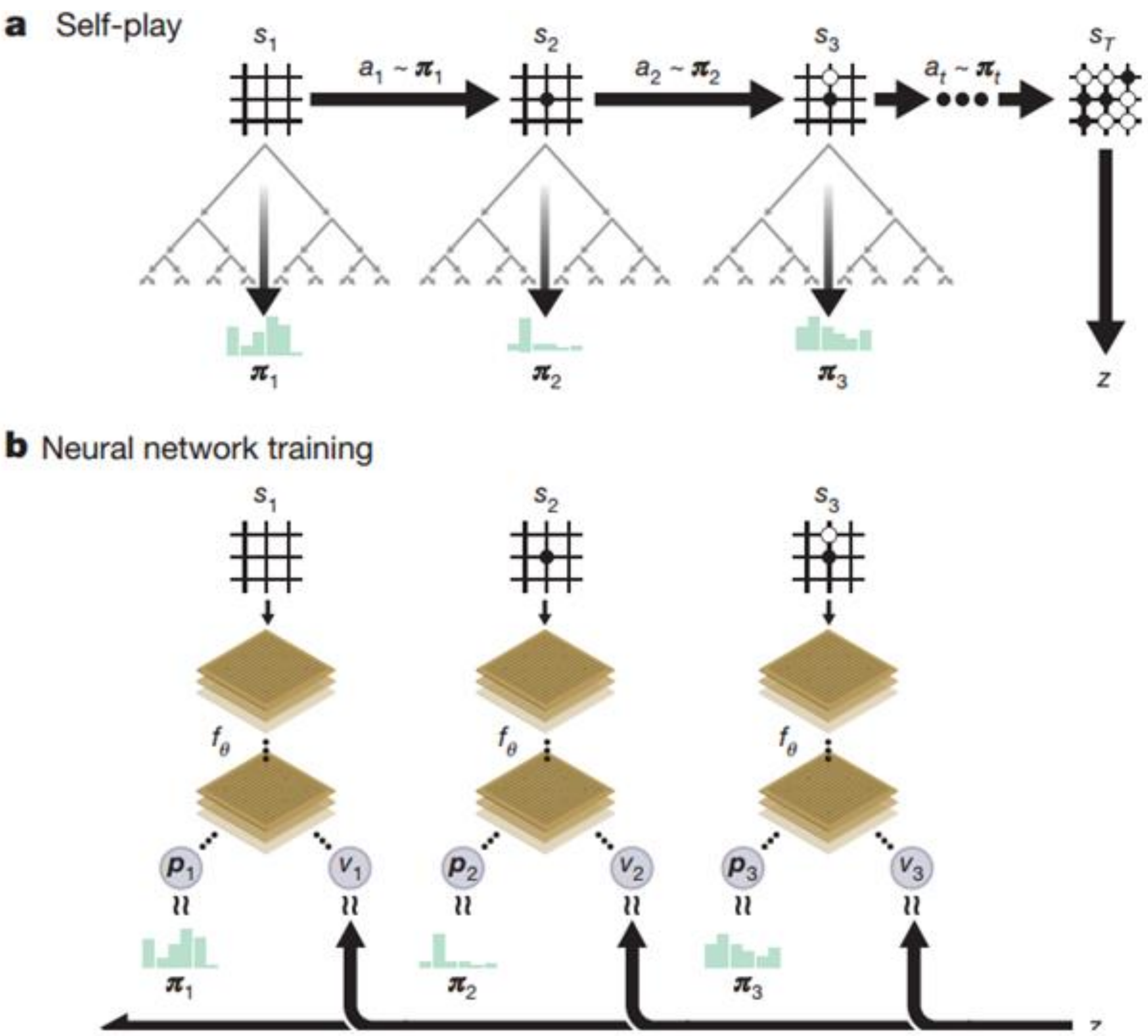
b. Expand and evaluate

c. Backup

d. Play

AlphaGo Zero

- No human data besides rules of the game
- Value and policy are trained on self play instead of human data
- Trained on 4 TPUs for 70 days
    - compared to tens of thousands of TPUs for Gemini

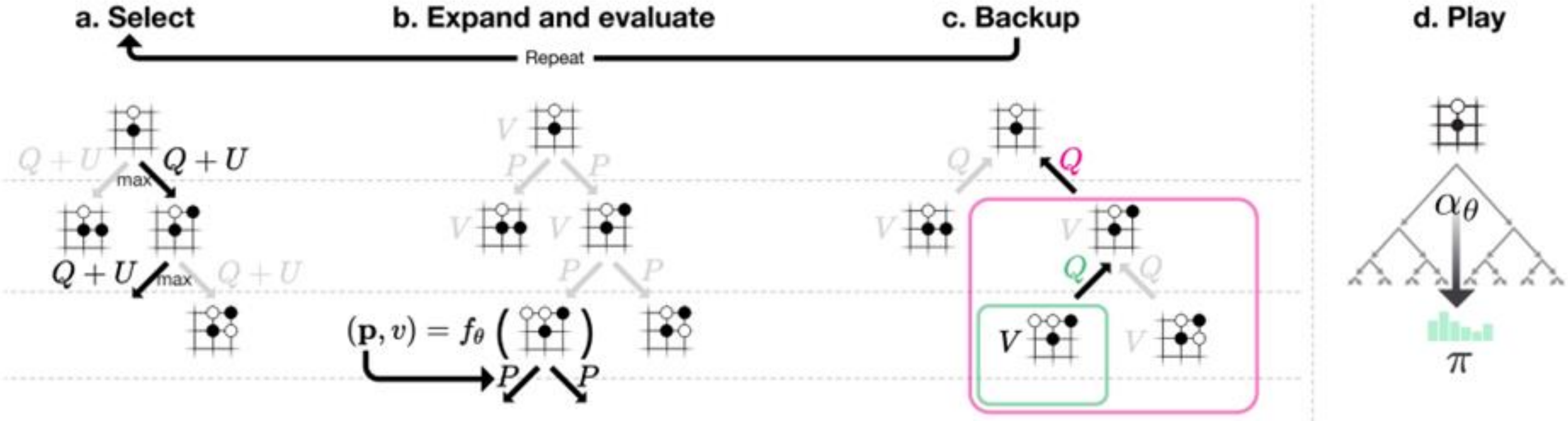


a Self-play
s_1
$a_1 \sim \pi_1$
s_2
$a_2 \sim \pi_2$
s_3
$a_t \sim \pi_t$
s_T
π_1
π_2
π_3
z
b Neural network training
s_1
s_2
s_3
f_θ
f_θ
f_θ
p_1
v_1
p_2
v_2
p_3
v_3
π_1
π_2
π_3
z

Neural Network Loss

$$(\boldsymbol{p}, v) = f_\theta(s) \quad \text{and} \quad l = (z - v)^2 - \boldsymbol{\pi}^{\mathrm{T}} \log \boldsymbol{p} + c\|\theta\|^2$$

Search Algorithm



Search Algorithm

- Each node s in the search tree contains edges (s, a) for all legal actions.
- Each edge stores a set of statistics, {N(s, a), W(s, a), Q(s, a), P(s, a)}
    - N: number of visits to that edge
    - W: Total value
    - Q: Average value
    - P: Policy output

$$a_t = \operatorname{argmax}(Q(s_t, a) + U(s_t, a))$$

$$U(s,a) = c_{\text{puct}} P(s,a) \frac{\sqrt{\sum_b N(s,b)}}{1 + N(s,a)}$$

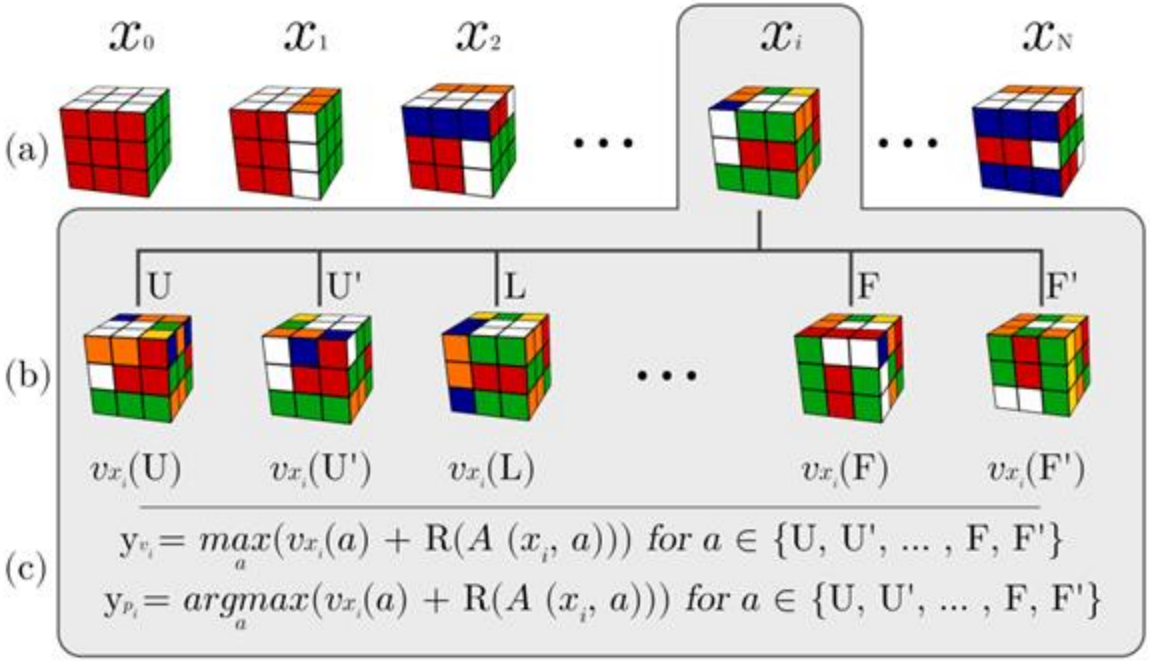
Expand and Evaluate

- When we reach a leaf node, we run the state through the neural network to get a value estimate and policy estimate
- Each edge (N, W, Q) is initialized to 0
- Backup value

Backup

- We update N, W, Q with the value that the neural network proposes
- $N(s, a)=N(s, a)+1$
- $W(s,a)= W(s,a) + v$
- $Q(s, a) = W(s, a)/N(s, a)$

Single-Agent Games


x_0
x_1
x_2
x_i
x_N
(a)
• • •
• • •
U
U'
L
F
F'
(b)
• • •
$v_{x_i}(\mathrm{U})$
$v_{x_i}(\mathrm{U'})$
$v_{x_i}(\mathrm{L})$
$v_{x_i}(\mathrm{F})$
$v_{x_i}(\mathrm{F'})$
(c)
$y_{v_i} = \max_a(v_{x_i}(a) + \mathrm{R}(A\ (x_i,\ a)))\ for\ a \in \{\mathrm{U}, \mathrm{U'}, \ldots, \mathrm{F}, \mathrm{F'}\}$
$y_{p_i} = \operatorname{argmax}_a(v_{x_i}(a) + \mathrm{R}(A\ (x_i,\ a)))\ for\ a \in \{\mathrm{U}, \mathrm{U'}, \ldots, \mathrm{F}, \mathrm{F'}\}$

McAleer et al. "Solving the Rubik's cube with approximate policy iteration." *ICLR*. 2018.
Agostinelli et al. "Solving the Rubik’s cube with deep reinforcement learning and search." *Nature Machine Intelligence*. 2019